HorseShare Documentation

Stolniceanu Iustin Stoica Mihai
Echipa: Şarja de la RobĀneŞti

January 14, 2026

Contents

1 Scurtă descriere a proiectului

HorseShare este o aplicație de ride-sharing inspirată de platforme precum Uber, dar cu o abordare unică și ecologică: înlocuiește mașinile cu trăsuri și cai. Proiectul a fost dezvoltat în cadrul cursului de Inginerie Software la UTCN și servește ca un exemplu practic de aplicare a principiilor de inginerie software într-un proiect full-stack.

Aplicația permite utilizatorilor ("Riders") să comande o cursă de la locația lor curentă către o destinație aleasă, vizualizând șoferii ("Carriage Drivers") disponibili pe o hartă interactivă. Șoferii, la rândul lor, pot vedea cererile de curse, le pot accepta sau respinge și își pot actualiza statusul în timp real.

2 Limbajele și tehnologiile folosite

Arhitectura proiectului este una de tip client-server, cu un frontend web și un backend RESTful.

2.1 Frontend

- **Vue.js (v3):** Un framework progresiv pentru construirea interfețelor utilizator.
- **Vite:** Unelte rapide pentru dezvoltarea web modernă, folosit pentru build și server de dezvoltare.
- **Tailwind CSS:** Un framework CSS de tip utility-first pentru stilizarea rapidă a componentelor.
- **Leaflet.js:** O bibliotecă JavaScript open-source pentru hărți interactive. A fost utilizată pentru afișarea hărții, a locațiilor utilizatorilor și a rutelor.
- **Firebase Authentication:** Folosit pentru autentificarea utilizatorilor prin email și parolă.

2.2 Backend

- **Python 3:** Limbajul de programare principal pentru server.
- **FastAPI:** Un framework web modern și rapid pentru construirea API-urilor cu Python, bazat pe type hints.
- **Uvicorn:** Un server ASGI (Asynchronous Server Gateway Interface) pentru rularea aplicației FastAPI.
- **Firebase Realtime Database & Firestore:** Folosite ca baze de date NoSQL pentru a stoca în timp real informații despre starea curselor, locațiile șoferilor și datele utilizatorilor.
- **Firebase Admin SDK:** Utilizat în backend pentru a interacționa în mod securizat cu serviciile Firebase.

2.3 DevOps

- **Docker & Docker Compose:** Tehnologii de containerizare folosite pentru a asigura un mediu de dezvoltare și producție consistent, izolat și ușor de instalat pentru frontend și backend.

3 Cerințe și Diagrama Use-Case

3.1 Cerințe funcționale

- CF1: Sistemul trebuie să permită crearea unui cont nou pentru utilizatori (Rider sau Driver).
- CF2: Sistemul trebuie să permită autentificarea utilizatorilor existenți.
- CF3: Utilizatorii trebuie să poată vizualiza o hartă interactivă.
- CF4: Rider-ul trebuie să poată vedea șoferii disponibili în apropiere.
- CF5: Rider-ul trebuie să poată selecta o destinație și să solicite o cursă.

- CF6: Rider-ul trebuie să vadă o estimare a prețului înainte de a comanda.
- CF7: Driver-ul trebuie să primească notificări pentru cererile de cursă noi.
- CF8: Driver-ul trebuie să poată accepta sau respinge o cerere de cursă.
- CF9: Rider-ul trebuie să vadă statusul cursei în timp real (acceptată, în drum, cursă în progres).
- CF10: Driver-ul trebuie să poată actualiza statusul cursei (ex: a preluat pasagerul, a finalizat cursa).
- CF11: Sistemul trebuie să stocheze informațiile despre utilizatori și curse.
- CF12: Atât rider-ul cât și driver-ul trebuie să își actualizeze locația pe hartă în timp real.

3.2 Cerințe non-funcționale

- **Performanță:** Timpul de răspuns pentru acțiunile critice (ex: acceptarea unei curse, actualizarea locației) trebuie să fie sub 3 secunde. Harta trebuie să rămână fluidă chiar și cu 50+ markere active.
- **Fiabilitate:** Sistemul trebuie să gestioneze erorile de conexiune la rețea fără a pierde starea curentă a utilizatorului (ex: reconectare automată la serviciul de heartbeat).
- **Usabilitate:** Interfața cu utilizatorul trebuie să fie intuitivă, permițând unui utilizator nou să comande o cursă în mai puțin de 1 minut de la autentificare.
- **Securitate:** Autentificarea trebuie să fie securizată prin Firebase Authentication. Toate rutele de backend care necesită autentificare trebuie să fie protejate corespunzător.
- **Mentenabilitate:** Codul trebuie să fie modular și bine documentat (prin design patterns) pentru a facilita modificări ulterioare.

3.3 Diagrama Use-Case

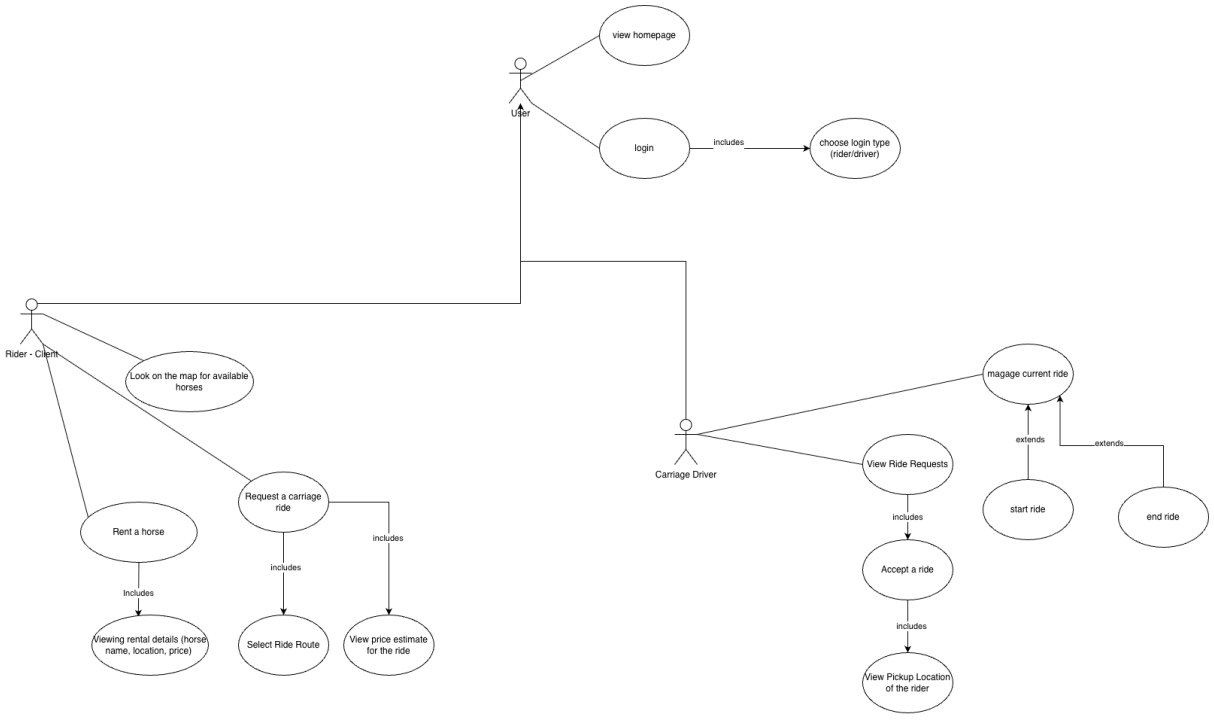

Figure 1: Diagrama Use-Case pentru aplicația HorseShare.

4 Diagrame

4.1 Diagrame realizate de Stolniceanu Iustin

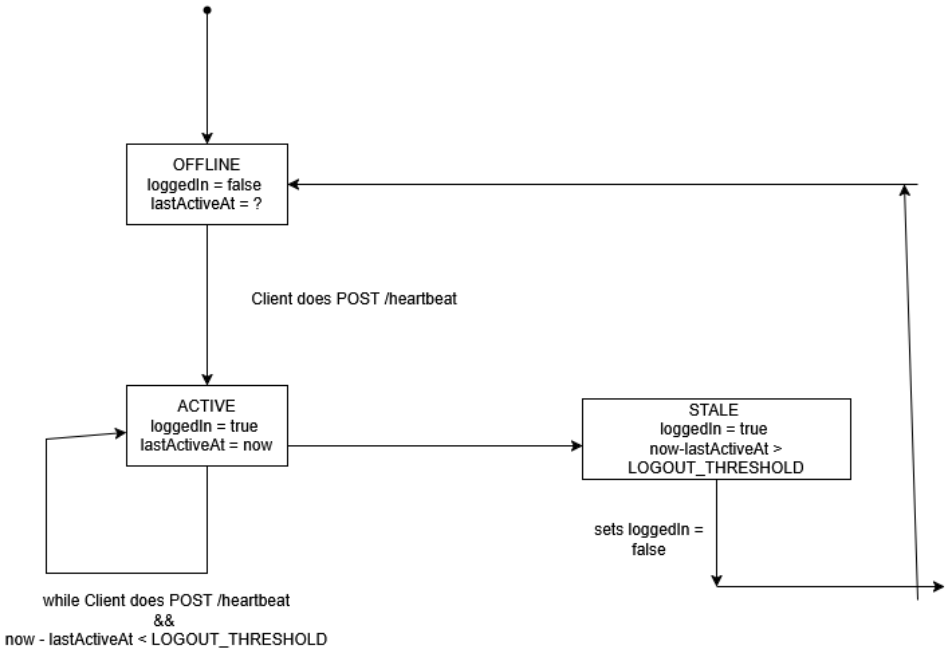

Figure 2: Diagrama de stări pentru Heartbeat (realizată de Stolniceanu Iustin).

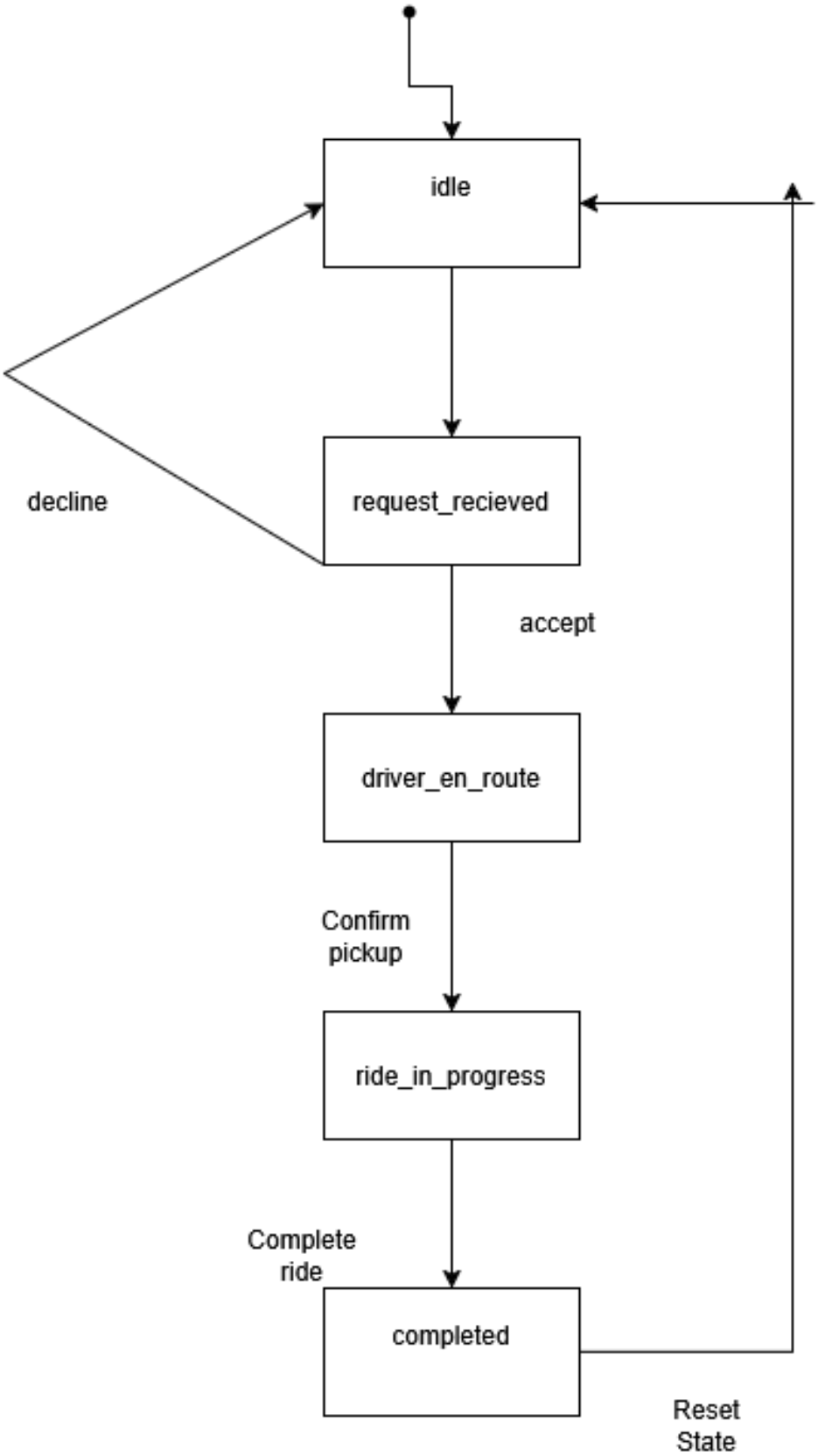

Figure 3: Diagrama de stări: Cursă - Perspectiva Șofer (realizată de Stolniceanu Iustin).

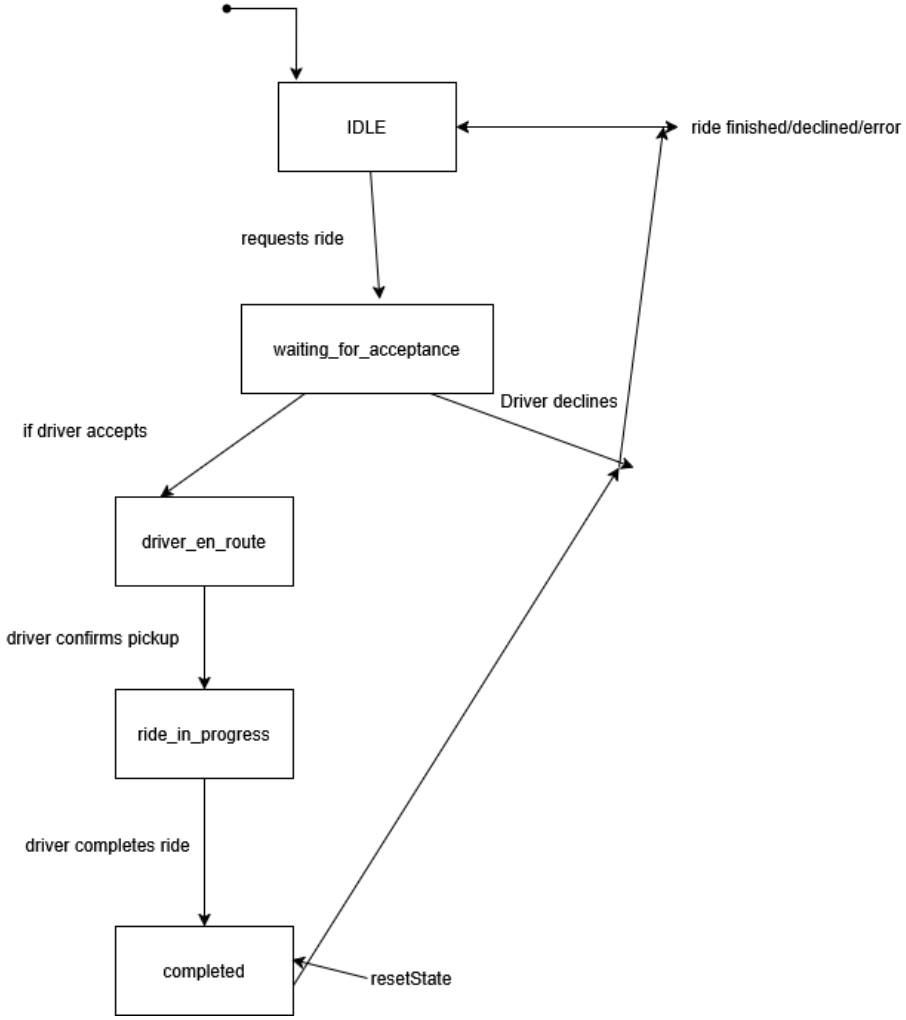

Figure 4: Diagrama de stări: Cursă - Perspectiva Utilizator (realizată de Stolniceanu Iustin).

4.2 Diagrame realizate de Stoica Mihai

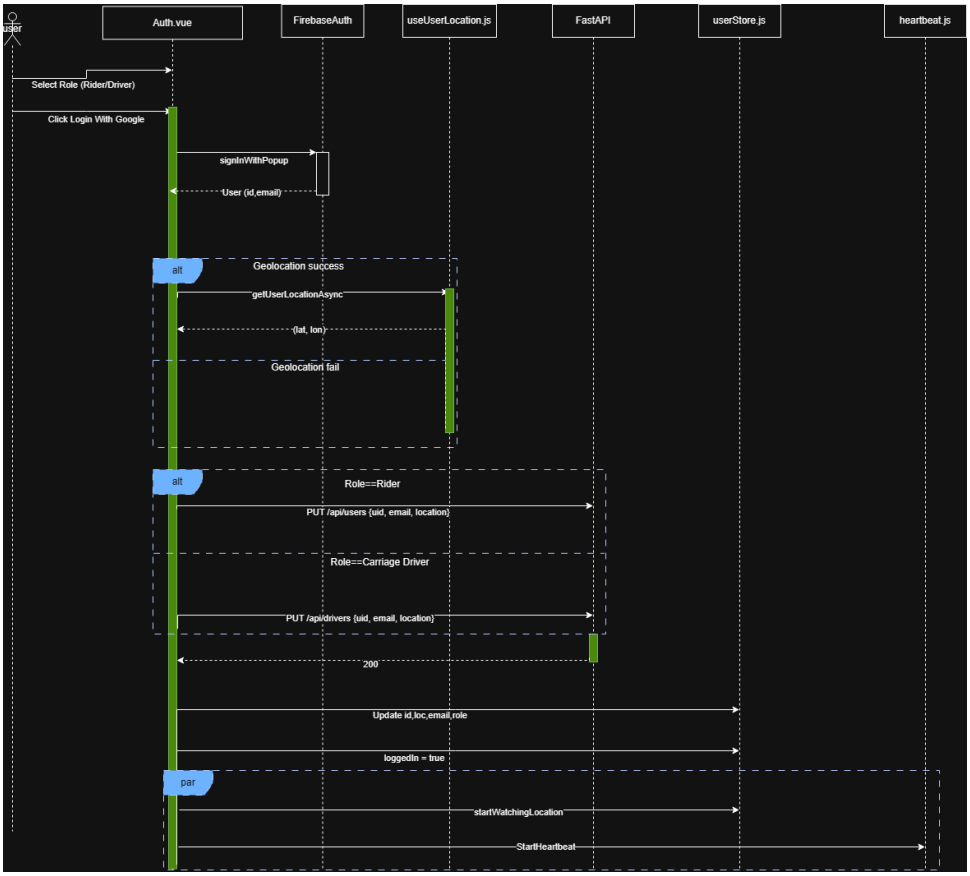

Figure 5: Diagrama de secvență pentru autentificare (realizată de Stoica Mihai).

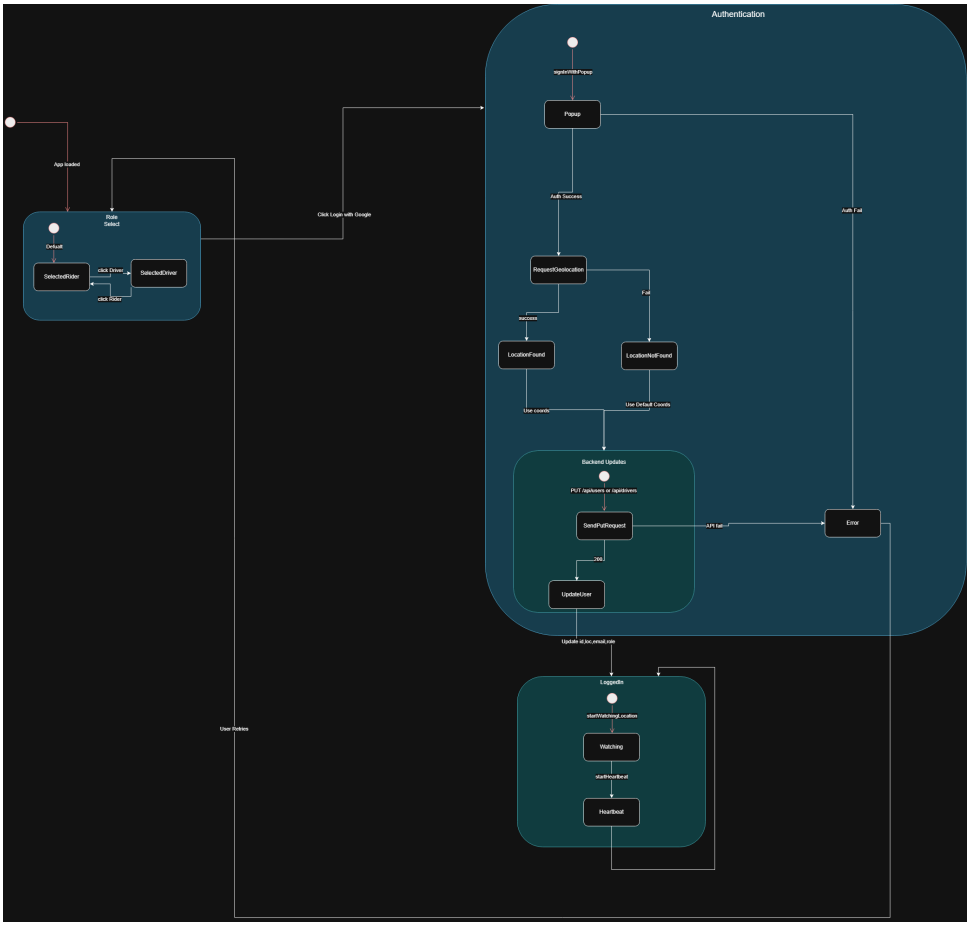

Figure 6: Diagrama de stări pentru autentificare (realizată de Stoica Mihai).

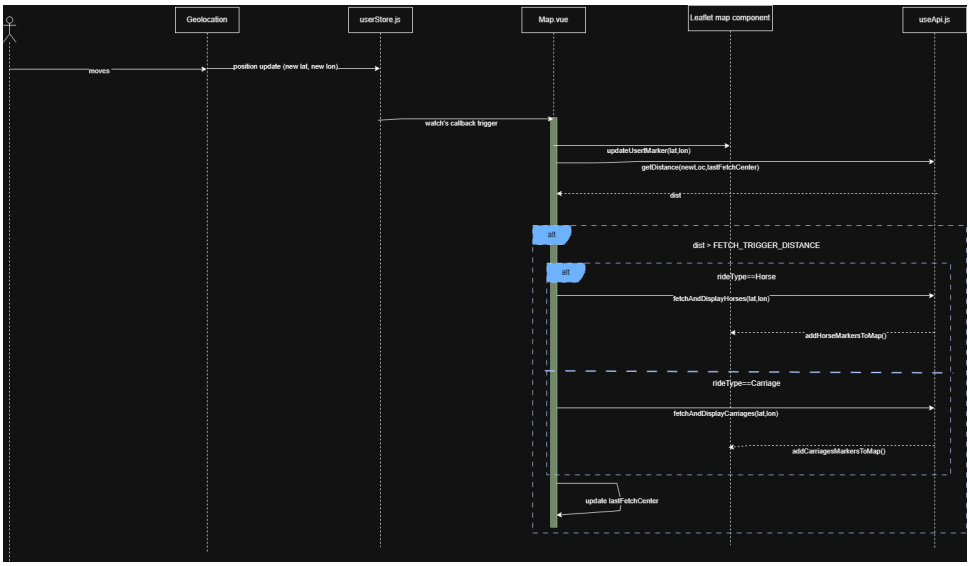

Figure 7: Diagrama de secvență pentru actualizarea locației (realizată de Stoica Mihai).

5 Design Patterns

5.1 Heartbeat

- **Nume persoană:** Stolniceanu Iustin

- **Problemă:** Într-o aplicație distribuită cum este HorseShare, backend-ul trebuie să știe în permanență care utilizatori (atât șoferi cât și clienți) sunt activi și online. Fără un mecanism activ de monitorizare, un utilizator care închide brusc aplicația sau pierde conexiunea la internet ar putea apărea în continuare ca fiind "online", ducând la alocarea incorectă a curselor către șoferi indisponibili sau la afișarea de informații învechite pe hartă.
- **Soluție:** Pattern-ul Heartbeat rezolvă această problemă prin implementarea unui mecanism prin care clientul trimite la intervale regulate un semnal ("bătaie de inimă") către server pentru a-și semnala prezența. Serverul, la rândul său, monitorizează aceste semnale. Dacă un client nu mai trimite semnale pentru o perioadă de timp definită, serverul îl consideră deconectat și îi poate actualiza statusul.
- **Implementare:** În proiect, funcționalitatea este implementată în fișierul `frontend/src/composables/heartbeat.js`. Funcția `startHeartbeat` inițiază un proces care, la un interval de timp regulat (definit de `HEARTBEAT_INTERVAL_MS`), trimite o cerere de tip POST către un endpoint specific de pe server (`/api/users/heartbeat` sau `/api/drivers/heartbeat`). Această cerere conține ID-ul unic al utilizatorului. În backend, serverul primește aceste cereri și actualizează un timestamp (ex: `lastActiveAt`) asociat utilizatorului respectiv în baza de date. Un proces separat pe backend poate apoi să verifice periodic și să marcheze ca offline utilizatorii al căror ultim heartbeat a depășit un anumit prag de timp.

5.2 Observer

- **Nume persoană:** Stoica Mihai
- **Problemă:** Aplicația necesită o comunicare și o sincronizare a stării în timp real între mai mulți utilizatori și server. De exemplu, un client (Rider) trebuie să vadă pe hartă cum locația șoferului se actualizează în timp ce acesta se îndreaptă spre el. De asemenea, clientul trebuie să fie notificat instantaneu când starea cursei se schimbă (ex: din "pending" în "accepted" sau din "picked$_up$"în"*completed*").*Ainterogaserverullaintervalescurte*(*polling*)*pentruaverificadacăexistămodificărieste*
- **Soluție:** Pattern-ul Observer oferă o soluție elegantă. Un obiect "subiect" (în acest caz, o intrare în Firebase Realtime Database, cum ar fi o cursă sau locația unui șofer) menține o listă de "observatori" (componentele din aplicația client). Când starea subiectului se modifică, acesta notifică automat toți observatorii, care își pot actualiza starea în consecință. Acest flux este ilustrat și în diagrama de secvență pentru actualizarea locației.
- **Implementare:** În `frontend/src/composables/useRide.js`, funcția `listenForRideUpdates` este un exemplu clar al acestui pattern. Ea folosește funcția `onValue` din SDK-ul Firebase pentru a "observa" o anumită cursă în baza de date. Se creează o referință la resursa din baza de date, iar funcția `onValue` atașează un callback (observatorul) care este executat ori de câte ori datele de la acea referință se modifică.

6 Scurt README

Acesta este un proiect de tip "Uber clone" care folosește cai și trăsuri, realizat pentru materia de Inginerie Software.

6.1 Cum se rulează proiectul

1. Asigurați-vă că aveți Docker și Docker Compose instalate.
2. Clonați repository-ul.
3. Creați un fișier `.env` în directorul rădăcină și adăugați variabilele de mediu necesare pentru cheile de API Firebase și alte configurări.
4. Rulați comanda `docker-compose up --build` din directorul rădăcină.
5. Accesați frontend-ul la `http://localhost:5173` și backend-ul va rula la `http://localhost:8000`.