RouteLink — RideShare Connect

Knowledge Transfer (KT)

Version: 2.0 (Consolidated)

Prepared by: Dinesh Sura

Environment: Localhost (Backend: http://localhost:8080, Frontend: http://localhost:5173)

Date: October 2025

Table of contents

---

1. Executive summary & objectives

RouteLink (RideShare Connect) is a route-aware carpooling application connecting drivers and riders who travel along similar routes. The platform focuses on: cost reduction, route-aware matching (along-the-way vs point-to-point), trust via verification and ratings, and a light-weight modern SPA UX.

Primary objectives - Drivers can post trips with start/end, polyline, date/time, seats and price. - Riders search along driver routes (polyline overlap scoring) and request seats. - Booking workflow supports request → confirm/decline → confirmation and seat consumption. - JWT-

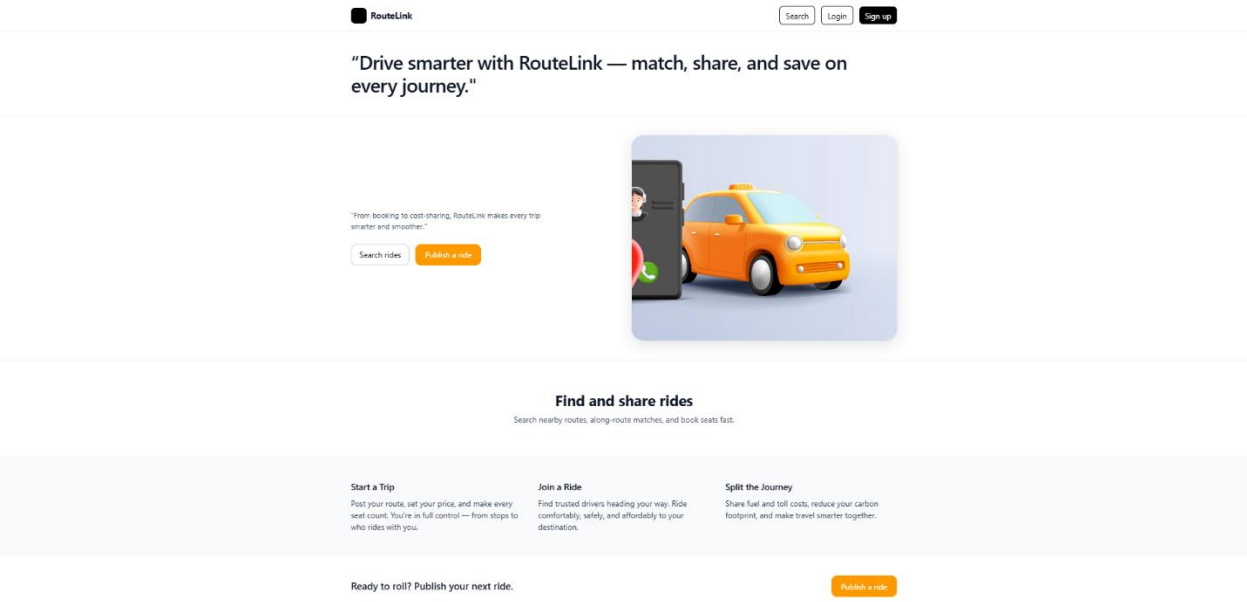
based security with role-based access (DRIVER / RIDER). - Ratings after completed trips with driver rating aggregation.

Success criteria for handover - Developer can run the backend & frontend locally, run core flows via Postman, add simple features, and deploy to a cloud host with environment variables.

2. System overview & user flows

User types - Rider: search trips, request booking, view bookings, rate completed trips. - Driver: create trips, manage bookings (confirm/decline), close/reopen trips, view ratings.

Key flows : 1. Signup/Login (auth) → issues JWT 2. Driver: create trip (polyline & metadata) 3. Rider: search trips using from/to/date → server decodes rider polyline and scores existing open trips 4. Rider: request booking → Booking status: REQUESTED 5. Driver: confirm booking → Booking status: CONFIRMED → trip seatsLeft reduced 6. Trip ends/closed → Rider allowed to post rating → update driver's ratingAvg and ratingCount



3. Technology stack & tools

- Backend: Java 17, Spring Boot 3.x, Spring Web, Spring Data JPA, Spring Security
- Build: Maven
- Database: PostgreSQL (production), H2 (dev/test optional)
- Frontend: React (Vite + TypeScript), React Router, Tailwind CSS, Axios

- Maps & Geo: Google Maps JavaScript API (Places Autocomplete, Directions), polyline encoding/decoding utilities
- Testing: Postman collections (auth, trips, booking, rating)
- Dev tools: IntelliJ/VS Code, pgAdmin, Git/GitHub

4. High-level architecture

- Client (SPA): Renders UI, obtains Google polyline for a proposed route (directions), sends route to backend when creating/searching trips. Stores JWT in localStorage; adds Authorization header to API calls.
- Backend (REST): Controllers → Services (business logic, transactions) → Repositories (JPA) → Database.
- Integration: Google Maps used primarily on frontend for user convenience; backend stores polylines and decodes/compares them for matching.

Diagram (logical)

```
[React SPA] <---HTTP---> [Spring Boot API] <---JPA---> [PostgreSQL]
                    |
                    +---> Google Maps (Directions/Places) (via frontend + optional backend calls)
```

5. Backend: packages, controllers, services, repositories

Package structure - com.routelink.auth - com.routelink.user - com.routelink.trip - com.routelink.booking - com.routelink.rating - com.routelink.security

Principles - Controllers: validation (@Valid), mapping DTOs, HTTP statuses - Services: transactional, enforce business rules (seat checks, role guards) - Repositories: Spring Data JPA interfaces with expressive methods (findOpenTripsByDate, findByTripIdAndStatus, etc.) - DTOs: shape API exposure separate from entities

Sample controller (trip creation)

```
@RestController
@RequestMapping("/api/trips")
@RequiredArgsConstructor
public class TripController {
  private final TripService trips;

  @PostMapping
  @PreAuthorize("hasRole('DRIVER')")
  public ResponseEntity<TripDto> create(@Valid @RequestBody TripRequest req) {
    return ResponseEntity.status(HttpStatus.CREATED).body(trips.create(req));
  }

  @GetMapping("/search")
  public List<TripSearchDto> search(@RequestParam String from,
                                    @RequestParam String to,
                                    @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
LocalDate date,
                                    @RequestParam(defaultValue = "1") int seats) {
    return trips.searchAlongRoute(from, to, date, seats);
  }
}
```

Sample service responsibilities - Create trip: validate input, decode polyline metadata, set seatsLeft = seatsTotal, save trip - Search: decode rider polyline, iterate open trips, compute route similarity score, filter by threshold, sort - Booking: request (check seatsLeft), confirm (consume seats within transaction), cancel (refund seats if required)

Transactional safety - Booking confirm must be transactional to avoid race conditions.

6. Frontend: structure, routing, state, key components

Project layout (recommended)

```
src/
  components/ (Header, Nav, Layout, protected)
  pages/  (Dashboard, CreateTrip, Search, MyBookings, login, signup,      posttrip, Rate)
  services/    (api wrappers using axios)
  context/     (AuthProvider)
  App.tsx
  main.tsx
```

Routing sample

```
const router = createBrowserRouter([
  { path: "/", element: <Dashboard/> },
  { path: "/login", element: <Login/> },
  { path: "/signup", element: <Signup/> },
  { path: "/search", element: <SearchTrips/> },
  { path: "/trip/create", element: <CreateTrip/> },
]);
```

Auth & axios - Store JWT in localStorage under token. - Axios interceptor adds Authorization: Bearer ${token} if present.

Key components - CreateTrip: collects origin/destination (PlacesAutocomplete), calls DirectionsService to retrieve polyline, posts trip to backend - SearchTrips: gets user's origin/destination, calls backend /api/trips/search, lists matches with score and request button - DriverPanel: incoming booking requests, confirm/decline buttons, close/reopen trip actions - MyBookings: shows booking status and rating action after completion

UX notes - Protect driver-only routes via role checks in AuthProvider - Show clear states for booking statuses: REQUESTED, CONFIRMED, DECLINED, CANCELLED, COMPLETED

7. API catalogue (selected endpoints)

All endpoints under /api/* except /api/auth/* which is public.

Auth

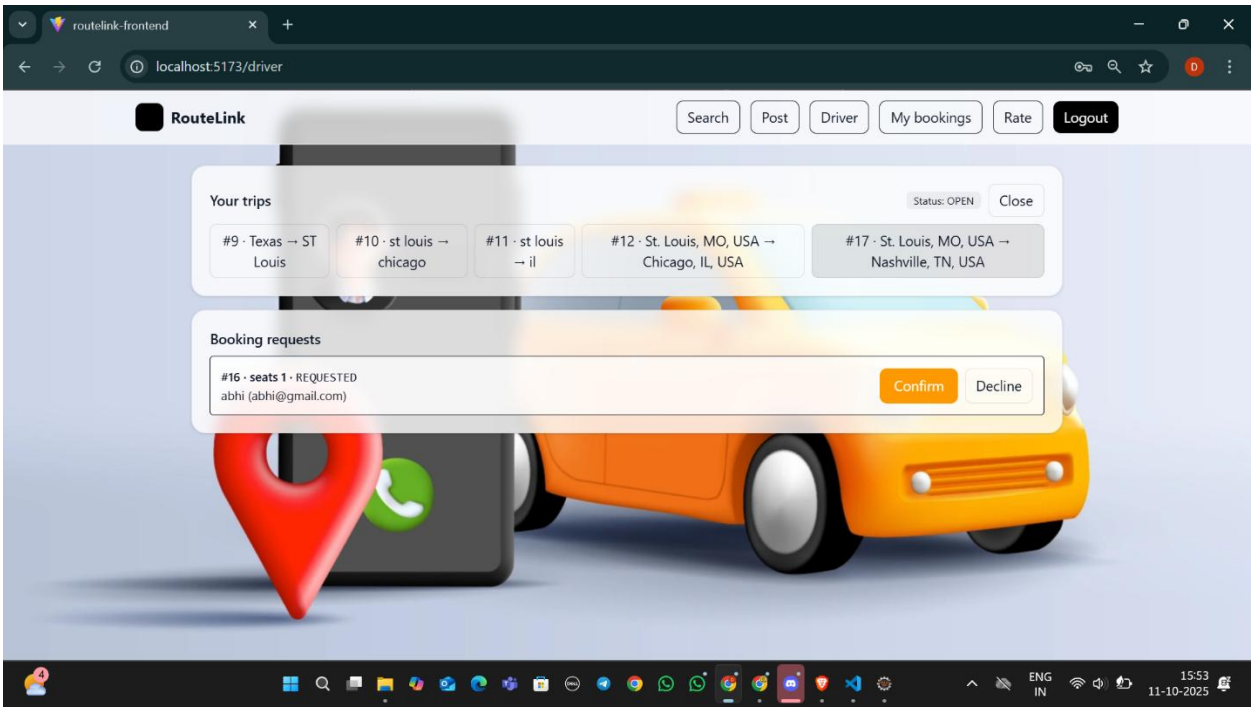
- POST /api/auth/signup — Register user Sample request: { "name":"Bob", "email":"bob@x.com", "password":"Passw0rd!", "role":"RIDER" } Response: { "token": "...", "user": { ... } }
- POST /api/auth/login — Authenticate Request: { "email":"bob@x.com", "password":"Passw0rd!" }

User

- GET /api/users — List users (Admin/dev)
- GET /api/auth/me — Get current user profile

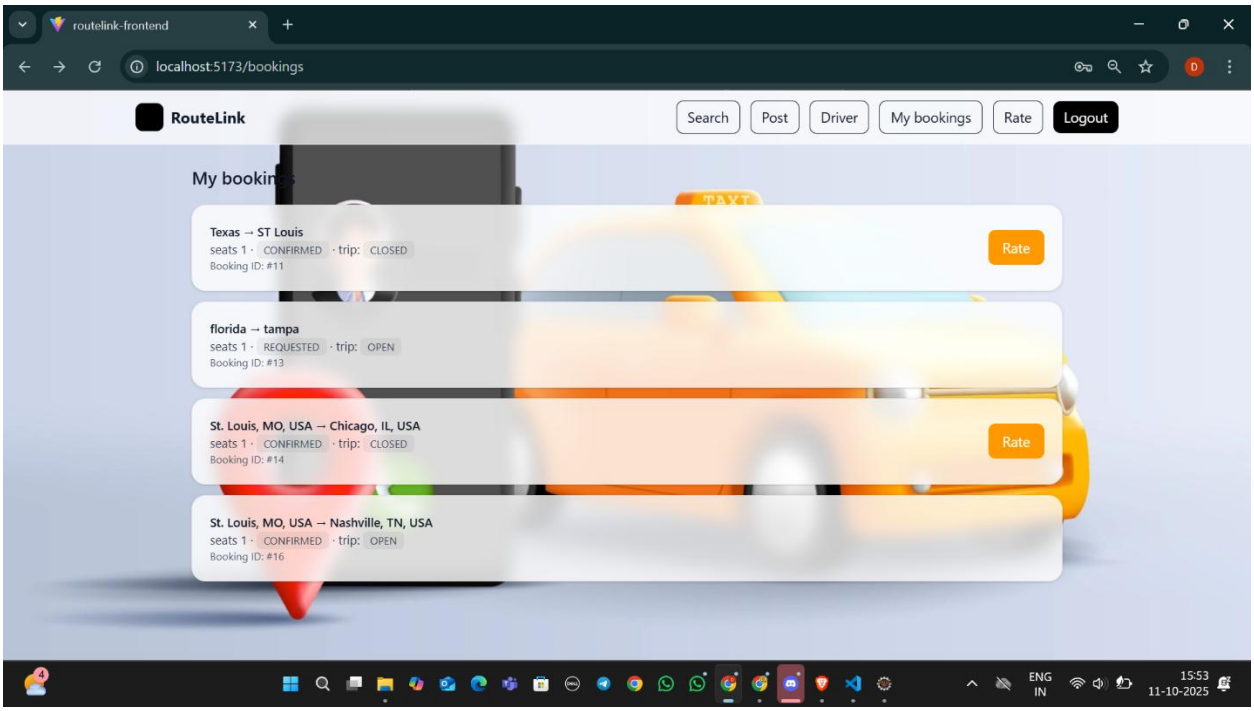
Trips

- POST /api/trips — Create trip (DRIVER) Request: { "start":"St Louis", "end":"Chicago", "rideAt":"2025-10-10T16:00", "seats":3, "price":30, "polyline":"..." }
- GET /api/trips/search — Search along route Example: /api/trips/search?from=St+Louis&to=Chicago&date=2025-10-10&seats=1
- GET /api/trips/{id} — Trip details
- PUT /api/trips/{id}/close — Close trip (DRIVER)
- PUT /api/trips/{id}/reopen — Reopen trip (DRIVER)

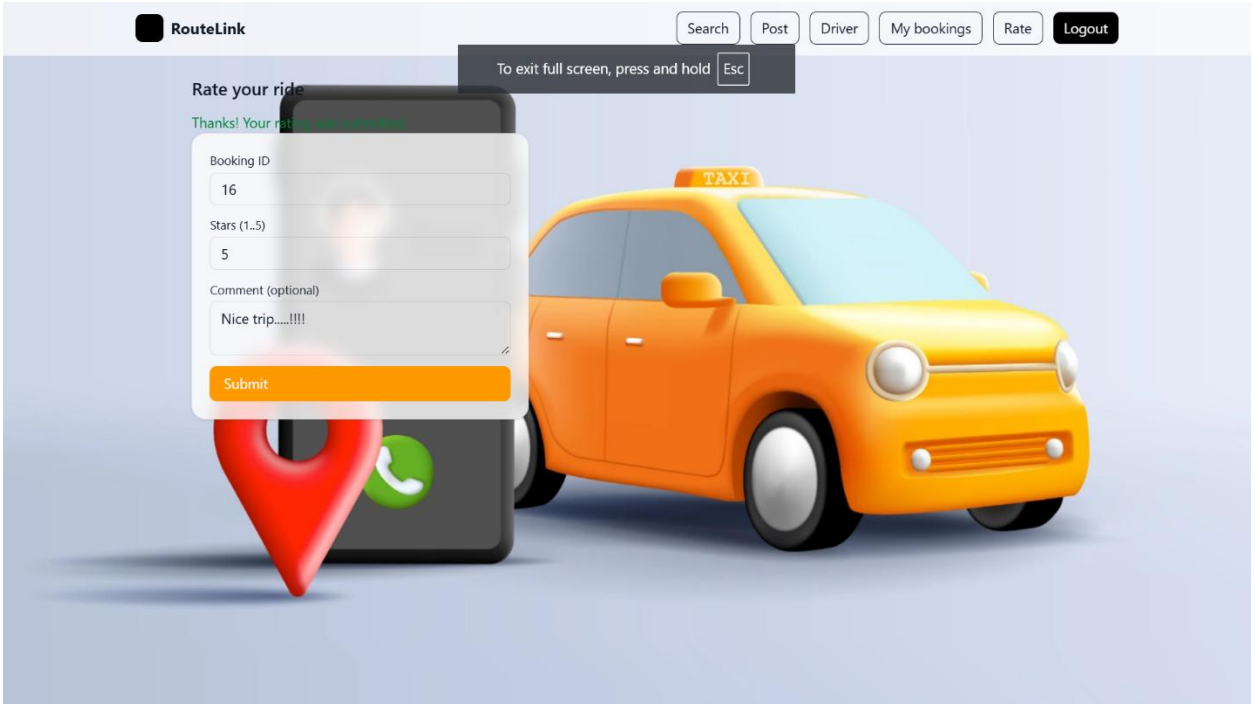


Bookings

- POST /api/bookings — Request booking { "tripId": 12, "seats": 1 }
- PUT /api/bookings/{id}/confirm — Driver confirms booking
- PUT /api/bookings/{id}/cancel — Rider cancels booking
- GET /api/bookings/me — List user’s bookings



Ratings



- POST /api/ratings — Submit rating { "bookingId":16, "stars":5, "comment":"Nice trip" }

Error handling - Use standardized JSON: { "timestamp":"...", "status":400, "error":"Bad Request", "message":"...", "path":"..." }

8. Data model & schema

Conceptual entities (columns abbreviated) - users(id, name, email, passwordHash, role, phone, verified, ratingAvg, ratingCount) - trips(id, driver_id, start_place, end_place, polyline, ride_at (timestamp), seats_total, seats_left, price, status) - bookings(id, trip_id, rider_id, seats, status, requested_at, confirmed_at) - ratings(id, booking_id, stars, comment, created_at)

Indexes - trips(ride_at), trips(status), users(email unique), optional: PostGIS geometry column index if using PostGIS

Example schema DDL (PostgreSQL brief)

```
CREATE TABLE users (...);
CREATE TABLE trips (...);
CREATE TABLE bookings (...);
CREATE TABLE ratings (...);
```

9. Google Maps integration & polyline matching

Where used - Frontend: Places Autocomplete + DirectionsRequest to obtain a polyline for rider's desired route and to help drivers draw routes. - Backend: store trip polyline, decode polyline into lat/lng sequence, compute overlap/score.

Matching strategy (recommended) 1. Obtain polyline for rider search (frontend) and send encoded polyline and bounding info to backend. 2. Backend decodes the polyline to a List<LatLng> (use existing polyline-decode libs). 3. For each open trip on the same date, decode trip polyline and compute similarity: use a sliding-distance metric (e.g., sample points and compute average perpendicular distance between rider and trip path), or compute percentage of rider route covered by driver route. 4. Score and filter by configurable threshold (e.g., score >= 0.6).

Performance - For scale, precompute simplified geometry (Ramer–Douglas–Peucker) and store as geojson; consider PostGIS + ST_DWithin / ST_LineLocatePoint indexing for fast spatial queries.

10. Security (JWT, roles, filters)

- Tokens: HS256 signed JWT stored in backend-secret jwt.secret (env var). Short expiry recommended (e.g., 1h) with refresh token strategy for longer sessions.
- SecurityConfig: disable sessions, authorize /api/auth/** to permitAll, require roles for driver actions, add JwtAuthFilter before UsernamePasswordAuthenticationFilter.
- Token storage: frontend uses localStorage (or HttpOnly cookie in production preferred).
- Common threats & mitigations:
    - CSRF: use stateless tokens, set CORS appropriately
    - XSS: sanitize UI outputs, do not inject raw HTML
    - Brute force: rate-limit /auth/login

11. Booking lifecycle & rating flow

Booking states - REQUESTED → CONFIRMED → COMPLETED - REQUESTED → DECLINED / CANCELLED

Seat accounting - Seats are reserved on confirm. Implement optimistic locking or DB-level transaction: reduce seats_left only inside the confirmation transaction.

Closing a trip - Trips can be closed manually or automatically when seats_left == 0 or rideAt is past - When closing: disallow new booking requests, allow rating submission for associated confirmed bookings

Rating - Only riders with confirmed bookings can rate the trip - Each rating updates driver's ratingAvg and ratingCount (use incremental average formula to avoid a full-table recalculation)

12. Testing strategy & Postman collection notes

- Maintain a Postman collection for common flows: signup/login, create trip, search, request booking, confirm booking, close trip, rate.
- Tests to include positive and negative cases (invalid auth, overbook seats, invalid dates, expired token)
- Unit tests: Service-level tests for booking confirm (concurrency), trip search scoring
- Integration tests: Mock Maps API responses for deterministic results

13. Deployment & environment configuration

Local dev - Backend: mvn spring-boot:run (or ./mvnw spring-boot:run) → http://localhost:8080 - Frontend: npm install && npm run dev → http://localhost:5173

Environment variables (examples)

```
SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/routelink
SPRING_DATASOURCE_USERNAME=postgres
SPRING_DATASOURCE_PASSWORD=password
JWT_SECRET=replace_this_with_secure_value
VITE_GOOGLE_MAPS_KEY=your_maps_key
```

Production considerations - Use HTTPS, secure JWT secret in vault, restrict Maps key by HTTP referrers or IP. - Use managed Postgres (ElephantSQL, AWS RDS) and CI/CD (GitHub Actions) to build & deploy.

Docker (optional) - Provide Dockerfile for backend, docker-compose for backend + postgres + frontend preview

14. Operability & troubleshooting guide

Common issues & fixes - 404 on /api/**: ensure backend is running on configured port and CORS allows frontend origin - H2 vs PostgreSQL: check spring.jpa.hibernate.ddl-auto; for production use validate or none - JWT 401 on protected endpoints: check token expiration, token

signing secret mismatch - Google Maps errors: verify VITE_GOOGLE_MAPS_KEY and HTTP referrers

Debug steps 1. Check backend logs for stack traces 2. Use Postman to verify auth token and API responses 3. Verify DB connections via psql or pgAdmin

15. Future roadmap & scaling

Short-term - Add refresh tokens, web socket for realtime notifications, in-app messaging Medium-term - PostGIS geo-indexing for scalable spatial search, payment gateway integration, driver verification (IDs) Long-term - Containerization (k8s), auto-scaling, analytics dashboard, ML-based route prediction

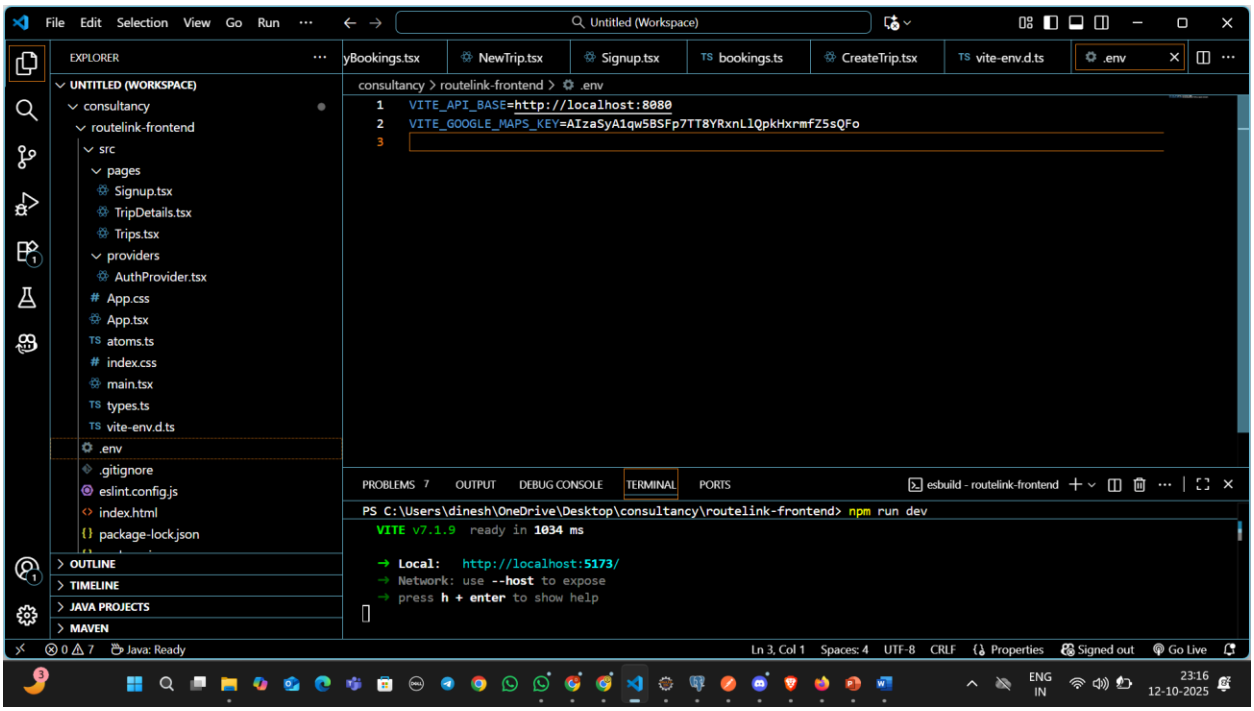
16. Handover checklist

- ☐ GitHub repo URL + access
- ☐ Postman collection exported
- ☐ .env template with example values (without real secrets)
- ☐ Google Maps API key (owner) and list of required APIs (Places, Directions)
- ☐ Deployment notes & required server credentials (if any)
- ☐ Screenshots & PPT used in final presentation

17. Appendix

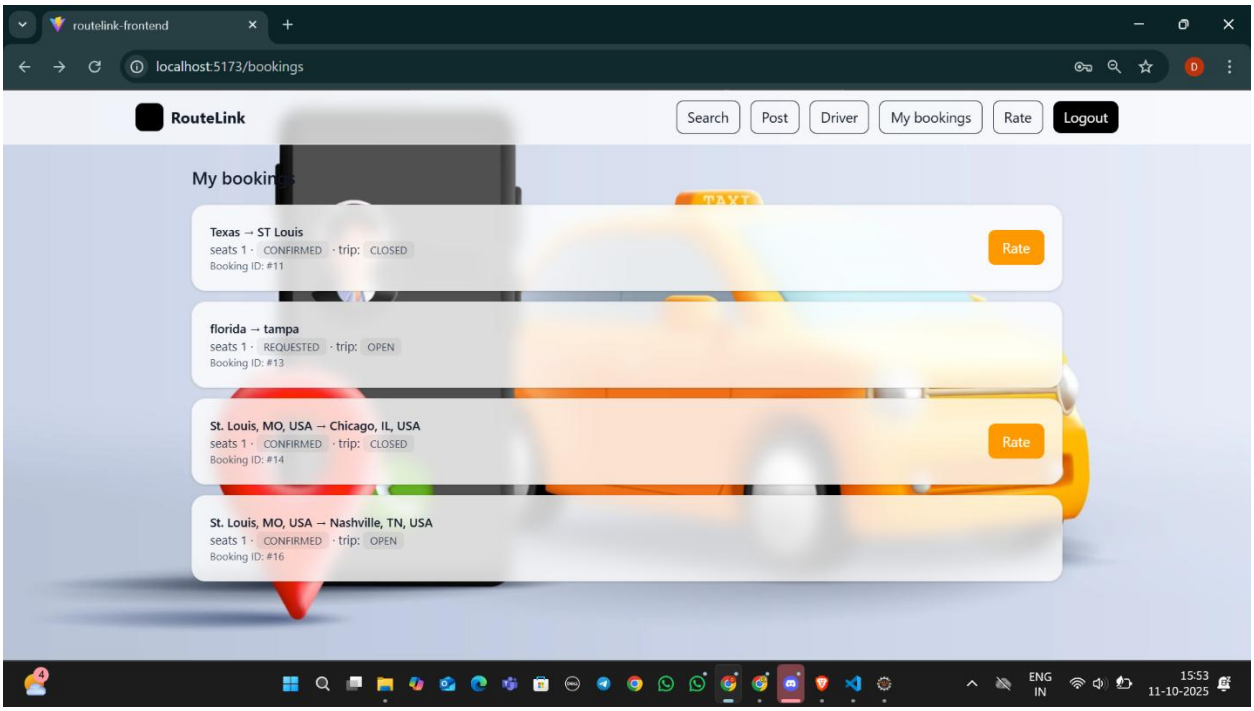
A. Sample env template

```
SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/routelink
SPRING_DATASOURCE_USERNAME=postgres
SPRING_DATASOURCE_PASSWORD=password
JWT_SECRET=replace_with_secret
VITE_GOOGLE_MAPS_KEY= you can replace google api console key
```

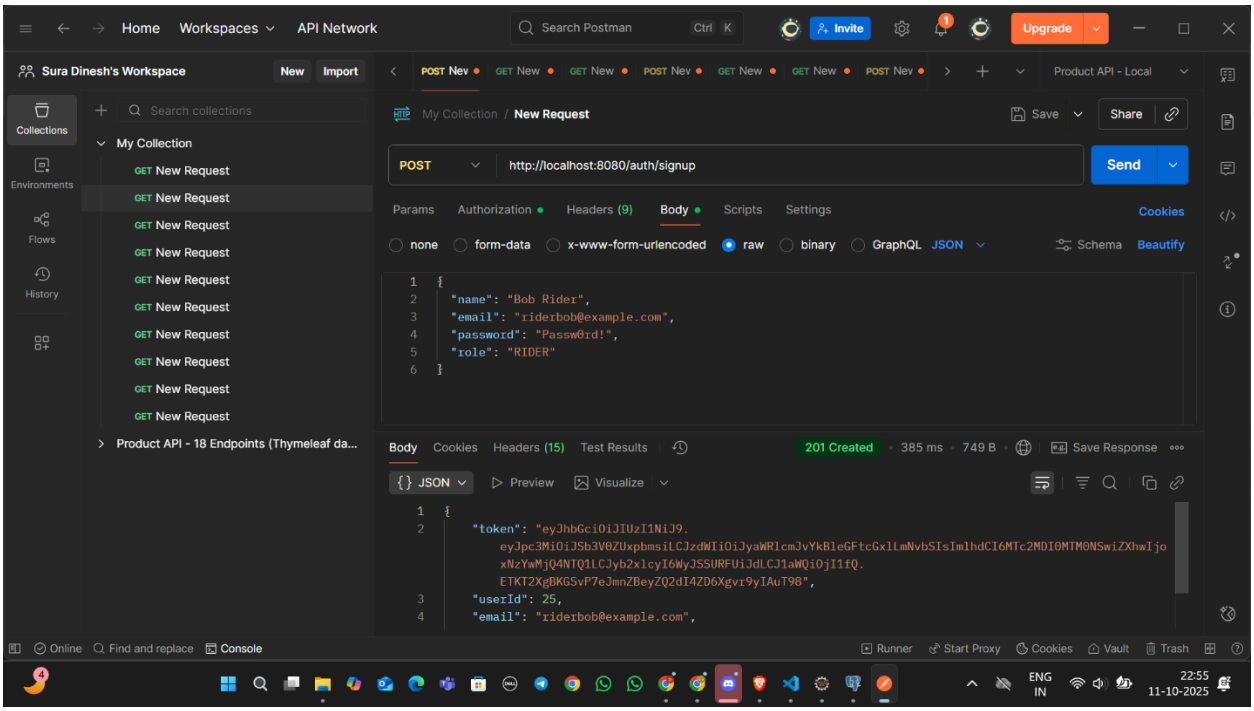


B. Booking confirm pseudo-code (transactional)

```
@Transactional
public void confirmBooking(Long bookingId) {
  Booking b = bookings.findById(bookingId).orElseThrow();
  if (b.isConfirmed()) throw new BadRequestException("Already confirmed");
  Trip t = b.getTrip();
  if (t.getSeatsLeft() < b.getSeats()) throw new BadRequestException("Not enough seats");
  t.setSeatsLeft(t.getSeatsLeft() - b.getSeats());
  b.setStatus(BookingStatus.CONFIRMED);
  bookings.save(b);
  trips.save(t);
}
```



C. Sample Postman test cases to include (brief) - Auth: signup → login → store token - Trip: create (driver) → search (rider) → request booking - Booking: confirm (driver) → verify seats reduced → close trip → rating


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Sura Dinesh's Workspace
New
Import
Collections
Environments
Flows
History
Search collections
My Collection
GET New Request
Product API - 18 Endpoints (Thymeleaf da...
POST Nev
GET New
Product API - Local
My Collection / New Request
Save
Share
POST
http://localhost:8080/auth/signup
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"name": "Bob Rider",
"email": "riderbob@example.com",
"password": "Passw0rd!",
"role": "RIDER"
}
Body
Cookies
Headers (15)
Test Results
201 Created
385 ms
749 B
Save Response
{} JSON
Preview
Visualize
{
"token": "eyJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJSb3V0ZUxpbmsiLCJzdWIiOiJyaWRlcmJvYkBleGFtcGxlLmNvbSIsImlhdCI6MTc2MDI0MTM0NSwiZXhwIjo
xNzYwMjQ4NTQ1LCJyb2xlcyI6WyJSSURFUiJdLCJ1aWQiOjI1fQ.
ETKT2XgBKGSvP7eJmnZBeyZQ2dI4ZD6Xgvr9yIAuT98",
"userId": 25,
"email": "riderbob@example.com",
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash
ENG
IN
22:55
11-10-2025