Servista

Bhavya(202412011) Malhar(202412075) Jayesh(202412012)
Hitiksha(202412024) Bhavika(202412084) Anjali(202412063)
Shashank(202412103) Pratyush(202412076) Akshay(202412125)

Dhirubhai Ambani University

Problem Definition

State the real-world problem addressed by the project. People struggle to find reliable, verified service providers. Existing solutions commonly lack:

- Transparent pricing
- Trust (reviews, identity verification)
- Seamless booking + payment integration

Target users:

- **Customers** — Search, book, pay and review services.
- **Service Providers** — Manage availability, jobs and earnings.
- **Admins** — Monitor bookings, users, revenue and catalog.

Objectives & Requirements

Objectives:

- Provide reliable booking and scheduling.
- Enable secure digital payments.
- Support provider and admin operations with dashboards.

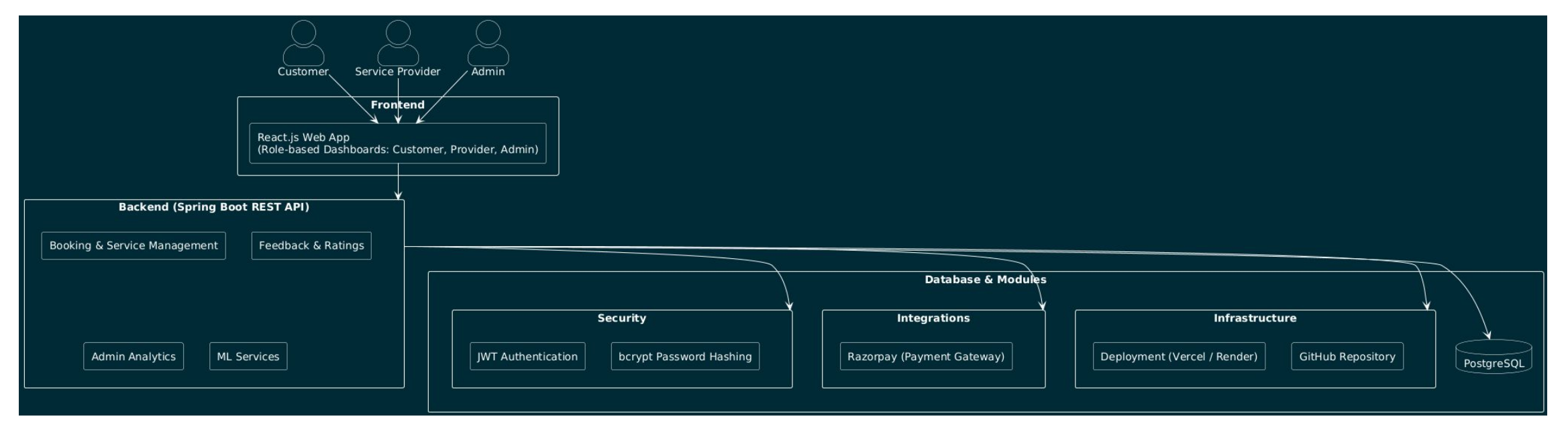
Functional Requirements:

- Role-based authentication (customers, providers, admins).
- Service catalog and provider management.
- Provider availability, booking workflow and ratings/feedback.
- Razorpay payment integration and invoice generation.

Non-Functional Requirements:

- Performance: load time 3s.
- Security: JWT + bcrypt; secure payment flows.
- Scalability: design for growth; target 99.9% availability.

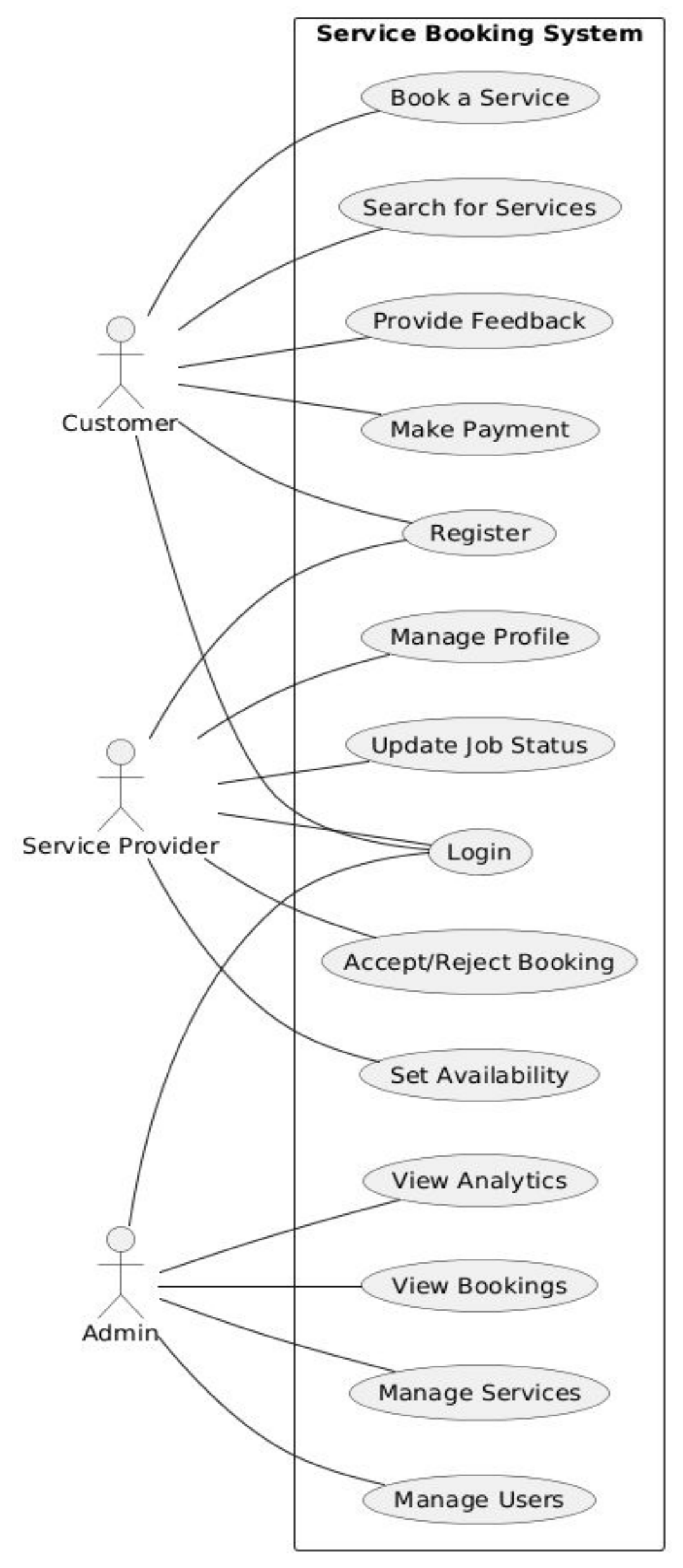
System Architecture

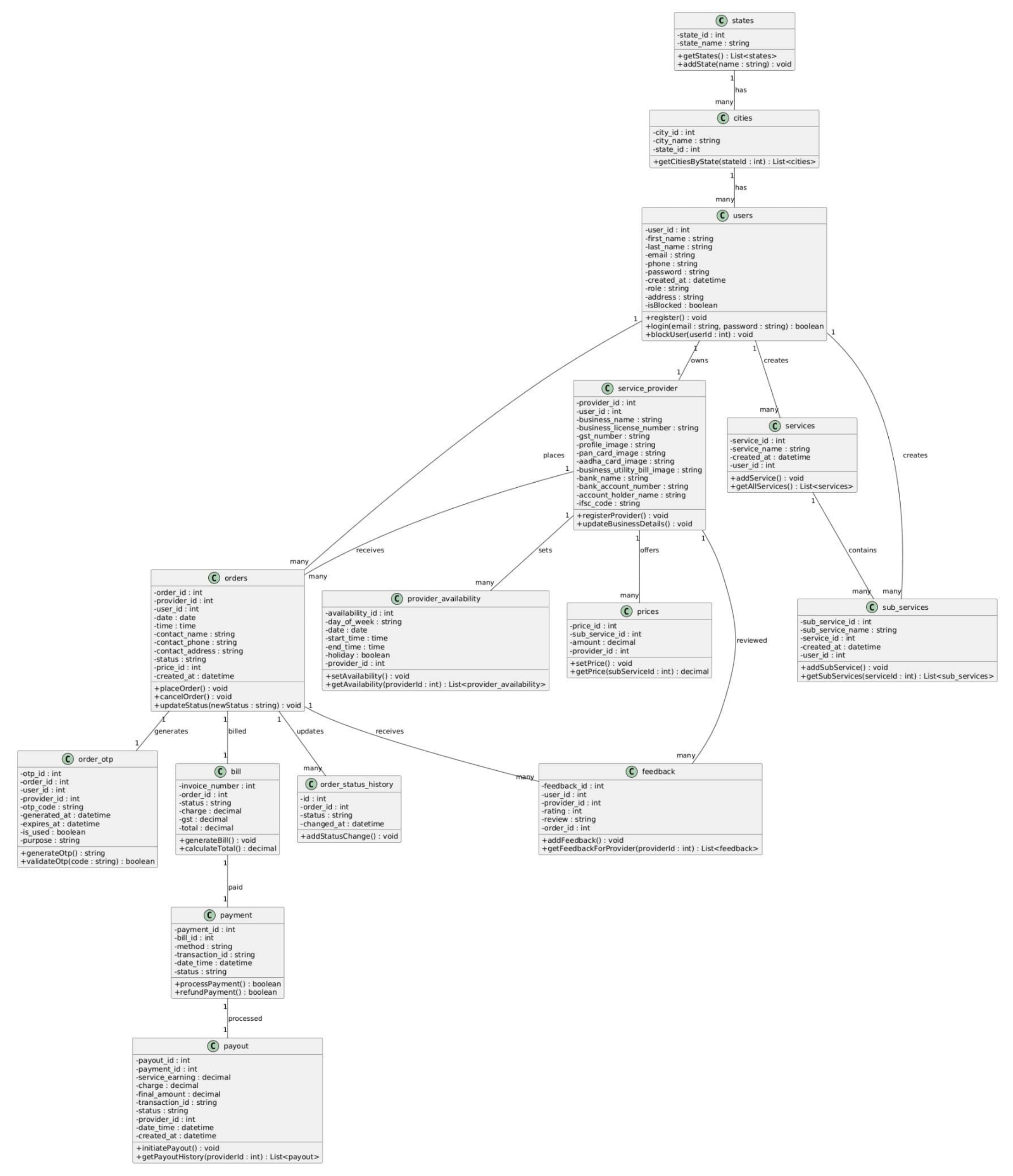


Describe each component:

- React client apps (customers, providers, admins) and Tailwind UI.
- Spring Boot backend with REST APIs and Spring Security.
- PostgreSQL relational database; persistence and transactions.
- Razorpay for checkout, verification and payouts.
- Cloud deployment (Render/Vercel) for frontend/backends.

Design Artifacts





Focus on UML diagrams and clean UI wireframes. Include clear actor/use-case relationships and primary classes/entities.

Implementation Highlights

Describe the important technical decisions, frameworks, patterns, and tools used.

Tech Stack:

- Frontend: React.js, Tailwind, Axios.
- Backend: Spring Boot, Spring Security, JWT authentication.
- Database: PostgreSQL.
- Payments: Razorpay (checkout + server-side verification).

Architectural Patterns:

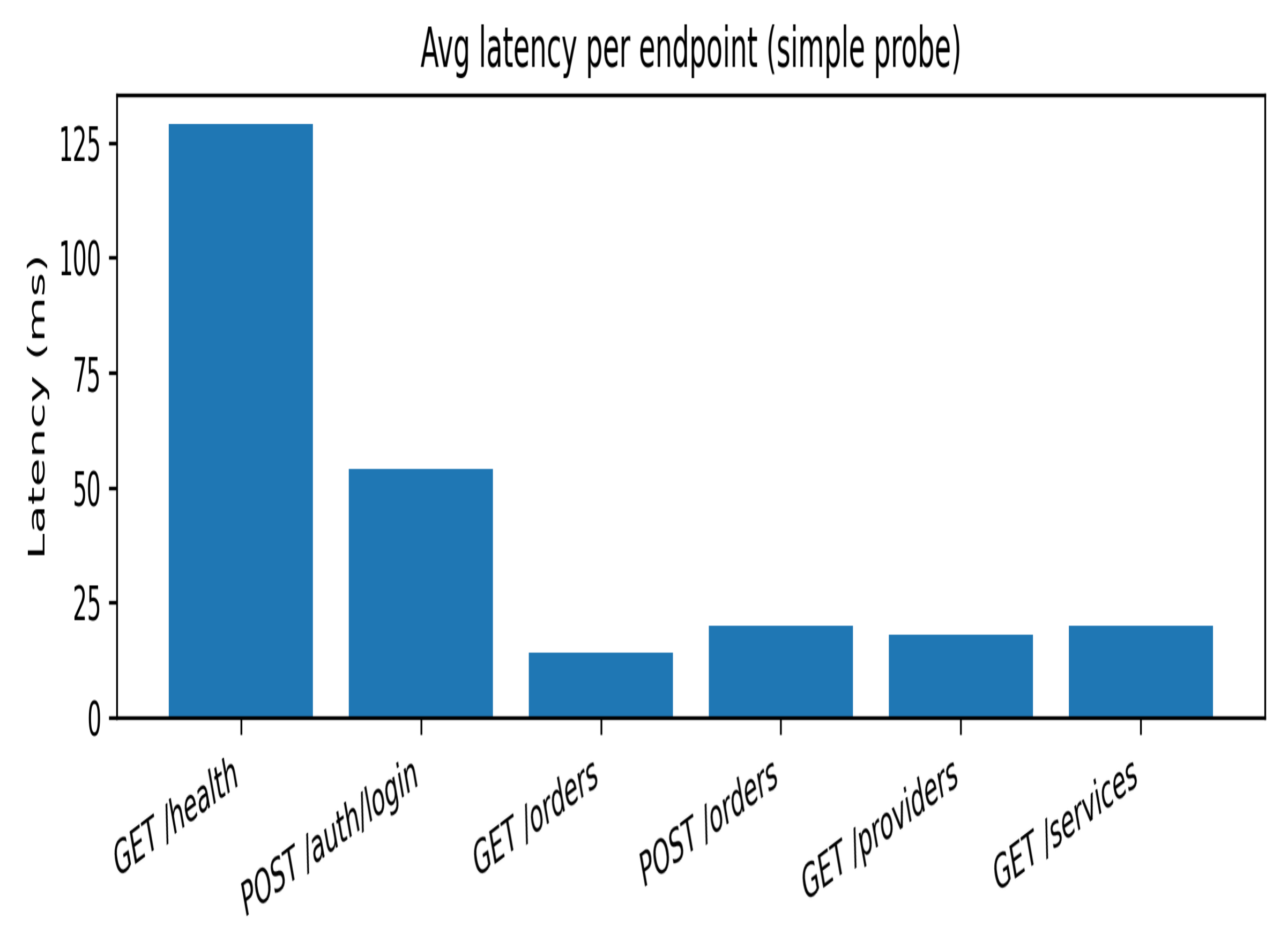
- Monolithic Spring Boot backend (candidate for microservice split later).
- Role-based access control; REST APIs; separation of concerns.

Key Logic / Workflows:

- Booking: search → select → check availability → provider response → payment.
- Payment: Razorpay checkout → verification → invoice generation → provider payout.

Results & Evaluation

Performance metrics, testing and validation highlights:



- Booking workflow: end-to-end tested (successful flows).
- Payment flow: validated using Razorpay and webhook handling.
- Provider dashboard: responsive and stable.

Challenges & Lessons

- Handling delayed/dropped payment webhooks and reconciling transactions.
- Ensuring accurate provider availability and avoiding double-booking.
- Securely storing provider documents and performing identity verification.

Lessons: robust retry/reconciliation for webhooks, defensive availability checks, and encrypting sensitive data at rest.

Future Work

Outline the roadmap:

- Migrate to microservices for scalability and independent deploys.
- Add real-time notifications and location-based provider matching.
- Implement CI/CD, automated tests and expanded analytics.

Acknowledgements

We express our sincere gratitude to **Sir Jayprakash Lalchandani** for his continuous guidance, valuable feedback and support throughout the development of this project.

Open-Source Libraries Used:

- React.js, React Router, Axios, Tailwind CSS, Lucide Icons
- Spring Boot, Spring Security, Spring Data JPA, Hibernate
- PostgreSQL, Docker
- Razorpay Java SDK

GitHub Video walkthrough Full report

- GitHub : https://github.com/malhar2460/Service-Booking-System
- Video : https://drive.google.com/file/d/1qvjiNqbTngURsRAqIkj3b33m-imHWz7X/view?usp=sharing
- Report : https://github.com/malhar2460/Service-Booking-System/blob/main/docs/Group$_1$0*.pdf*