

ExploreR — Generator de rute

Documentație proiect

Echipă: Razvan Valeanu, Emanuel Tutelea , Raul Negrea

An/Grupă: *Anul 3, grupa 7*

Disciplina: *Inginerie Software*

Cuprins

1 Scurtă descriere a proiectului	2
1.1 Obiective	2
2 Limbaje și tehnologii folosite	2
3 Cerințe funcționale + diagrama Use-Case	2
3.1 Cerințe funcționale	2
3.2 Diagrama Use-Case	4
4 Cerințe non-funcționale	5
5 Diagrame	6
5.1 Razvan Valeanu	6
5.2 Emanuel Tutelea	9
5.3 Raul Negrea	11
6 Design Patterns	12
6.1 Singleton — (Emanuel)	12
6.2 Factory Method — (Raul)	12
6.3 Facade — (Razvan)	13
7 Scurt README	13
7.1 Structură proiect	13
7.2 Rulare locală (exemplu)	13
7.3 Funcții principale	14

1 Scurtă descriere a proiectului

ExploreR este o aplicație web care permite generarea automată de rute (pentru mașină, bicicletă sau pe jos) pe baza unor preferințe ale utilizatorului (zonă, distanță, punct de start). Utilizatorii pot salva rutele generate, le pot partaja într-un feed comun și pot interacționa social prin like-uri și comentarii.

1.1 Obiective

- Generarea de rute valide pe drumuri reale, cu parametri personalizabili.
- Persistența rutelor (salvare, detalii).
- Partajarea rutelor către un feed public/privat (în funcție de cerință).
- Interacțiuni sociale: like, comentariu, vizualizare detalii.

2 Limbaje și tehnologii folosite

- **Backend:** PHP (Laravel) — API REST, validări, persistență, autentificare.
- **Frontend:** Vue 3 + Vite — interfață, pagini, componente, integrare cu API.
- **Hartă / vizualizare:** Leaflet (sau echivalent) — randare traseu, marker-e, interacțiune.
- **Bază de date:** SQLite (dev) / (*optional: MySQL/PostgreSQL în producție*).
- **Altele:** Git (versionare), JSON (schimb date), Postman/Insomnia (testare API).

3 Cerințe funcționale + diagrama Use-Case

3.1 Cerințe funcționale

1. Autentificare

- Utilizatorul se poate autentifica pentru a accesa funcțiile de salvare/partajare/interacțiuni.

2. Generare rută

- Utilizatorul selectează zona/punctul de start, distanță.
- Sistemul generează o rută pe drumuri reale și o afișează pe hartă.

3. Salvare rută

- Utilizatorul poate salva o rută generată (titlu, descriere, parametri, geometrie).
- Sistemul generează/atribuie un *slug* unic pentru rută.

4. Partajare rută

- Utilizatorul poate marca o rută ca partajată pentru a apărea în feed.

5. Feed rute partajate

- Sistemul afișează o listă cu rute partajate (autor, dată, număr like-uri/comentarii).
- Utilizatorul poate deschide detaliile unei rute din feed.

6. Like / Unlike

- Utilizatorul poate da like sau retrage like-ul de la o rută.
- Sistemul actualizează contorul de like-uri.

7. Comentarii

- Utilizatorul poate vizualiza comentariile unei rute și poate adăuga un comentariu nou.
- Sistemul actualizează contorul de comentarii.

8. Vizualizare detalii rută

- Sistemul afișează geometria rutei pe hartă și metadate (distanță, timp estimat, etc. dacă există).

3.2 Diagrama Use-Case

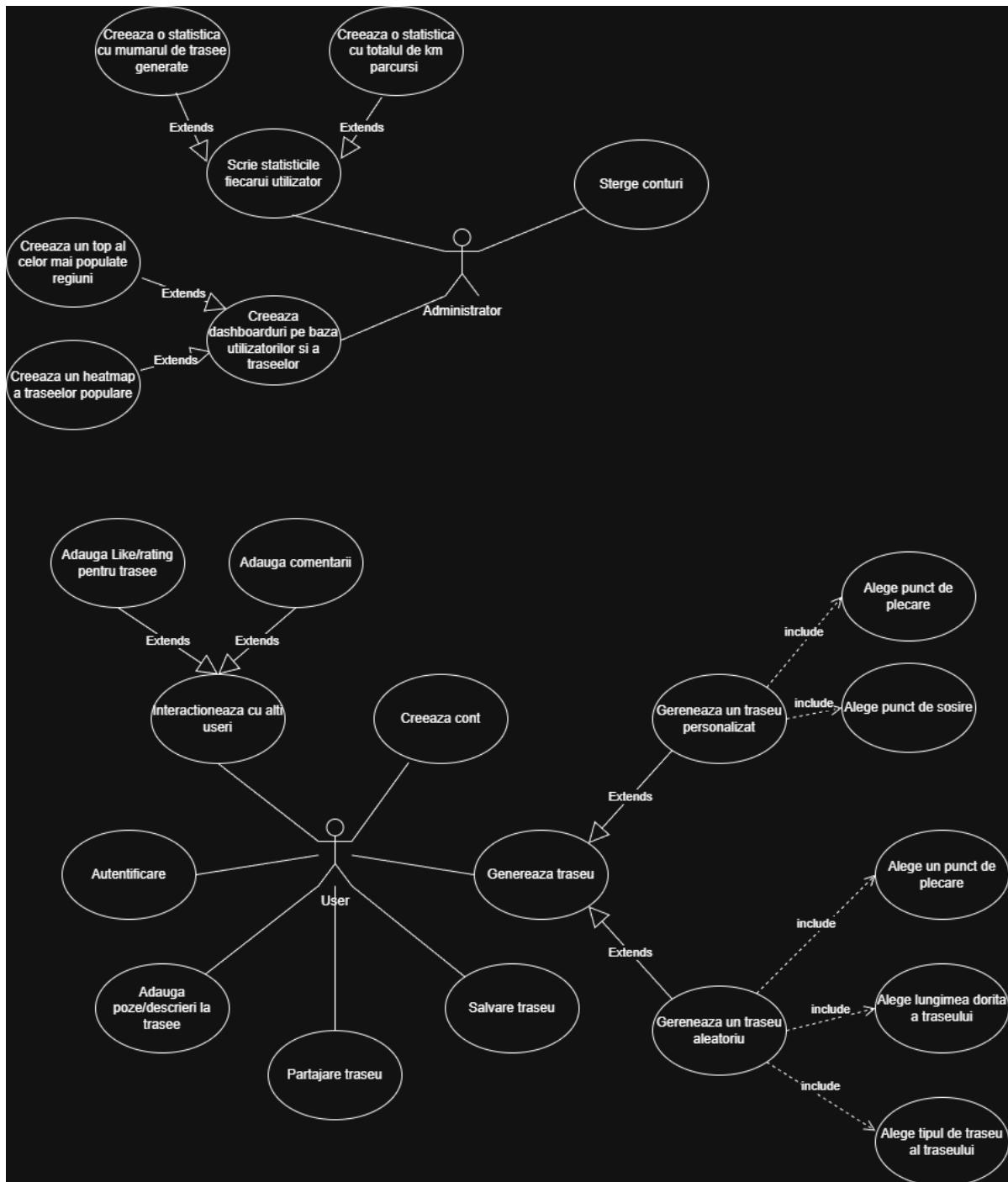


Figura 1: Diagrama Use-Case — ExploreR

4 Cerințe non-funcționale

- **Performanță:** afișarea feed-ului și a detaliilor rutei trebuie să fie responsivă; cererile API uzuale sub 1s în mediu local.
- **Fiabilitate:** datele salvate (rute, like-uri, comentarii) trebuie persistate corect și coerent.
- **Securitate:** endpoint-urile de like/comentariu/salvare necesită utilizator autentificat; validare input (lungime text, tipuri).
- **Ușurință în utilizare:** UI clar (butoane Generate/Save/Share/Like/Comment), feedback vizual la acțiuni.
- **Scalabilitate:** structura permite migrarea de la SQLite la o bază de date server fără rescrieri majore.
- **Mantenabilitate:** separare pe straturi (frontend componente, backend controllers/services/models), cod modular.

5 Diagramme

5.1 Razvan Valeanu



Figura 2: Activity Diagram — (Razvan Valeanu)

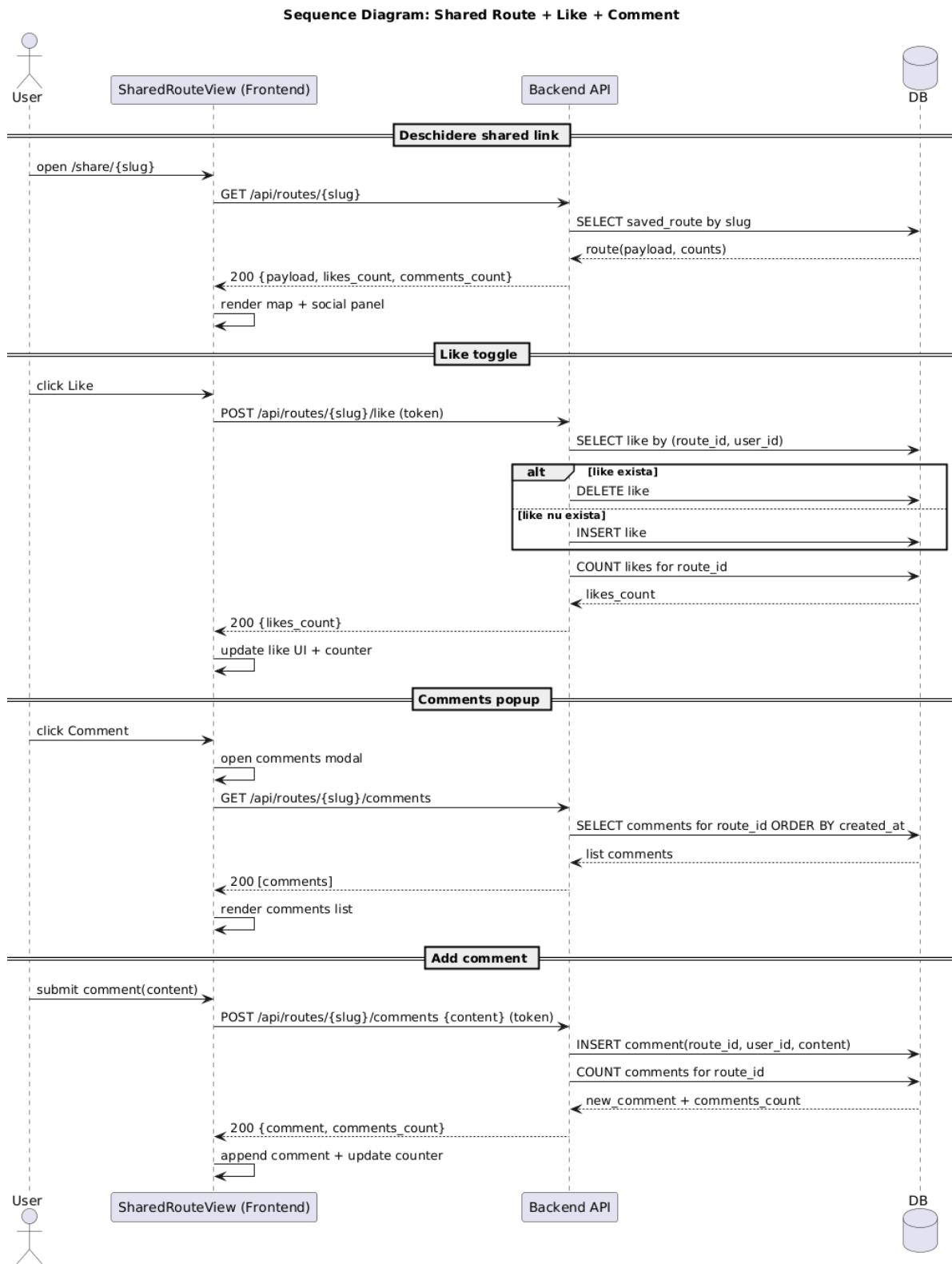


Figura 3: Sequence Diagram — (Razvan Valeanu)

UML State Diagram: Shared Route Social (Like + Comment)

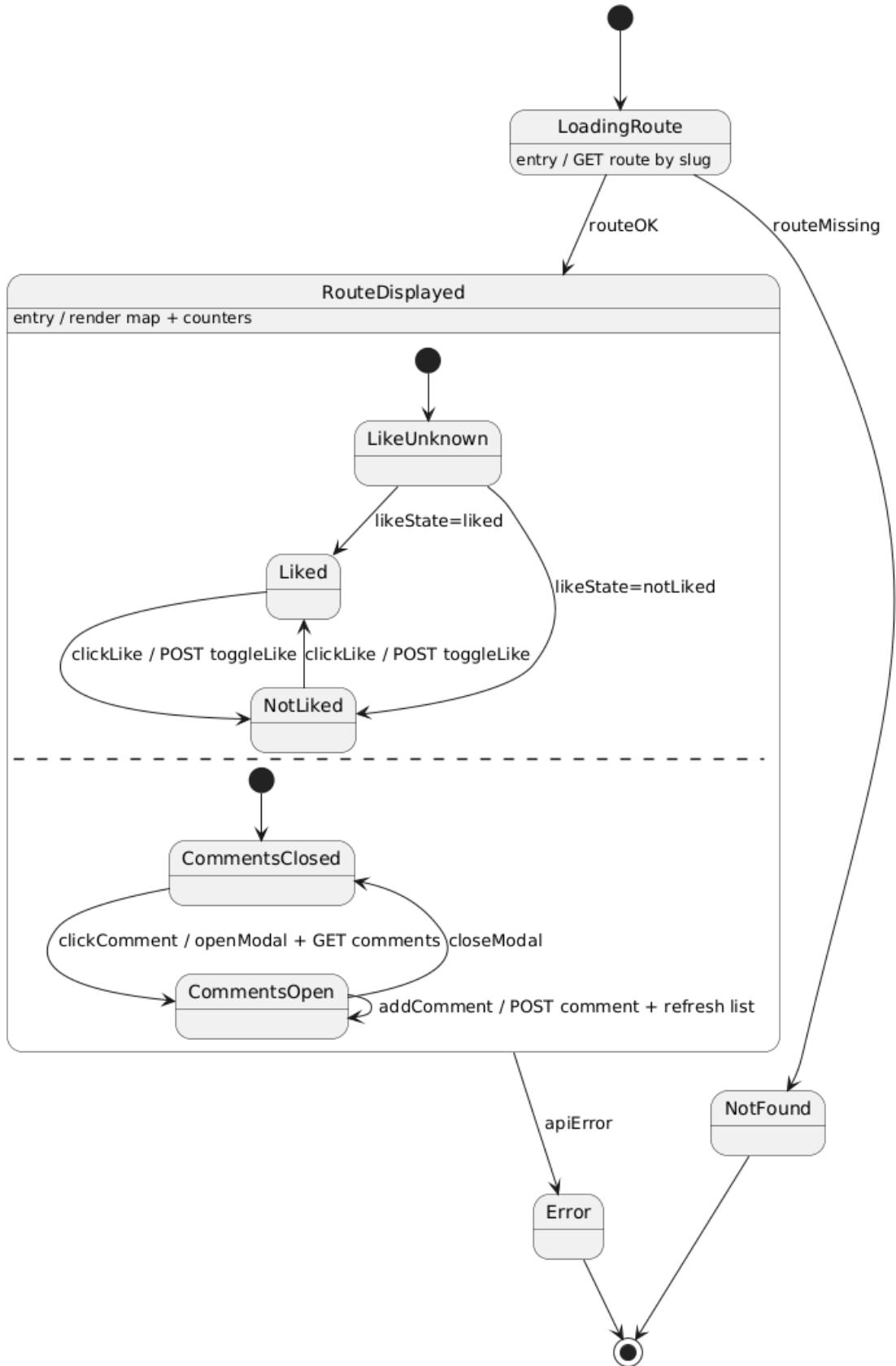


Figura 4: State Diagram — (Razvan Valeanu)

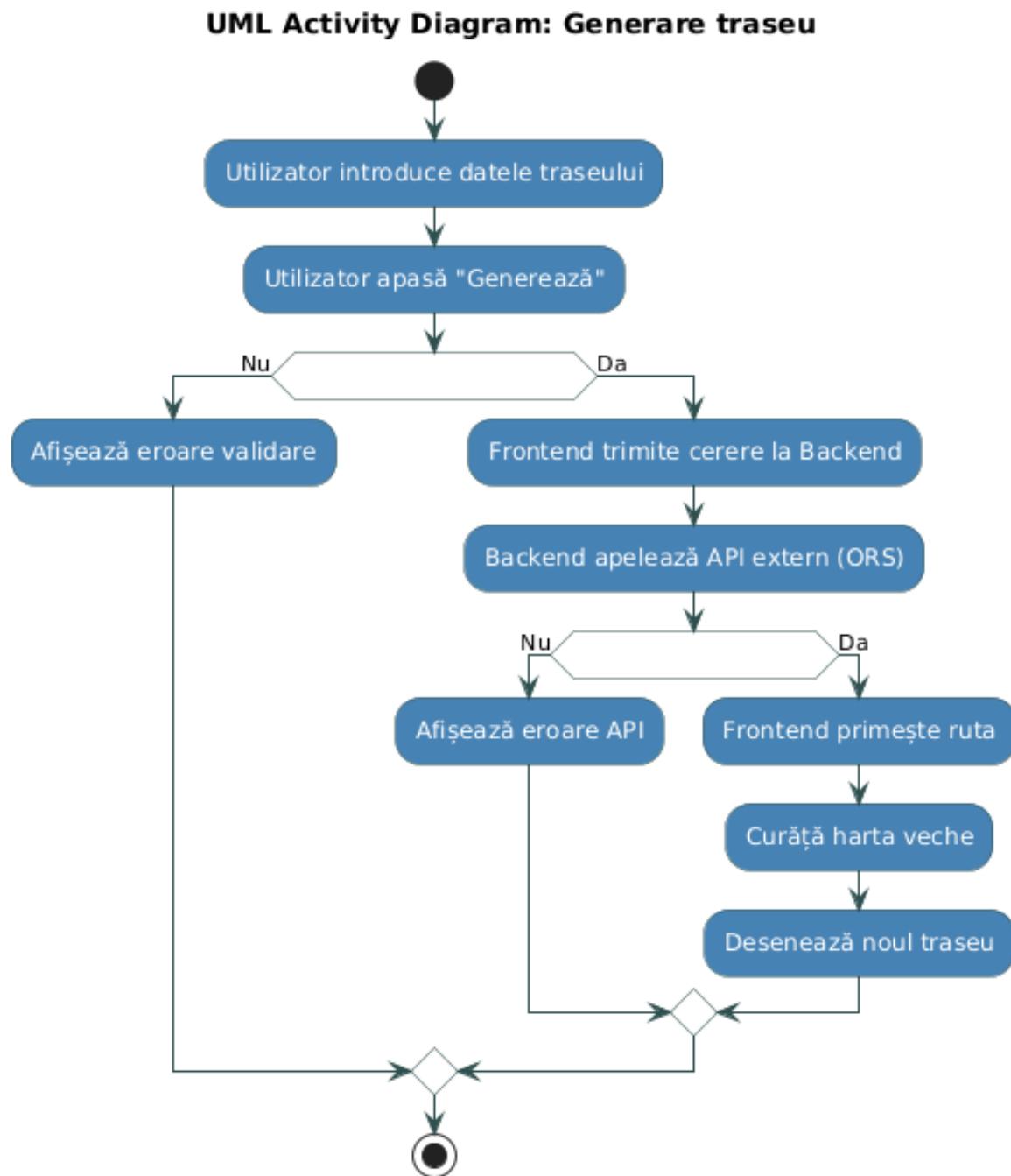


Figura 5: Activity Diagram — (Emanuel)

Sequence Diagram: Generare Traseu

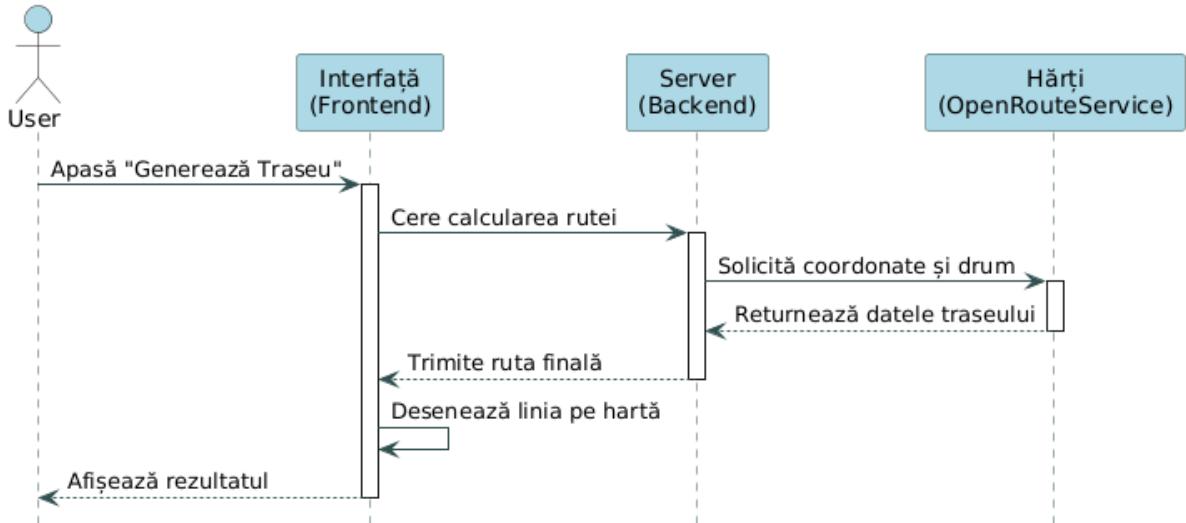


Figura 6: Sequence Diagram — (Emanuel)

UML State Diagram: Generare Traseu

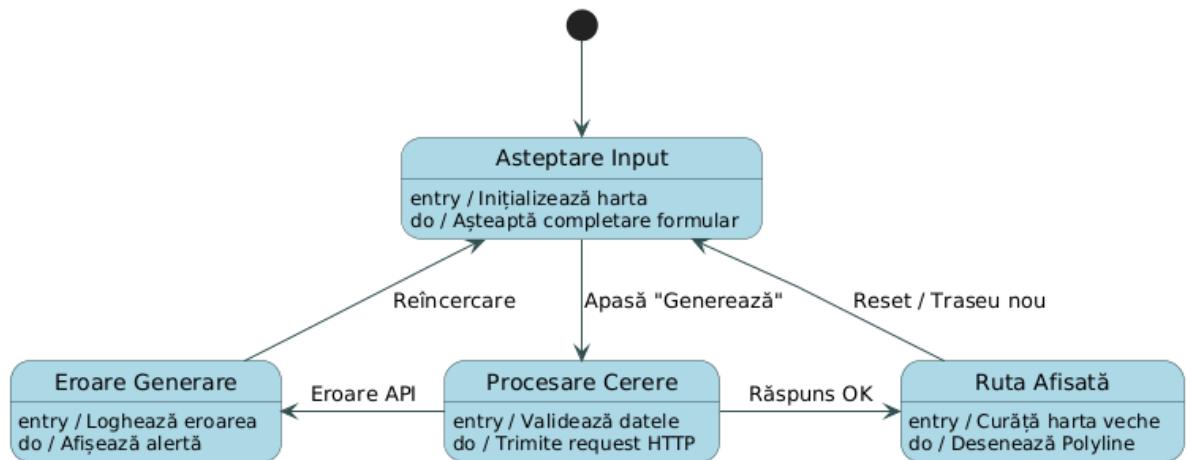


Figura 7: State Diagram — (Emanuel)

5.3 Raul Negrea

UML Activity Diagram: Selectare punct de start pe hartă

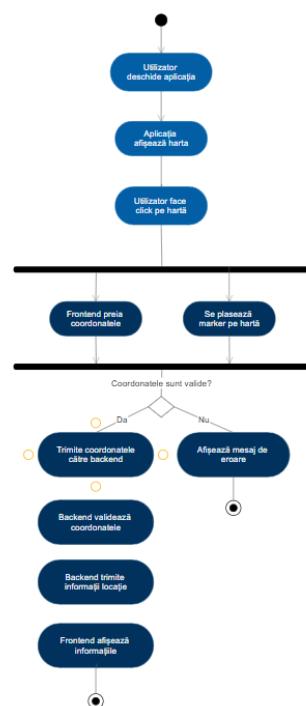


Figura 8: Activity Diagram — (Raul)

Sequence Diagram: Selectarea punctului de start pe hartă

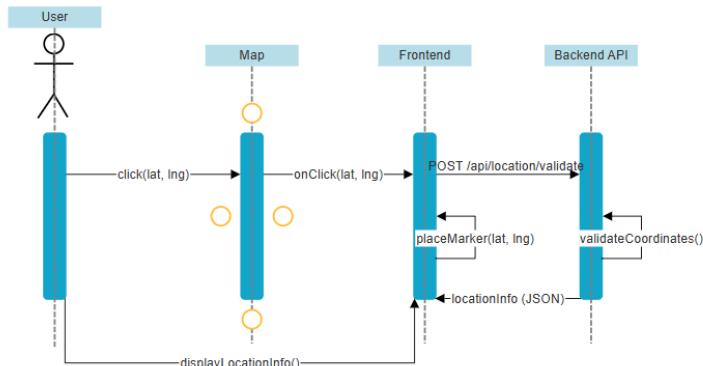


Figura 9: Sequence Diagram — (Raul)

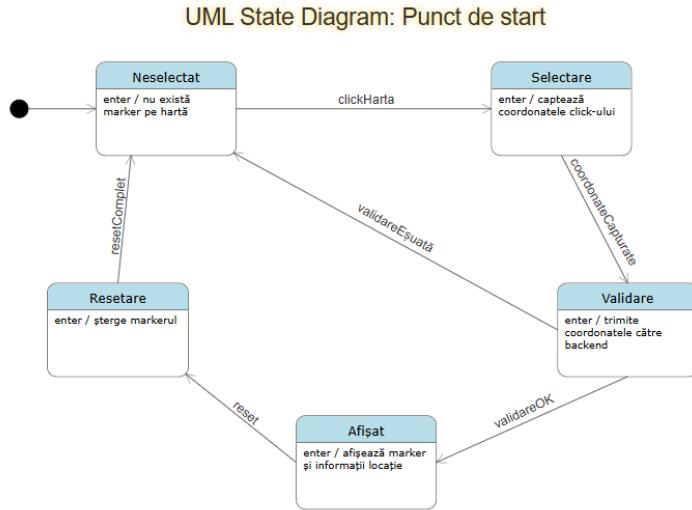


Figura 10: State Diagram — (Raul)

6 Design Patterns

6.1 Singleton — (Emanuel)

Problema identificată: Aplicația folosește serviciul OpenRouteService și are nevoie de cheia API în mai multe rute. Citirea repetată a acestei chei din fișierul de configurare (.env) la fiecare request este ineficientă.

Soluția (Singleton): Am implementat o clasă cu instanță unică care citește cheia o singură dată și o păstrează în memorie pe durata execuției, oferind un punct de acces global.

Detalii legate de implementare:

- Am creat clasa KeyManager în folderul Providers, având constructorul privat.
- Cheia este încărcată o singură dată din .env la prima utilizare.
- În restul aplicației (în web.php), cheia se obține doar prin apelul: `KeyManager::getInstance() ->getKey()`.

6.2 Factory Method — (Raul)

Problema identificată: Există mai multe tipuri de algoritmi de generare trasee (ex.: traseu circular). Dacă alegerea algoritmului este implementată printr-un `if/else` mare în codul principal, devine greu de extins și de testat.

Soluția (Factory Method): Se definește o interfață comună pentru generatorii de rute și clase concrete pentru fiecare algoritm. O fabrică creează instanță potrivită în funcție de opțiunea aleasă, fără să fie nevoie să modificăm codul principal la fiecare algoritm nou.

Detalii legate de implementare:

- Interfață (ex.: `IRouteGenerator`) cu metoda `generate(params)`.
- Clase concrete: `CircularGenerator`, `RandomPointsGenerator`, etc.
- `RouteGeneratorFactory::create(type)` returnează generatorul corect.
- Codul principal lucrează doar cu interfața (`IRouteGenerator`), deci rămâne neschimbat.

6.3 Facade — (Razvan)

Problema identificată: Integrarea cu API-urile de hartă și generare trasee implică mai multe apeluri și pași (geocoding, request către serviciul de rutare, parsare răspuns, conversie în format pentru hartă, erori/retry). Dacă UI sau controller-ele folosesc direct aceste API-uri, codul devine împărațios, repetitiv și greu de întreținut.

Soluția (Facade): Se creează o singură interfață unificată care ascunde complexitatea API-urilor și expune metode simple (de nivel înalt) pentru aplicație.

Detalii legate de implementare:

- Se definește un serviciu (ex.: `RouteFacade`) care încapsulează:
 - accesul la cheia API / setări;
 - apelurile către serviciile de hartă (geocoding/tiles dacă e cazul);
 - apelurile către serviciul de rutare;
 - normalizarea rezultatului (ex.: polyline/GeoJSON + metadate).
- UI/Controller-ul apelează doar metode precum:
 - `generateRoute(params)`
 - `getRoutePreview(routeId/slug)`
 - `exportGeoJson(route)`
- Beneficiu direct: schimbarea furnizorului de rutare/hărți afectează doar `RouteFacade`, nu tot proiectul.

7 Scurt README

7.1 Structură proiect

- `/backend` — Laravel API (auth, rute, like, comentarii, feed)
- `/frontend` — Vue + Vite (UI, map, feed, route details)

7.2 Rulare locală (exemplu)

Backend (Laravel):

1. `cd backend`
2. `cp .env.example .env` (dacă nu există `.env`)
3. `composer install`
4. `php artisan key:generate`
5. `php artisan migrate`
6. `php artisan serve`

Frontend (Vue + Vite):

1. `cd frontend`
2. `npm install`
3. `npm run dev`

7.3 Funcții principale

- Generare rută + afișare pe hartă
- Salvare rută (cu slug)
- Partajare în feed
- Like/unlike + comentarii (cu actualizare UI)