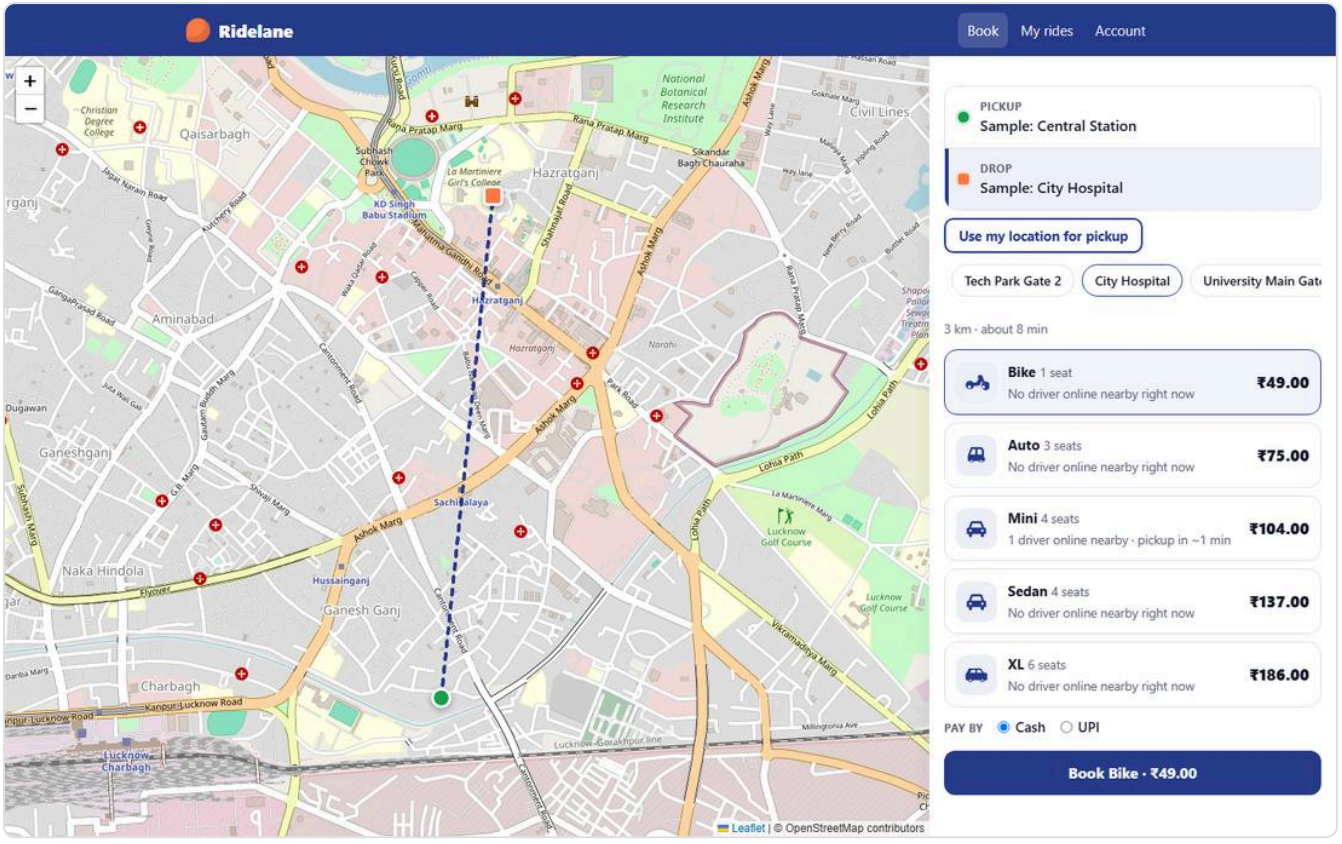
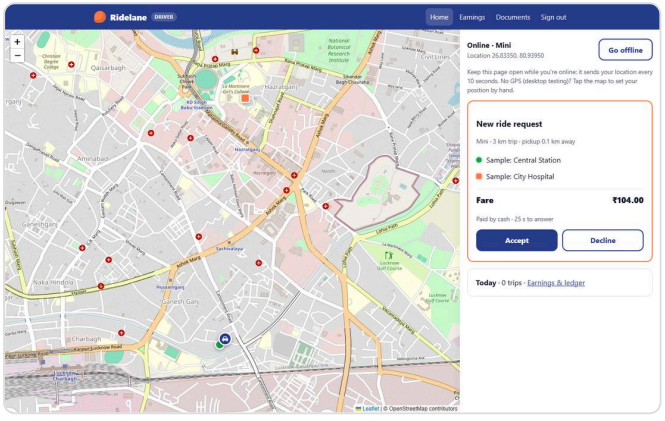
CODE CRAFT · INSTALLATION GUIDE

Ridelane Ride Booking App

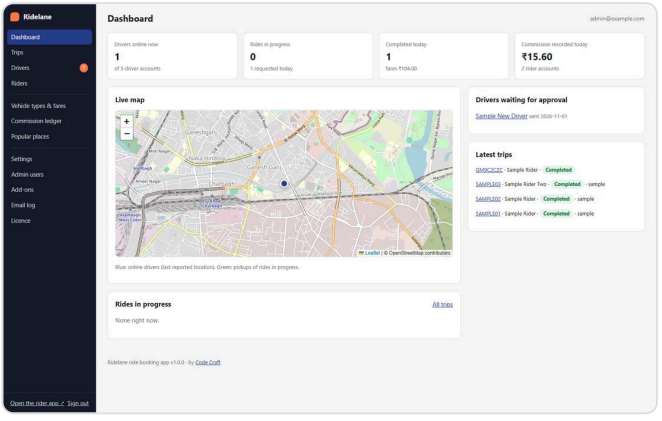
Version 1.0: a rider web app, a driver panel, an admin panel, a REST API and an Expo mobile app, in Node.js + Express + EJS + Prisma. This guide takes you from the download to a live app on Hostinger, cPanel or a VPS.



The rider web app: pickup and drop on the map, and the fare for every vehicle type before booking. Name, colours and logo change in Admin → Settings.



Driver panel: a ride request with 30 seconds to accept.



Admin dashboard: drivers online, rides in progress and a live map.

1. What is in the download

Folder / file	What it is
`server.js`	Start file (port from `PORT`, default 3470). Hostinger, cPanel and PM2 run this.
`src/app.js`	Express app: security headers, sessions, CSRF, routing.
`src/lib/`	Fare formula (`fare.js`), distances and GPS filtering (`geo.js`), matching and ride steps (`rides.js`), accounts, uploads, settings, licence, mail, add-ons.
`src/routes/`	`web.js` rider app, `partner.js` driver panel (/driver), `admin.js` admin (/admin), `api.js` REST API (/api/v1).
`views/`, `public/`	EJS templates, CSS and the Leaflet page scripts.
`prisma/`	Database schema and the sample-data script (`seed.js`).
`mobile/`	Expo (React Native) app with rider and driver modes. See `mobile/README.md`.
`addons/`	Folder for your own add-ons (hooks: ride.requested, ride.statusChanged, ride.completed, app.hourly). None are included.
`test/`	`unit.test.js` and `api.test.js` (run with `npm test`).
`README.md`, `API.md`	Reference documentation: features, money rules, security, every API endpoint.

Requirements

- Node.js 18.18 or newer (20 or 22 LTS recommended).
- Database: nothing extra for SQLite (default). For MySQL 8 or PostgreSQL 13+, an empty database and a user.
- A domain with https for the live app, and your own map tile provider for real traffic (see section 6).
- For the mobile app: Node.js, Expo Go or an emulator to try it, and EAS plus your own Google Play / Apple developer accounts to publish.

2. Run it on your computer

1. Unzip the download and open a terminal in the folder.
2. Install and set up. `npm run setup` creates `.env` with a random `APP_SECRET`, the tables and the sample data.

```
npm install
npm run setup
npm start
```

- Rider web app: `http://localhost:3470/`
- Driver panel: `http://localhost:3470/driver`
- Admin: `http://localhost:3470/admin`

`npm run setup -- --reseed` wipes everything and loads the sample data again. `npm run setup -- --no-seed` creates empty tables; the first visit to `/admin` then runs a setup wizard that creates your admin account and asks for your base currency.

If port 3470 is busy the app says so and stops: change `PORT` in `.env`.

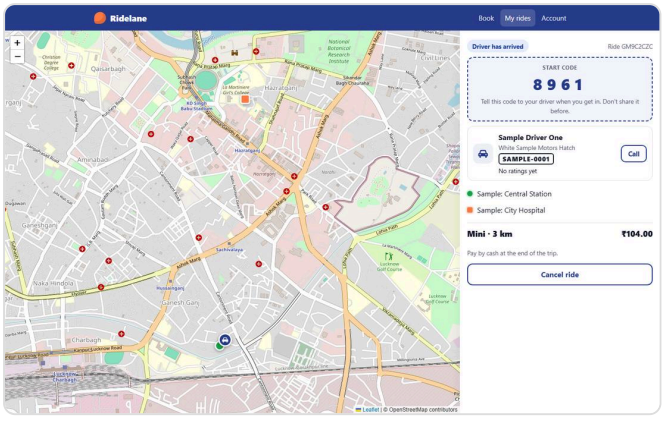
Sample logins (sample data only)

Role	Email	Password
Admin (/admin)	`admin@example.com`	`Demo-Admin-2026!`
Rider	`rider@example.com`, `rider2@example.com`	`Demo-Rider-2026!`
Driver, approved (Mini, Mini, Auto, Bike)	`driver@example.com` … `driver4@example.com`	`Demo-Driver-2026!`
Driver waiting for approval (Sedan)	`newdriver@example.com`	`Demo-Driver-2026!`

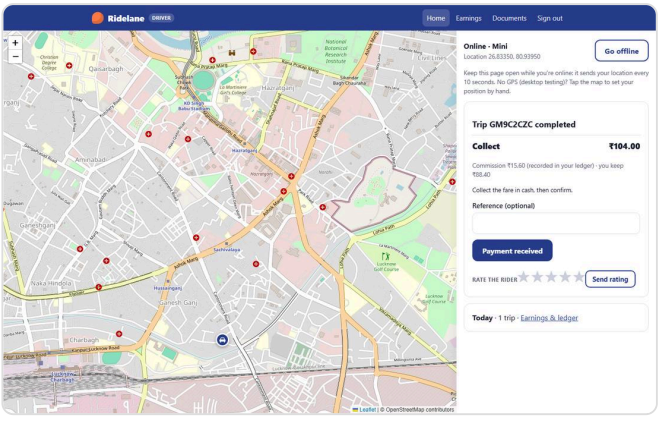
> These accounts are flagged as sample data. Change their passwords or delete them before you go live. The sample city uses place names like "Sample: Central Station" on a real map; the seed creates three completed sample trips and no ratings.

3. Try a full trip

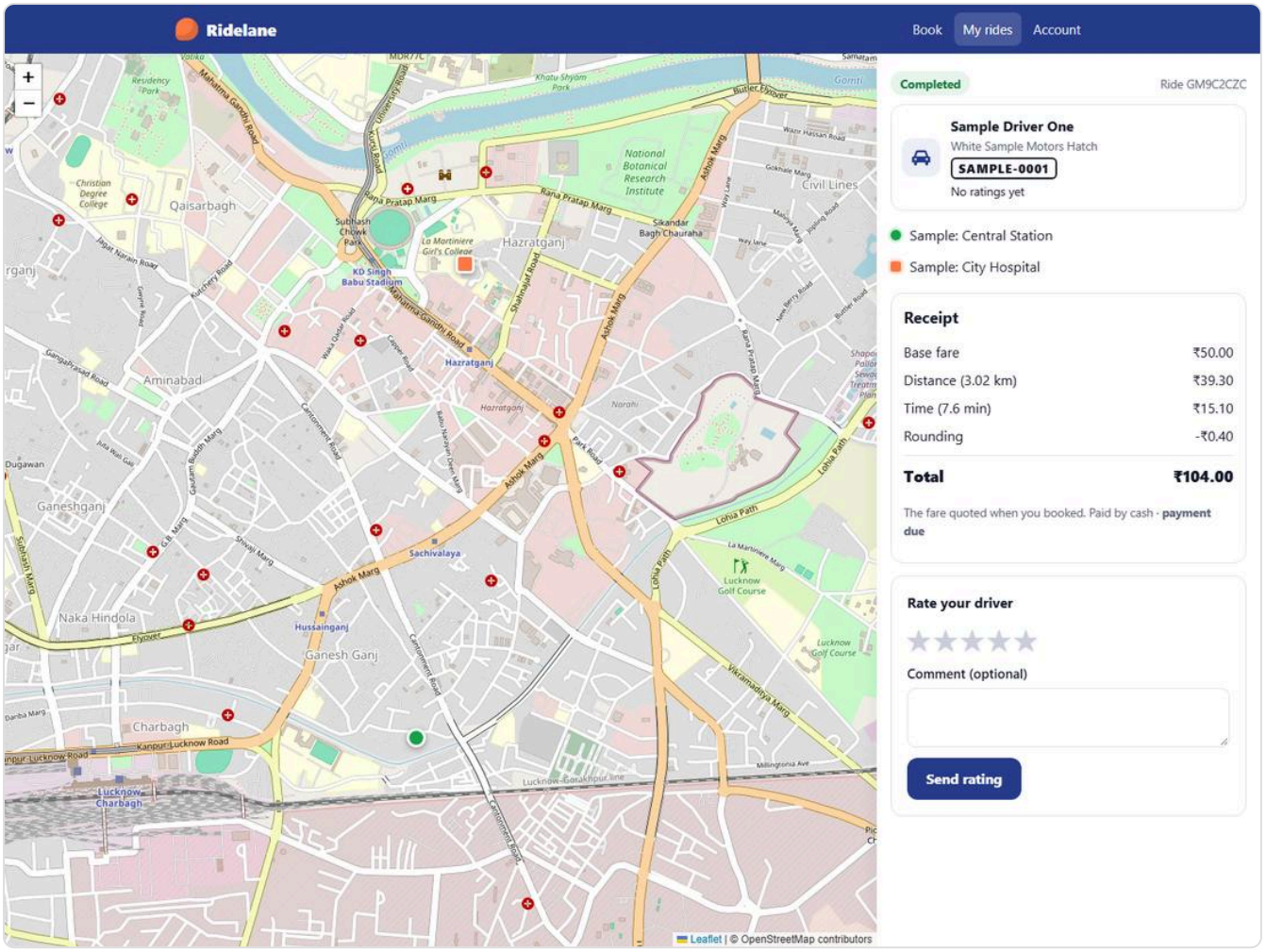
1. In a private window, sign in as `driver@example.com` at `/driver`. Allow location, or tap the map near Central Station, and press **Go online**.
2. In a normal window, sign in as `rider@example.com`, pick *Central Station → City Hospital*, choose **Mini** and book.
3. The driver panel shows the request with a 30-second countdown: **Accept** → **I have arrived** → type the code shown on the rider's page → **Start trip** → **Complete trip** → **Payment received** → rate.
4. The rider's page updates by itself and ends with the receipt and a rating form.
5. In `/admin`, open the trip timeline and the commission ledger.



Rider: driver details and the 4-digit start code.



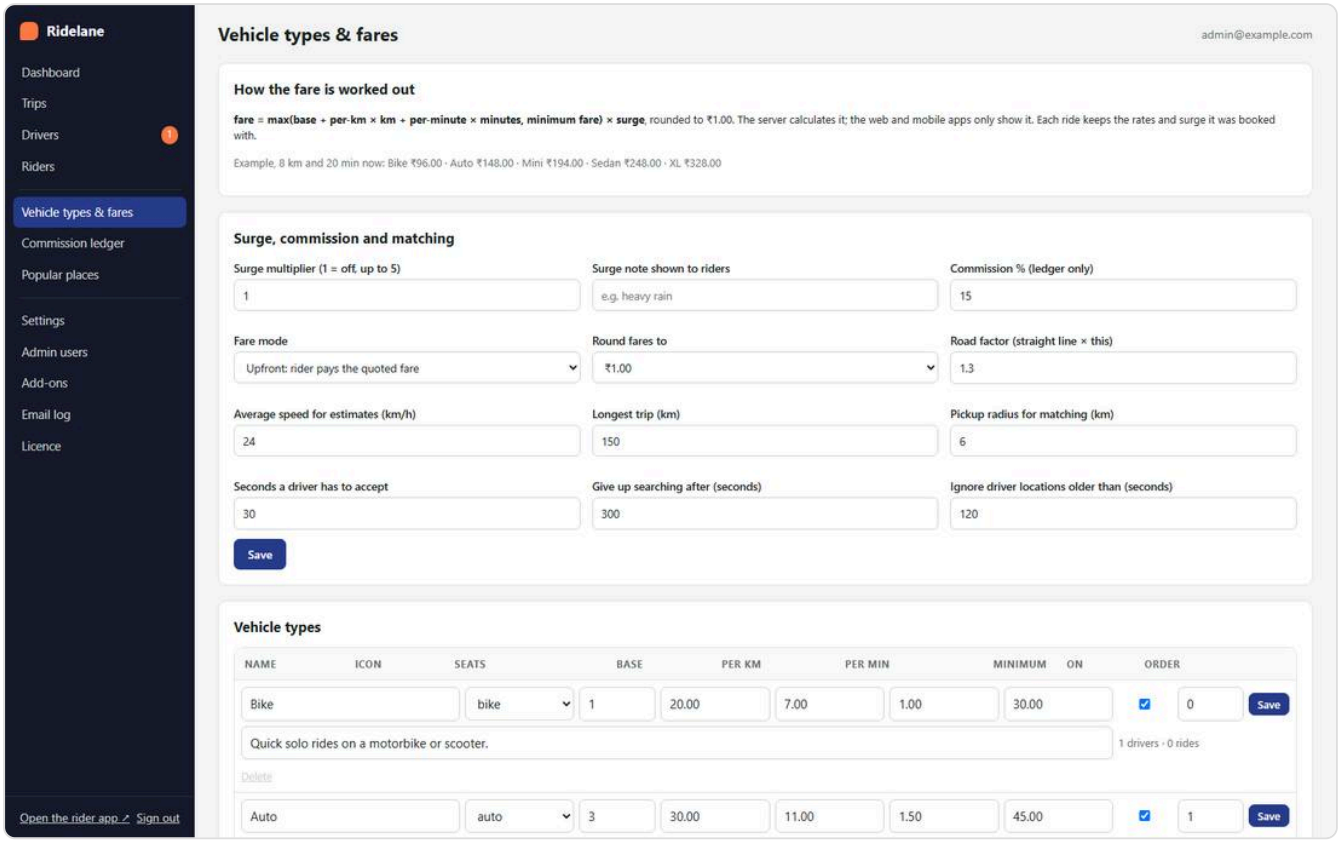
Driver: trip completed, amount to collect and commission.



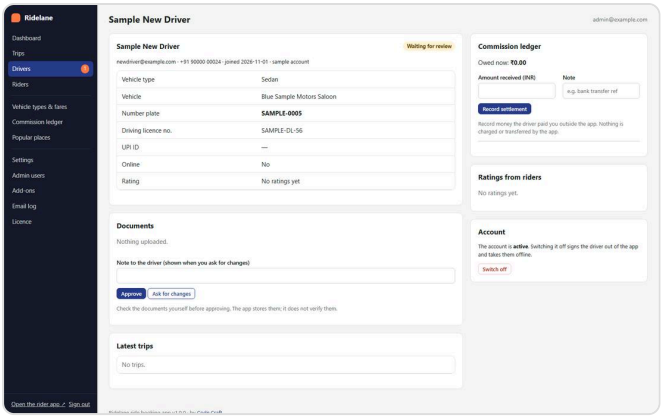
Receipt with the fare breakdown and the driver rating form.

4. Set up your business in the admin

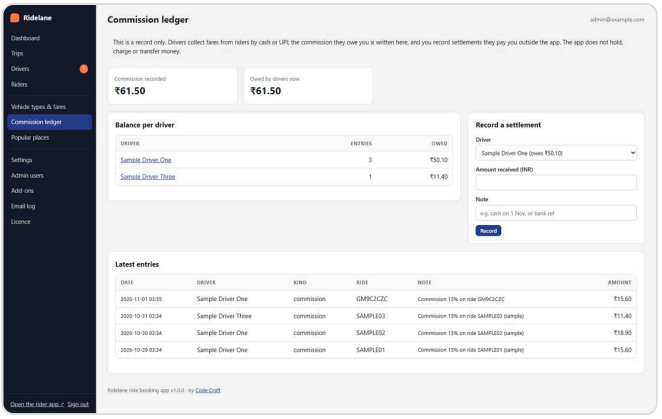
Screen	What to set
Settings	Business name and contact details, logo and colours, social links, map tiles, how riders pay (cash, UPI) and your business UPI ID, SEO, maintenance mode, live chat, analytics, cookie notice, SMTP (with a test email).
Vehicle types & fares	Each vehicle type: name, icon, seats, base fare, per km, per minute, minimum fare, on/off and order. Also surge (1 to 5) and its note, commission %, fare mode (upfront or metered), rounding, road factor, average speed, longest trip, pickup radius and the accept / search / location timeouts.
Popular places	The quick-pick places riders see under the map.
Drivers	Review documents, approve or ask for changes with a note (the driver gets an email), switch accounts off, see ratings and the driver's ledger.
Commission ledger	Balance per driver; record settlements drivers pay you outside the app.
Admin users, Email log, Add-ons, Licence	More admins, every email the app tried to send, your add-ons, licence activation.



Vehicle types & fares: the fare formula, surge, commission and matching settings.



Driver review: documents and the approve button.



Commission ledger and settlements.

5. How fares and money work

```
subtotal = base fare + per-km x km + per-minute x minutes   (per vehicle type)
subtotal = max(subtotal, minimum fare)
total    = subtotal x surge (1 to 5, set by hand), rounded (default: whole rupees)
```

- Fares are calculated on the server only. The web app and the mobile app only show them. The rates and surge are stored on each ride when it is booked.
- Distance and time are estimates: straight-line distance × road factor (default 1.3) at an average speed (default 24 km/h). No routing service is used. To use real road routes, replace `estimateTrip()` in `src/lib/geo.js` with a routing provider.
- Upfront mode (default): the rider pays the quoted fare. Metered mode: the final fare is recalculated from the GPS distance the driver reported and the real trip time.
- Riders pay the driver in cash or by UPI (a `upi://pay` link and QR code). The app does not see or verify the transfer; the driver confirms it.
- Commission is a ledger entry only. There is no wallet and no stored value: in India a wallet that holds money needs an RBI prepaid payment instrument licence.
- No payment gateway is included. The add-on system can host one, but none has been built or tested for this app.

6. Put it online

Hostinger (Node.js Web App) or cPanel (Setup Node.js App)

1. Create a Node.js app (Node 20 or 22) and upload the files, without `node_modules`.
2. Start file: `server.js`. Run `npm install`, then `npm run setup` (or `npm run setup -- --no-seed` for an empty app and the setup wizard).
3. In the environment variables or `.env`: `APP_URL=https://your-domain`, `TRUST_PROXY=true`, a long random `APP_SECRET`. The host sets `PORT`.
4. Restart the app and open `/admin`.

VPS

```
npm install
npm run setup -- --no-seed
npm install -g pm2
pm2 start server.js --name ridelane
pm2 save
```

Put Nginx in front as a reverse proxy to port 3470 and add https with Certbot. Set `TRUST_PROXY=true`.

MySQL or PostgreSQL

Set `DATABASE_URL` in `.env` (`mysql://user:pass@localhost:3306/db` or `postgresql://…`) and run `npm run setup` again: it switches the Prisma provider for you.

Maps

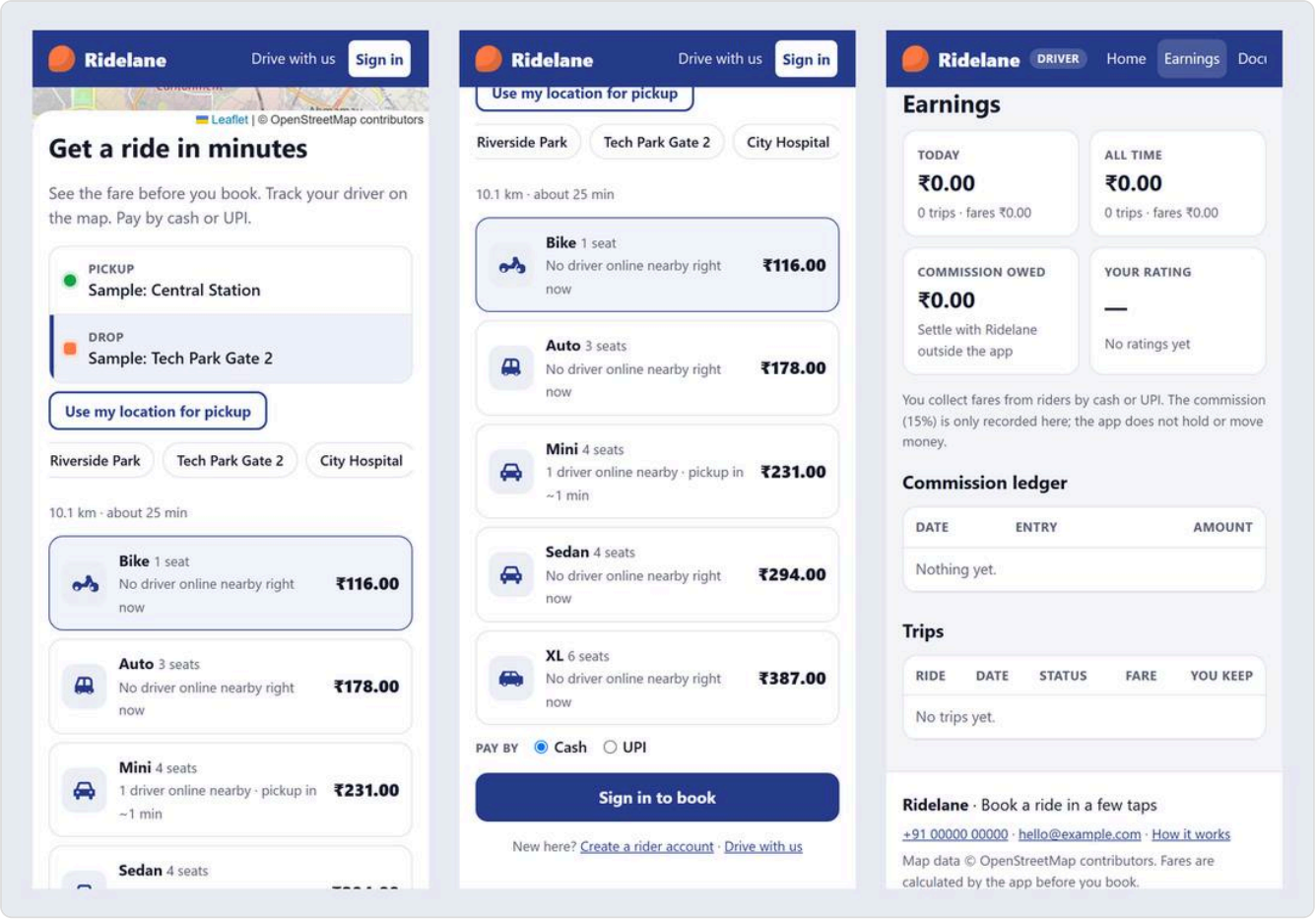
> The public OpenStreetMap tile servers are for light use only. Before going live, set your own tile URL (any `https://…/{z}/{x}/{y}.png`, from a commercial tile service or your own tile server) in Admin → Settings → Map, and keep the attribution it requires.

Licence

Paste your licence key in Admin → Licence on the live domain. Until it is active, new bookings are switched off. One key covers one business on one live domain at a time. The app checks the key with coodecraft.com once a day and keeps a signed token, so it keeps working offline for at most 14 days. You can move the key to another domain up to 3 times a year (deactivate it on the old domain first). Local preview (`localhost`, `127.0.0.1`, `*.localhost` and `*.test`) needs no key.

7. REST API and mobile app

The REST API is at `/api/v1`. Sign in with `POST /auth/login` to get a bearer token (valid 90 days) and send it as `Authorization: Bearer <token>`. Every endpoint, role and error code is in `API.md`.



The web app at phone width: booking, fares and the driver earnings page.

Run the mobile app

```
cd mobile
npm install
cp .env.example .env      # set SERVER_URL to your backend
npx expo start            # w = web, a = Android, i = iOS, or Expo Go
```

From the Android emulator use `http://10.0.2.2:3470`; from a phone on the same Wi-Fi use your computer's LAN IP. Published apps need https. A rider account opens rider mode, a driver account opens driver mode; admins use the web admin.

Store builds

Set the app name, scheme, bundle id and brand colour in `mobile/.env` (see `app.config.ts`), set `SERVER_URL` per profile in `eas.json`, then build with EAS:

```
npx eas-cli@latest init
npx eas-cli@latest build -p android --profile preview
```

Native Android and iOS builds have not been run or tested on a device. The app passes its TypeScript check and web export, and rider and driver modes were tried in the web build. Test your own builds before publishing.

8. Going-live checklist and limits

Before you take real bookings

1. Delete the sample accounts or change their passwords; replace the sample places, vehicle types and rates.
2. Set `APP_URL`, `TRUST_PROXY=true` and a long `APP_SECRET`; serve the app over https.
3. Set your map tile provider, your business UPI ID and commission rate.
4. Set up SMTP and send the test email (Admin → Settings).
5. Activate the licence. Back up `prisma/ride.db` (or your MySQL/PostgreSQL database) and the `storage/kyc` folder regularly.
6. Run `npm test` after any change to the code.

Not included

- No payment gateway, no wallet, no automatic UPI verification.
- No push notifications or SMS: screens update by polling while they are open.
- No turn-by-turn routing, no traffic-aware times, no background location.
- No scheduled or shared rides, promo codes or cancellation fees.
- Document checks are manual: the app stores the files, your staff look at them.
- Running a ride service may need local permits, insurance and driver checks. That is up to you.

Support: **coodecraft.com/contact**. Packages with setup include installation on your hosting by us.