SYSTEM FLOW

Work-Scheduling Platform — System Flow & API

How to Run (quick ref)

- Unzip file
- cd into the folder and run the command below

```
#Command to start the application
docker compose up -d --build

#Command to stop the application
docker compose down -v
```

- UI → http://localhost:3000
- API → http://localhost:8080/api

Overview

A lightweight logistics dispatcher where **drivers** start a shift and receive **loads** (pickup → drop-off), and **admins** create loads and track status. The server auto-dispatches loads to eligible on-shift drivers and tracks progress through stops.

- Tech: Spring Boot (Java 21), React + Vite, Postgres + PostGIS, Docker Compose.
- UIs: `/driver` (driver flow) and `/admin` (admin flow).
- Persistence: Drivers, Shifts, Loads; Flyway migrations (V1–V4).

Driver Flow (Frontend → Backend)

1) Login

- **Endpoint:** `POST /api/drivers/login`

  Body:

  ```
  { "username": "alex" }
  ```

  Responses:

  - `201 Created` + `Location: /api/drivers/{id}` + **DriverDto** (new driver)
  - `200 OK` + **DriverDto** (existing)
  - **DriverDto**:

    ```
    { "id":"UUID", "name":"alex", "onShift":false, "latitude":null, "longitud
    e":null }
    ```

- **Frontend:** Store `driver.id`, route to the Shift screen.

---

2) Shift Management

2a) Start shift → fetch assignment

1. **Start Shift**

   `POST /api/drivers/{driverId}/shift/start`

   Body:

   ```
   { "latitude": 40.01499, "longitude": -105.27055 }
   ```

   Response: `200 OK` (shift opened; may or may not have an assignment yet)

2. **Get Driver Assignment**

   `GET /api/drivers/{driverId}/assignment`

   Responses:

   - `200 OK` + **LoadSummaryDto** (load becomes `RESERVED`, `currentStop="PICKUP"`)
   - `204 No Content` (no suitable load yet)

- **Frontend guidance:**

  While on shift without a load, poll:

  - `GET /api/drivers/{driverId}/state` every 10–20s
  - `GET /api/drivers/{driverId}/assignment` every 20–30s with jitter

2b) Already on shift → show state

- **Endpoint:** `GET /api/drivers/{driverId}/state`

  Response:

```
{
  "driver": {"id":"UUID","name":"alex","onShift":true,"latitude":40.01499,"lo
ngitude":-105.27055},
  "shift":  {"id":"UUID","startedAt":"ISO","startLat":40.01499,"startLng":-10
5.27055},
  "load":   {
    "id":"UUID",
    "status":"RESERVED"|"IN_PROGRESS",
    "currentStop":"PICKUP"|"DROPOFF",
    "pickup":{"lat":..,"lng":..},
    "dropoff":{"lat":..,"lng":..},
    "assignedDriver":{"id":"UUID","name":"alex"}
  }
}
```

- **Frontend:**
  - If `load=null`: show **End shift** + **Get assignment**.

- If `status="RESERVED"`: show **Complete Next Stop (pickup)** + **Reject**.
- If `status="IN_PROGRESS"`: show **Complete Next Stop (drop-off)**.

---

3) Load Progress

3a) Complete next stop (acceptance is implicit)

- **Endpoint:** `POST /api/drivers/{driverId}/complete-next-stop`

  Body: none

  Response:

```
{
 "completed": {
  "loadId": "51a615e6-0bfc-4e39-9b89-7092016308e2",
  "pickupLat": 40.01499,
  "pickupLng": -105.27055,
  "dropoffLat": 39.7392,
  "dropoffLng": -104.9903,
  "status": "IN_PROGRESS",
  "nextStop": "DROPOFF"
 },
 "nextAssignment": null}
```

- Semantics:
  - Completing **PICKUP → IN_PROGRESS**: `nextStop="DROPOFF"`, `nextAssignment=null`.
  - Completing **DROPOFF → COMPLETED**: server **immediately** searches for a new load and returns it as `nextAssignment`.
- **Timeout rule (current impl):**

  If a driver **fails to complete the pickup within 2 minutes**, the system automatically **unassigns** the load and marks the driver off-shift.

> In real deployments, this timeout would be much longer (e.g. 30–60 minutes), but 2 minutes was used for demonstration/testing.

3b) Reject load (only before pickup)

- **Endpoint:** `POST /api/drivers/{driverId}/loads/{loadId}/stops/reject`

  Effect: Unassigns load (→ `AWAITING_DRIVER`) **and ends the driver's shift**.

  Response: `200 OK`

- **Frontend:** `GET /state` shows off-shift.

---

4) End shift (no active load)

- **Endpoint:** `POST /api/drivers/{driverId}/shift/end`

  Precondition: Driver has **no** active load.

  Responses: `200 OK` or `409 Conflict`

- **Frontend:** After success, `GET /state` → off-shift.

Admin Flow

Create Load

- **Endpoint:** `POST /api/loads`

  Body:

```
{
  "pickup":  { "lat": 39.7392, "lng": -104.9903 },
  "dropoff": { "lat": 33.4484, "lng": -112.0740 }
}
```

- **Server behavior:**

  1. Creates load with `AWAITING_DRIVER` and `currentStop="PICKUP"`.

2. **Auto-dispatch** runs immediately: finds nearest on-shift, unassigned driver; sets `RESERVED`. If none available, stays `AWAITING_DRIVER`.

- **Frontend:**
  - After 201, refresh `GET /api/loads` to see new load and assigned driver (if any).
  - Drivers discover new loads via `/state` polling.

Load queries

- **List Loads:** `GET /api/loads?status=AWAITING_DRIVER|RESERVED|IN_PROGRESS|COMPLETED`

  Response: `200 OK` + `LoadSummaryDto[]`
- **Get by Id:** `GET /api/loads/{id}` → `200 OK` or `404 Not Found`
- **(Optional) Drivers list:** `GET /api/drivers` → `200 OK` with `onShift`/last location.

Dispatch Logic

- Triggered on:
  - Driver starts shift.
  - Admin creates load.
  - Driver completes drop-off.
- Strategy: closest driver to pickup (PostGIS KNN).
- Auto-unassign rule: reserved load → if **pickup not completed in 2 minutes**, unassign and end shift.

Known Edge Cases

- End shift with active load → `409 Conflict`.
- Reject after pickup not allowed.
- Refresh/browser close → driver state restored via `GET /state`.
- Double-dispatch avoided via DB transaction & locks.

If I Had More Time (Next Steps)

Backend

- Add **Swagger/OpenAPI docs** for clear endpoint usage.
- Improve **error and exception handling** (currently minimal).
- **Code cleanup**: inline comments, consistent naming, remove debug logs.
- **Testing**: expand unit & integration test coverage with meaningful test cases.
- Authentication & roles for Admin vs Driver.
- Better observability (metrics, traces, health endpoints).
- More advanced dispatch (ETA, priority loads).

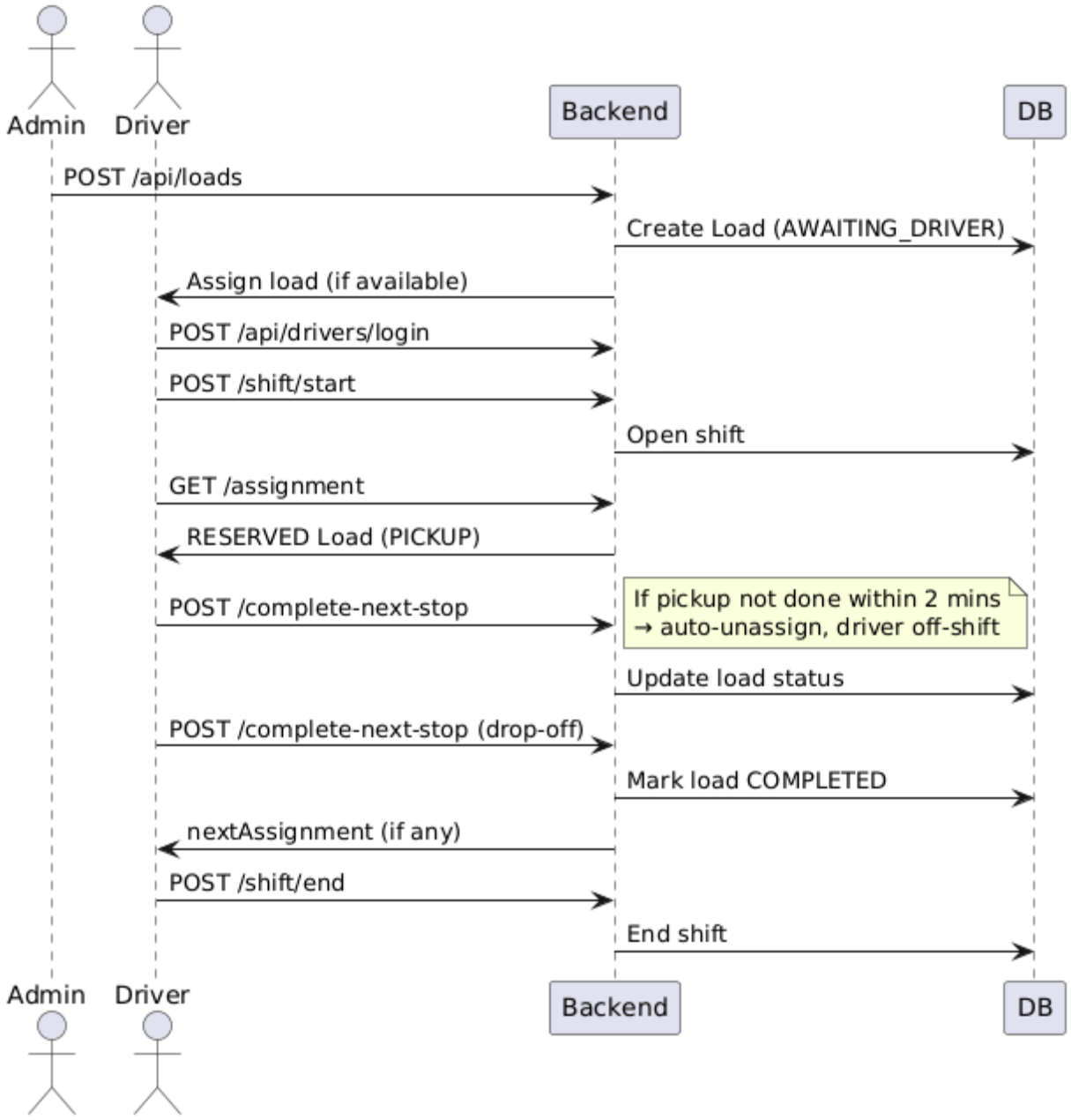
Database

- PostGIS routing functions (`ST_DistanceSphere` with road network data).
- Enforce one active shift per driver with partial indexes.

Frontend

- More map visualization (routes, real-time driver markers).
- State machine to model driver/load lifecycle.
- UI feedback: toast notifications, clearer error states.

Sequence Diagram


Admin
Driver
Backend
DB
POST /api/loads
Create Load (AWAITING_DRIVER)
Assign load (if available)
POST /api/drivers/login
POST /shift/start
Open shift
GET /assignment
RESERVED Load (PICKUP)
POST /complete-next-stop
If pickup not done within 2 mins
→ auto-unassign, driver off-shift
Update load status
POST /complete-next-stop (drop-off)
Mark load COMPLETED
nextAssignment (if any)
POST /shift/end
End shift
Admin
Driver
Backend
DB