UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

HorseShare - clona Bolt/Uber

Inteligenta Artificiala

Autori: Stoica Mihai-Nicusor
Stolniceanu Iustin-Pavel
Grupa: 30237

Facultatea de Automatica
si Calculatoare

8 ianuarie 2026

Cuprins

1 Introducere

1.1 Contextul și Motivația

În era digitală actuală, economia de tip "gig economy" și serviciile de ride-sharing au transformat fundamental modul în care oamenii se deplasează. Proiectul **HorseShare** propune o soluție software care adaptează modelul clasic de e-hailing la transportul ecvestru. Această aplicație răspunde nevoii de a digitaliza un sector tradițional, oferind acces rapid și sigur la mijloace de transport inedite.

1.2 Obiectivul Proiectului

Scopul principal al aplicației HorseShare este de a facilita conexiunea dintre posesorii de cai sau trăsuri ("Carriage Drivers") și clienți ("Riders"). Aplicația digitalizează complet procesul de rezervare, eliminând negocierea directă și nesigură.

1.3 Funcționalități Principale

Conform analizei cerințelor, sistemul deservește două tipuri majore de interacțiuni:

- **Transport în regim de taxi (Carriage Ride):** Utilizatorul solicită o cursă, specifică o rută (Pickup și Drop-off), primește o estimare de preț și este preluat de un driver.
- **Închiriere (Rent a Horse):** Utilizatorul poate vizualiza pe hartă caii disponibili pentru închiriere. O caracteristică importantă este simularea prezenței cailor în zonele fără acoperire prin generare procedurală.

2 Analiza Cerințelor și Modelarea Sistemului

2.1 Diagrama Use Case

Diagrama de mai jos evidențiază rolurile principale: *Rider* și *Carriage Driver*.

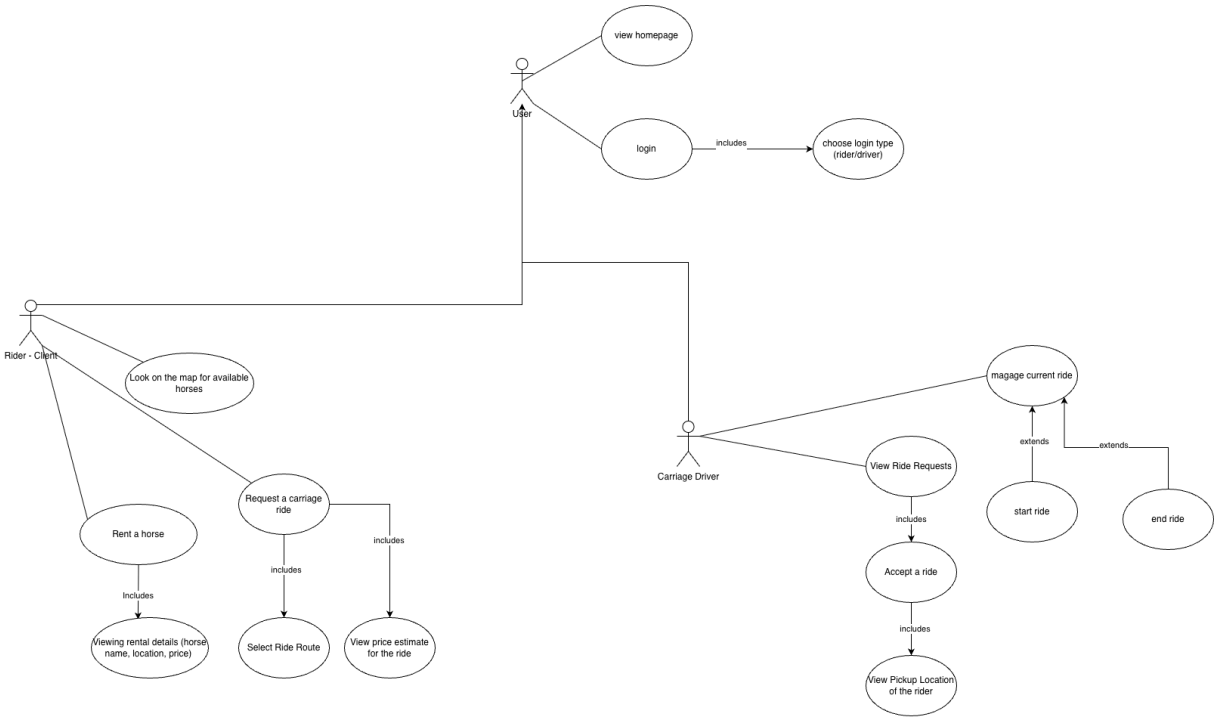

Figura 1: Diagrama Use Case a aplicației HorseShare

2.2 Modelarea Autentificării

Securitatea accesului este gestionată prin Firebase Auth. Diagramele următoare detaliază fluxul de logare și stările sesiunii.

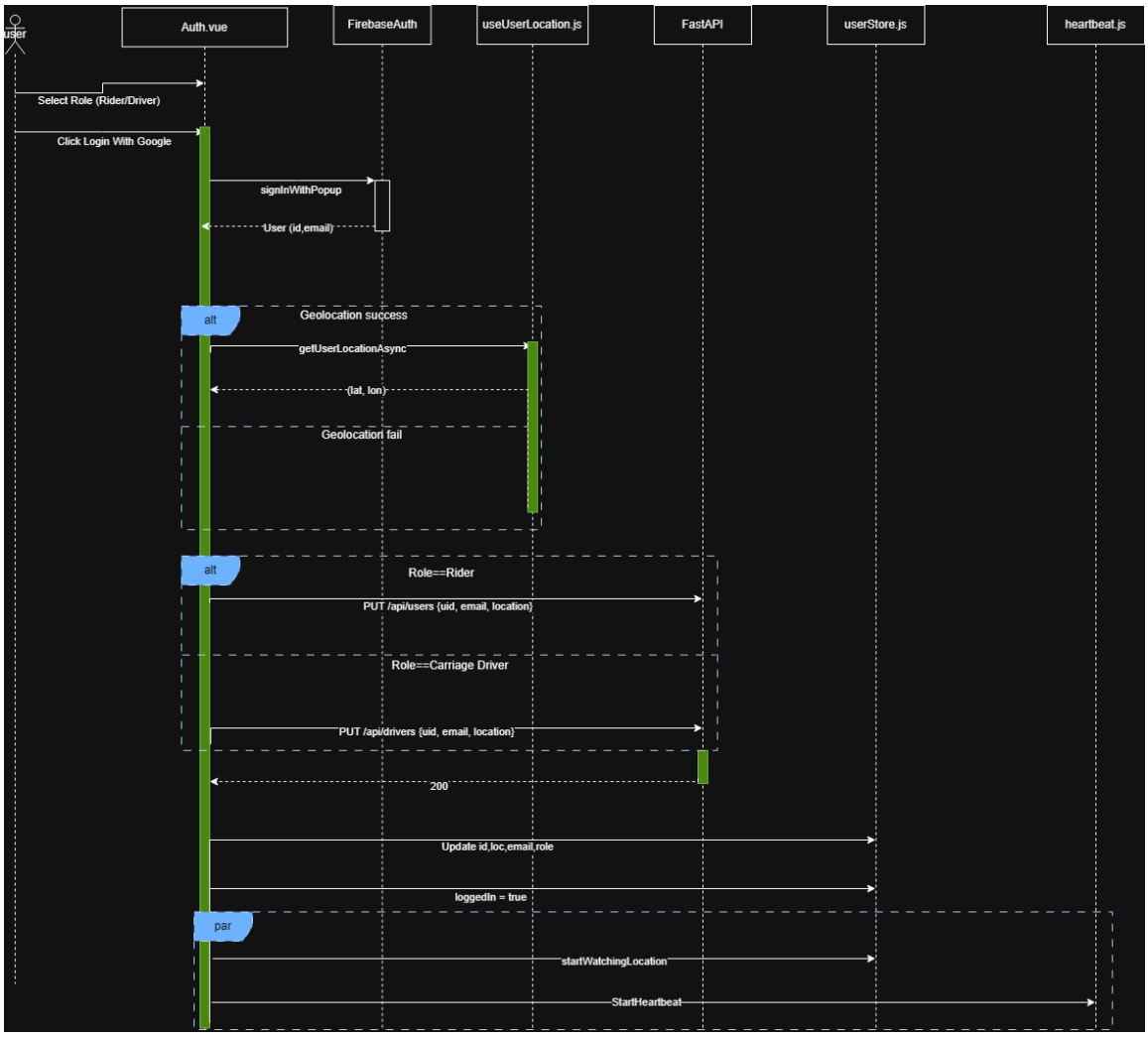

Figura 2: Diagrama de Secvență: Procesul de Autentificare

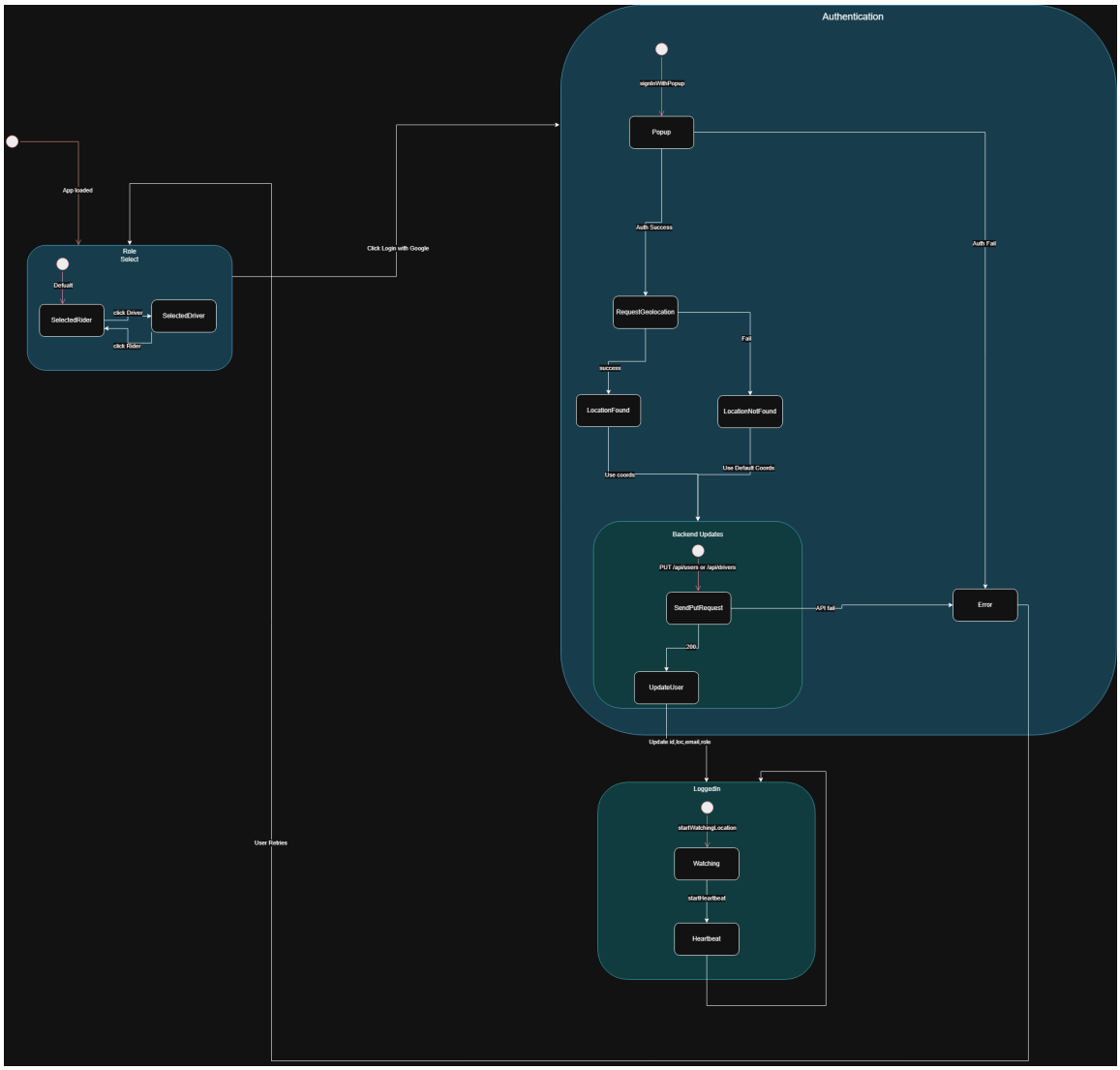

Figura 3: Diagrama de Stare: Ciclul de viață al sesiunii (Login/Logout)

3 Arhitectura Sistemului

Aplicația utilizează o arhitectură decuplată *Client-Server* bazată pe REST API și WebSockets (prin Firebase).

3.1 Componentele Tehnologice

- **Frontend:** *Vue.js* - Interfață reactivă Single Page Application.
- **Backend:** *FastAPI (Python)* - Procesare logică, calcule geometrice și generare date.
- **Bază de Date:** *Firebase Realtime Database* - Stocare NoSQL.
- **Hărți:** *Leaflet.js + OpenStreetMap* - Randare hărți și calcul rute.

3.2 Modelul Datelor (Schema NoSQL)

Datele sunt organizate în colecții JSON. Structura este definită de clasele Pydantic din backend (‘main.py‘):

Colecție	Câmpuri Cheie	Descriere
`users`	uid, email, location, loggedIn, role	Profilul și locația clienților.
`drivers`	uid, email, location, name, loggedIn	Driveri activi și disponibilitatea lor.
`horses`	id, name, location (lat, lon)	Cai generați procedural pentru închiriere.
`rides`	rideId, rider_uid, driver_uid, pickupLocation, destination, price, status	Detaliile complete ale unei curse (stări: pending, accepted, completed).

Tabela 1: Schema bazei de date Firebase

4 Design Detaliat și Fluxuri de Date

4.1 Actualizarea Locației (Heartbeat)

Pentru a menține consistența datelor spațiale, aplicația folosește un mecanism de *Heartbeat*. Dacă un utilizator nu trimite semnal timp de 30 de secunde, este marcat automat ca *Offline*.

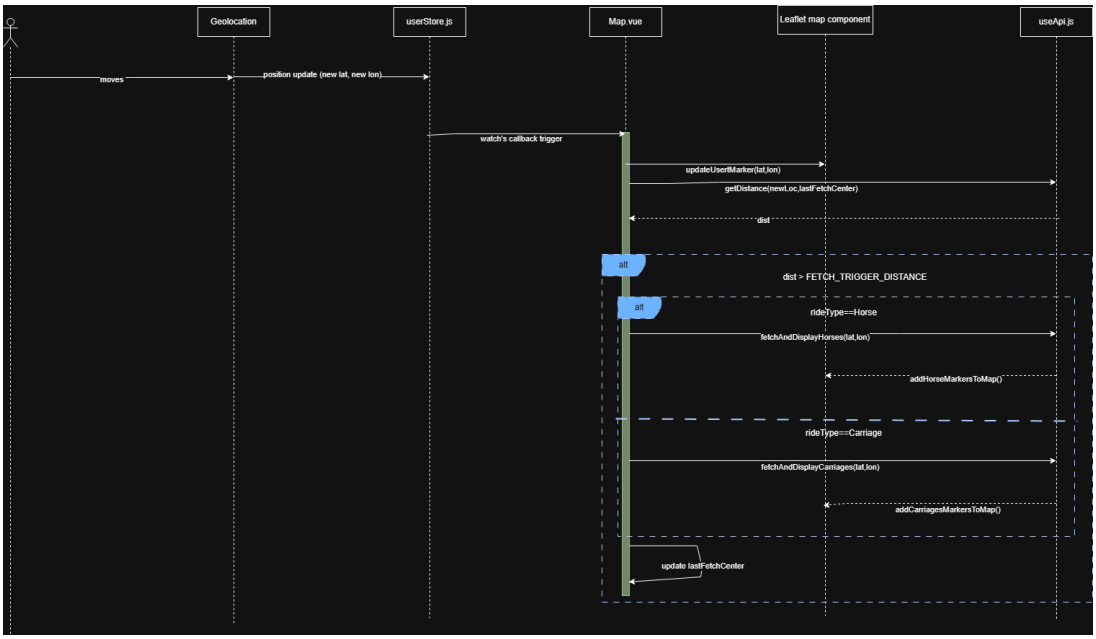

Figura 4: Secvența de actualizare a locației (Heartbeat)

4.2 Fluxul Cursei (Ride Lifecycle)

Interacțiunea dintre Rider și Driver este sincronizată în timp real.

4.2.1 Perspectiva Clientului (Rider)

Clientul inițiază cererea și interfața se adaptează automat la schimbările de stare din baza de date.

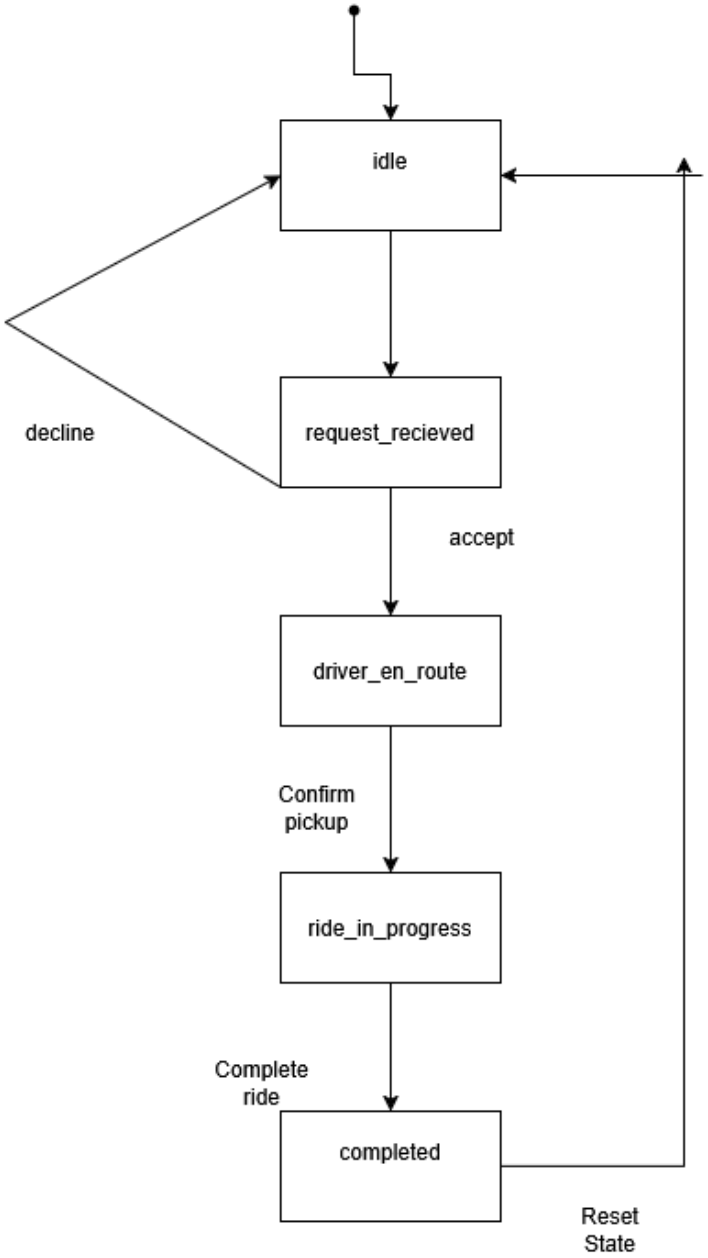

Figura 5: Fluxul cursei din perspectiva Clientului

4.2.2 Perspectiva Șoferului (Driver)

Șoferul primește cereri filtrate pe baza proximității geografice.

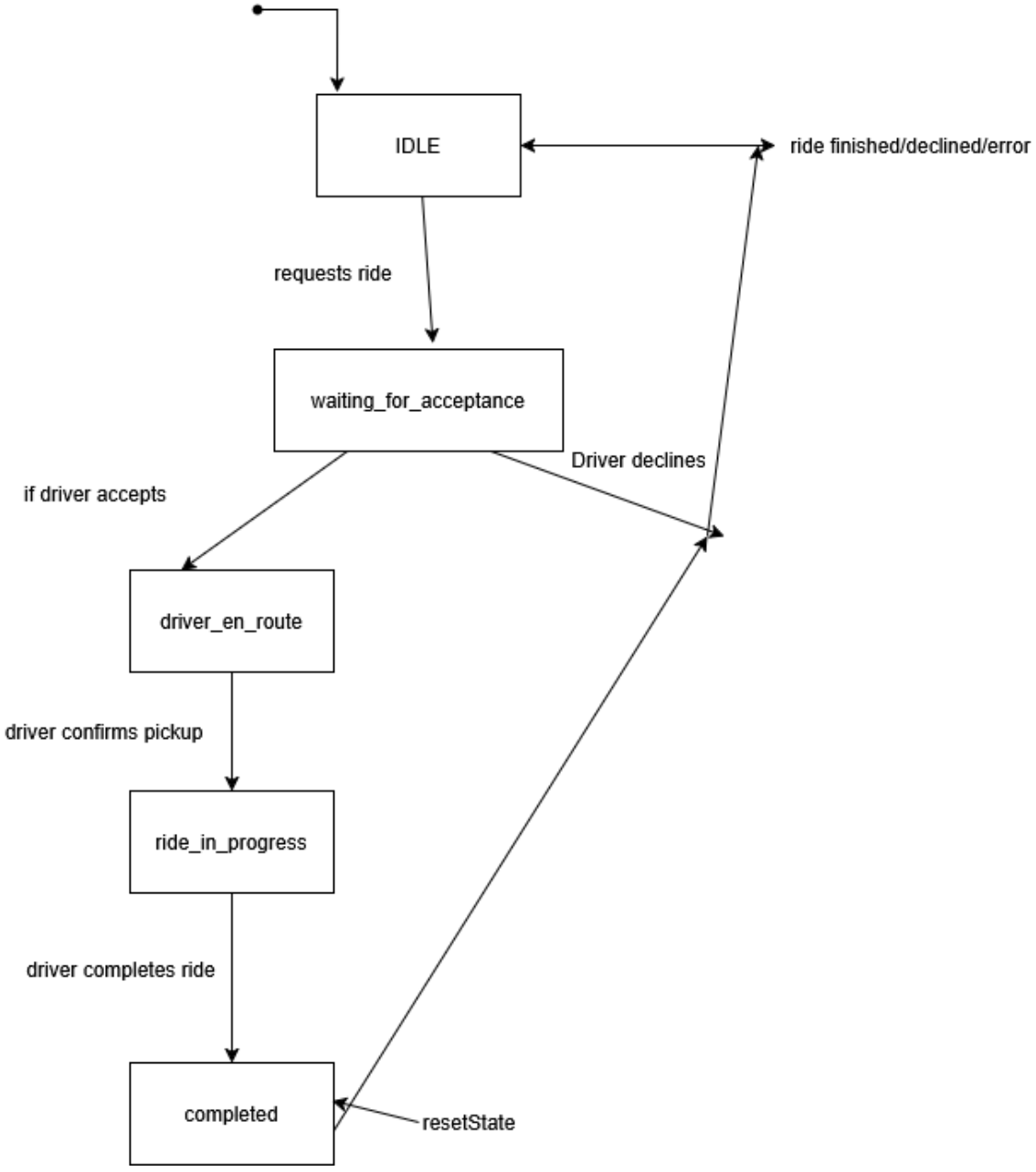

Figura 6: Fluxul cursei din perspectiva Șoferului

5 Implementare

Această secțiune descrie componentele software critice, împărțite în logica de backend (server-side) și frontend (client-side).

5.1 Backend: Generarea Procedurală a Cailor

Deoarece sistemul funcționează într-un mediu simulat (fără cai reali echipați cu GPS), a fost necesară implementarea unei soluții de populare automată a hărții în `horse_utils.py`.

```
def generate_horses(center_lat: float, center_lon: float, range_km:
    float, count: int):
    # Conversie km in grade aproximative
    range_degrees = range_km / EARTH_RADIUS_KM * (180 / math.pi)
    horse_list = []

    for _ in range(count):
        # Generare punct aleator in cerc folosind coordonate polare
        distance = math.sqrt(random.random()) * range_degrees
        angle = random.random() * 2 * math.pi

        lat = center_lat + distance * math.cos(angle)
        lon = center_lon + distance * math.sin(angle)

        if -90 < lat < 90 and -180 < lon < 180:
            horse_list.append({
                "lat": lat,
                "lon": lon,
                "name": random.choice(names)
            })
    return horse_list
```

Listing 1: Algoritmul de generare procedurală (horse_utils.py)

5.2 Backend: Filtrarea Spațială și Geofencing

Pentru eficiență, serverul returnează doar șoferii aflați într-o anumită rază față de client. Calculul se face folosind formula Haversine în `driver_utils.py`.

```
def compute_drivers_in_range(all_drivers_data, lat, lon, range):
    drivers_in_range = []
    for driver_data in all_drivers_data.values():
        if driver_data.get("loggedIn") == True:
            loc = driver_data.get("location")
            # Verifica distanta sferica
            if is_in_range(loc[0], loc[1], lat, lon, range):
                drivers_in_range.append(driver_data)
    return drivers_in_range
```

Listing 2: Identificarea șoferilor din proximitate

5.3 Backend: Managementul Sesiunilor (Cleanup)

Fișierul `main.py` gestionează un proces de fundal (*background task*) care elimină utilizatorii inactivi pentru a menține baza de date curată.

```
async def run_cleaup_job_loop():
    while True:
```

```python
        try:
            # Utilizatorii care nu au dat heartbeat in ultimele 30s
            stale_time = datetime.now(timezone.utc) - timedelta(seconds
    =30)

            await db_set_inactive_users("users", stale_time)
            await db_set_inactive_users("drivers", stale_time)
        except Exception as e:
            print(f"Cleanup failed: {e}")

        # Ruleaza la fiecare 60 de secunde
        await asyncio.sleep(60)
```

Listing 3: Bucla de curățare a sesiunilor inactive (main.py)

5.4 Frontend: Integrarea Hărții (Leaflet)

Pe partea de client, utilizăm *Leaflet Routing Machine* pentru a desena ruta optimă. Componenta `useMap.js` gestionează instanța hărții și marker-ele.

```javascript
routingControl = L.Routing.control({
  waypoints: [userLatLng, dest],
  routeWhileDragging: true,
  createMarker: (i, waypoint, n) => {
    // Marker personalizat doar pentru destinatie
    if (i === 0) return null
    return L.marker(waypoint.latLng, { draggable: true })
  }
}).addTo(mapInstance)
```

Listing 4: Configurarea rutei în Vue.js (useMap.js)

5.5 Frontend: Logica de Ride-Booking

Calculul prețului estimativ și inițierea tranzacției se realizează în `useRide.js` înainte de a trimite cererea către API.

```javascript
// Calculul pretului estimativ bazat pe distanta
if (destLat && destLon) {
    const distMeters = getDistance(lat, lon, destLat, destLon);
    const distKm = distMeters / 1000;
    price = Math.round(BASE_RATE + (distKm * RATE_PER_KM));
}

// Trimiterea cererii catre Backend
const response = await fetch(`${API_URL}/api/rides`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        rider_uid: userStore.uid,
        price: price,
        status: 'pending'
    })
```

```
});
```

Listing 5: Calculul prețului și trimiterea cererii (useRide.js)

5.6 Frontend: Sincronizare (Heartbeat)

Pentru a menține conexiunea activă, clientul trimite periodic un semnal "keep-alive" prin `heartbeat.js`.

```
export const startHeartbeat = () => {
    if (heartbeatIntervalId) clearInterval(heartbeatIntervalId);

    sendHeartbeat(); // Trimite imediat primul ping

    // Seteaza intervalul de 30 secunde
    heartbeatIntervalId = setInterval(sendHeartbeat, 30000);
    console.log(‘Heartbeat loop started.‘);
};
```

Listing 6: Bucla de Heartbeat (heartbeat.js)

6 Concluzii

Proiectul **HorseShare** a reușit să implementeze un sistem complex de ride-sharing, integrând servicii de hărți, baze de date în timp real și algoritmi de backend pentru generarea dinamică de conținut. Arhitectura scalabilă permite gestionarea eficientă a sesiunilor de utilizatori și a curselor, oferind o experiență fluidă similară aplicațiilor comerciale majore.