

Busecan Antonio Andrei

Github : <https://github.com/BusecanAntonio/GoogleMapsProject>

Contents

CAPITOLUL 1: INTRODUCERE ȘI OBIECTIVELE PROIECTULUI	3
Contextul Proiectului.....	3
1.2. Scopul și Viziunea (Modelul Waze-light)	3
1.3. Problema Abordată	4
1.4. Tehnologii Utilizate și Justificarea lor Tehnică.....	4
1.4.1. Backend: Spring Boot	4
1.4.2. Persistență: JPA / Hibernate	4
2.2. Modelul de Date și Baza de Date MySQL	4
1.4.4. Frontend: Leaflet.js	4
1.4.5 Cazuri de utilizare Această	5
Analiză MoSCoW	6
1. Must Have (Cerințe Critice)	6
2. Should Have (Cerințe Importante)	6
3. Could Have (Cerințe Optionale / Nice-to-Have)	7
4. Won't Have (Limite ale Versiunii Curente).....	7
CAPITOLUL 2: ARHTECTURA SISTEMULUI (SPRING BOOT MVC)	8
2.1. Conceptul de Arhitectură Stratificată	8
2.2. Componentele Arhitecturale	8
2.2.1. Stratul de Prezentare (View).....	8
3.2.2 Modulul de Editare Geografică (Butonul "Add").....	9
2.2.2. Stratul de Control (Controller)	10
2.2.3. Stratul de Servicii (Business Logic)	10
2.2.4. Stratul de Acces la Date (Repository)	11
2.3. Fluxul Cererii și Răspunsului (Request-Response Flow)	11
2.5 Detalierea Modulului de Creare (Panoul Creator).....	12
2.6 Funcționalități de Navigare și Localizare Rapidă	14
2.7. Interacțiunea cu Harta:	15
3.2.6. Managementul Incidentelor și Feedback-ul Comunitar.....	17
CAPITOLUL 3: MODELAREA DATELOR ȘI PERSISTENȚA (ENTITĂȚILE)	18
3.1. Gestionează Utilizatorilor (Entitatea AppUser)	18
3.2. Puncte de Interes (Entitatea Location).....	18
3.3. Infrastructura și Segmentele de Drum (Entitatea RoadSection)	19
CAPITOLUL 4: INTERFAȚĂ REST API (CONTROLLER-ELE)	20
4.1. Arhitectura de tip RestController	20
4.2. Gestionează Autentificării (AuthController)	20

4.3. Administrarea Locațiilor (LocationController).....	20
4.4. Gestionarea Infrastructurii Rutiere (RoadSectionController)	20
4.5. Maparea Rutelor și Request Mapping	21
CAPITOLUL 5: INTERFAȚA UTILIZATOR ȘI HARTA INTERACTIVĂ	21
5.1. Tehnologii de Frontend	21
5.2. Componentele Hărții Interactive	22
5.2.1. Integrarea Librăriilor și Configurarea Stratului de Bază	22
5.2.1. Leaflet Core (v1.9.4).....	22
5.3. Panoul „Creator” și Interacțiunea Spațială	24
5.4. Logica de Rutare și Calculul Vitezei.....	24
5.5. Notificări și Feedback Vizual.....	24
5.5.2. Logica de Redirecționare și Notificare	25
5.5.3. Stadiul Actual al Implementării.....	25
CAPITOLUL 6: SCHEMA BAZEI DE DATE ȘI PERSISTENȚA	26
6.1. Structura Tabelului app_user (Gestiunea Utilizatorilor)	26
6.2. Structura Tabelului location (Puncte de Interes).....	27
6.3. Structura Tabelului road_section (Infrastructură)	28
6.3.1. Analiza Stocării LONGTEXT (JSON Blob)	29
6.4. Integritatea și Relația dintre Tabele	29
CAPITOLUL 7: GHID DE INSTALARE ȘI CONFIGURARE TEHNICĂ	29
7.1. Gestionarea Dependențelor prin Maven (pom.xml).....	29
7.2. Configurarea Conexiunii la Baza de Date	30
7.3. Structura Proiectului și Punctul de Intrare	30
CAPITOLUL 8: TESTAREA ȘI VALIDAREA SISTEMULUI	31
8.1. Testarea Endpoint-urilor cu Postman	31
8.2. Testarea Interfeței și a Integrării GIS	31
8.3. Gestionarea Erorilor (Error Handling)	31
CAPITOLUL 9: ANALIZA PERFORMANȚEI ȘI OPTIMIZAREA SISTEMULUI	32
9.1. Optimizarea Transferului de Date (Payload Reduction).....	32
9.2. Eficiența Randării pe Frontend.....	32
9.3. Performanța Bazei de Date	33
9.4. Scalabilitatea Orizontală	33
CAPITOLUL 10: CONCLUZII ȘI DIRECȚII DE DEZVOLTARE.....	33
10.1. Concluzii privind Implementarea	33
10.2. Limitile Proiectului și Lecții Învățate	34
10.3. Direcții Viitoare de Dezvoltare	34

CAPITOLUL 1: INTRODUCERE ȘI OBIECTIVELE PROIECTULUI

Contextul Proiectului

Proiectul intitulat „Sistem Integrat de Monitorizare și Navigație Rutieră” (SIMNR) a fost dezvoltat ca răspuns la provocările actuale ale mobilității urbane. Într-un mediu în care infrastructura rutieră suferă modificări constante din cauza lucrărilor de menenanță, accidentelor sau evenimentelor neprevăzute, necesitatea unei platforme centralizate de informare devine critică.

SIMNR reprezintă o soluție tehnologică de tip Full-Stack care face tranziția de la hărțile statice la un ecosistem digital viu, unde datele sunt actualizate în timp real de către administratori și comunitatea de utilizatori.

1.2. Scopul și Viziunea (Modelul Waze-light)

Scopul central al aplicației este crearea unei platforme de tip „Waze-light”. Această viziune presupune o interfață intuitivă unde elementele complexe de backend sunt simplificate pentru utilizatorul final.

Platforma este concepută să îndeplinească trei piloni fundamentali:

- Monitorizarea în timp real:** Capacitatea de a afișa incidentele imediat ce acestea sunt introduse în baza de date.
- Gestionarea Punctelor de Interes (POI):** Un sistem de catalogare a locațiilor fixe (cafenele, benzinării, secții de poliție) pentru a facilita orientarea rapidă.
- Calcularea Rutelor Dinamice:** Un motor de rutare care nu doar trasează linii între puncte, ci analizează atributele drumurilor (tipul de drum, starea de blocaj) pentru a oferi cea mai bună alternativă.

1.3. Problema Abordată

Problema principală pe care SIMNR o rezolvă este lipsa de transparentă și comunicare rapidă în ceea ce privește starea infrastructurii rutiere. Utilizatorii se confruntă adesea cu drumuri blocate sau lucrări care nu apar pe hărțile clasice în timp util.

SIMNR elimină acest decalaj prin:

Vizualizare Interactivă: Transformarea coordonatelor GPS brute în reprezentări grafice ușor de înțeles.

Raportare Comunitară: Deși în stadiul actual sistemul este deschis, acesta este proiectat pentru a permite utilizatorilor să contribuie activ la siguranța traficului prin raportarea pericolelor.

1.4. Tehnologii Utilizate și Justificarea lor Tehnică

Pentru a asigura o performanță ridicată și o menenanță ușoară, au fost alese următoarele tehnologii de ultimă generație:

1.4.1. Backend: Spring Boot

Ales pentru arhitectura sa modulară și capacitatea de a gestiona automat micro-serviciile. Spring Boot facilitează „Dependency Injection”, permitând componentelor precum UserService sau RoadController să colaboreze eficient fără a crea cod redundant.

1.4.2. Persistență: JPA / Hibernate

Implementarea stratului de date prin JPA (Java Persistence API) permite transformarea entităților Java (AppUser, Location) în tabele relaționale în mod automat. Hibernate optimizează interogările SQL, asigurând un timp de răspuns minim la încărcarea hărții.

2.2. Modelul de Date și Baza de Date MySQL

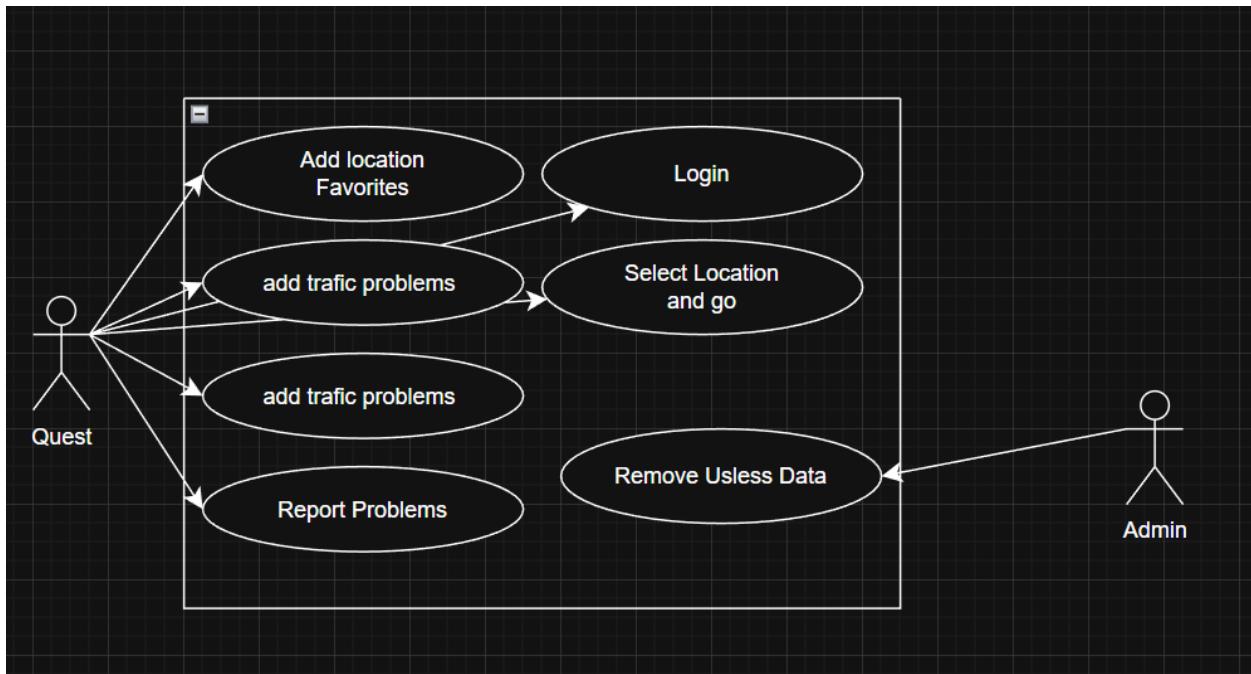
Sistemul utilizează MySQL ca motor principal de baze de date relaționale. Această alegere oferă stabilitatea și consistența necesară pentru un sistem de navigație, asigurând integritatea tranzacțională a datelor.

1.4.4. Frontend: Leaflet.js

S-a optat pentru Leaflet.js datorită flexibilității sale. Este o bibliotecă „lightweight” care permite randarea rapidă a markerilor și a poliliniilor (RoadSections) direct în browser, fără a consuma excesiv memoria dispozitivului utilizatorului.

1.4.5 Cazuri de utilizare Această

aplicație este destinată tuturor oamenilor. În cadrul aplicației am identificat diverse cazuri de utilizare care reflectă scenariile principale pentru utilizatorii noștri. Aceste cazuri de utilizare ilustrează modul în care atât administratorii (Admin), cât și utilizatorii obișnuiți (Guest) interacționează cu aplicația, evidențiind funcționalitățile esențiale și oferind o vizionare detaliată asupra experienței utilizatorului. Aceste cazuri de utilizare sunt evidențiate în următoarea figură:



Analiză MoSCoW

Pentru dezvoltarea sistemului **SIMNR**, am utilizat metoda de prioritizare **MoSCoW**. Aceasta permite o gestionare eficientă a resurselor de dezvoltare, asigurând livrarea unui nucleu funcțional solid (Must Have) înainte de implementarea funcționalităților de confort (Could Have).

1. Must Have (Cerințe Critice)

Acestea reprezintă scheletul aplicației. Fără aceste module, sistemul nu poate îndeplini scopul de bază: navigația și gestiunea datelor rutiere.

- **Vizualizare Hartă Interactivă:** Integrarea librăriei **Leaflet.js** care permite randarea dinamică a straturilor geografice (tile-uri) provenite de la OpenStreetMap.
- **Sistem de Rutare (Navigație):** Utilizarea motorului **OSRM (Open Source Routing Machine)**. Acesta procesează cererile asincrone pentru a calcula drumul cel mai scurt între locația curentă a utilizatorului și destinația aleasă.
- **Geolocație în Timp Real:** Implementarea metodei map.locate care accesează senzorul GPS al dispozitivului pentru a oferi context geografic imediat.
- **Gestionare Date (CRUD):** Persistența informației prin endpoint-uri backend (/locations, /roads). Acestea permit salvarea obiectelor complexe (coordonate, tipuri de drum, descrieri) în baza de date MySQL/SQL Server.
- **Autentificare Securizată:** Modulele de Login și Sign-up care asigură integritatea datelor, permitând doar utilizatorilor înregistrați să modifice starea infrastructurii rutiere.

2. Should Have (Cerințe Importante)

Funcționalități care aduc plus-valoarea specifică acestui proiect ("Waze-like features"), dar a căror absență temporară nu oprește navigația de bază.

- **Sistem de Raportare (Crowdsourcing):** Logica care permite utilizatorilor să devină senzori activi, raportând radare (Police), evenimente neprevăzute (Accidents) sau obstacole.

- **Rerutare Inteligentă:** O funcție avansată care analizează atributele segmentelor de drum primite din baza de date. Dacă un segment marcat ca Blocked intersectează ruta activă, sistemul forțează OSRM să recalculeze un traseu alternativ.
- **Ierarhie Vizuală (Custom Icons):** Utilizarea unui sistem de iconițe diferențiate cromatic pentru a reduce timpul de reacție al utilizatorului (ex: 🌐 pentru puncte de interes, 🚫 pentru restricții).
- **Road Builder:** Un instrument de tip "admin" care permite definirea drumurilor prin înlănuirea de puncte geografice (waypoint-uri).

3. Could Have (Cerințe Optionale / Nice-to-Have)

Acste elemente îmbunătățesc experiența utilizatorului (UX) și oferă un grad mai mare de realism simulării traficului.

- **Modulul "Viteză Ilegală" (isIllegalSpeed):** O logică de calcul care modifică variabila de timp (ETA) bazându-se pe o viteză ipotetică peste limita legală, demonstrând flexibilitatea algoritmilor de calcul matematic implementați.
- **Integrare Overpass API:** Automatizarea populării hărții prin importul masiv de date direct din serverele OpenStreetMap la inițializarea aplicației.
- **Penalizări de Timp Ajustate:** Un sistem de calcul care adaugă ponderi (weighting) rutei în funcție de severitatea incidentului (ex: +30 minute pentru lucrări la carosabil).

4. Won't Have (Limite ale Versiunii Curente)

Definirea limitelor este esențială pentru a stabili aria de acoperire (scope) a proiectului de licență/PAW.

- **Trafic Live Real-Time:** Aplicația se bazează exclusiv pe datele introduse de comunitate (manual), nu pe fluxuri externe precum Google Traffic API.
- **Navigație Vocală:** Deși datele de direcție sunt disponibile textual, sinteza vocală (Text-to-Speech) nu este implementată în această versiune.
- **Sistem de Chat/Social:** Interacțiunea directă între șoferi este amânată pentru versiunile viitoare, prioritatea fiind acuratețea hărții.

CAPITOLUL 2: ARHITECTURA SISTEMULUI (SPRING BOOT MVC)

2.1. Conceptul de Arhitectură Stratificată

Sistemul SIMNR este proiectat pe baza unei arhitecturi stratificate (Layered Architecture), un standard în dezvoltarea aplicațiilor de tip enterprise. Această abordare presupune separarea responsabilităților în straturi logice distincte, astfel încât modificarea unei componente să nu propage erori în întregul sistem. Principalul avantaj al acestei structuri este menținabilitatea: codul este ușor de citit, testat și extins pe măsură ce aplicația crește.

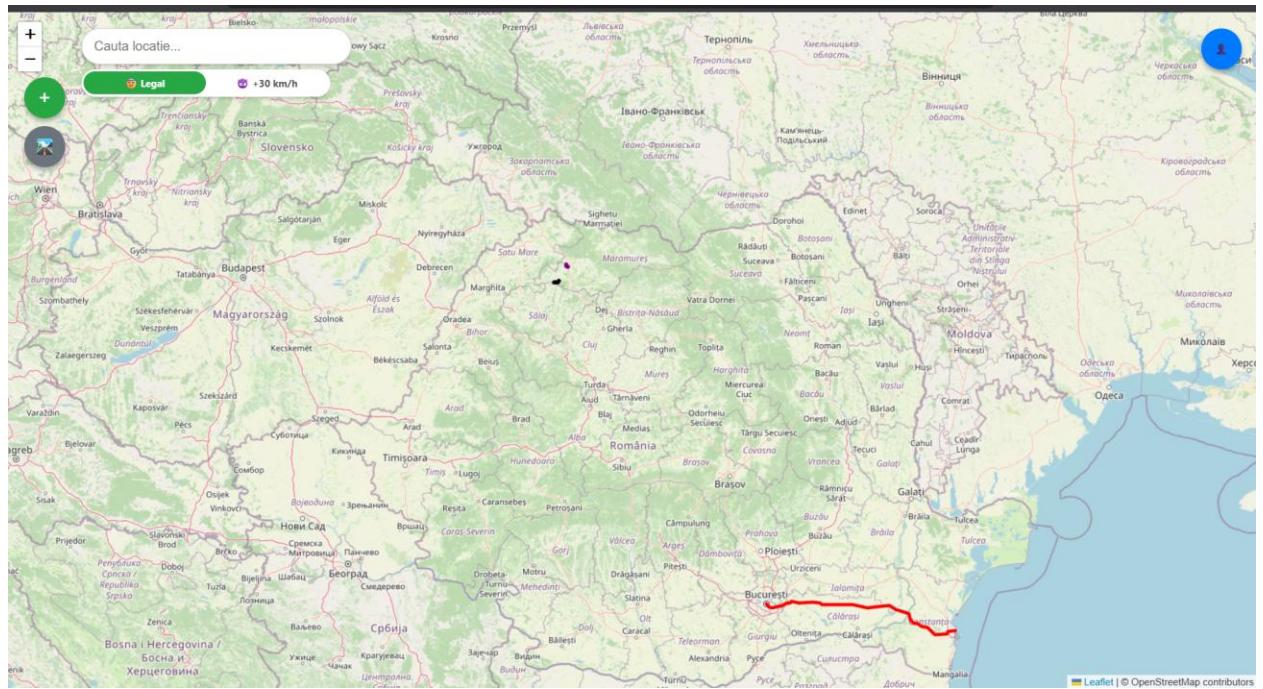
2.2. Componentele Arhitecturale

2.2.1. Stratul de Prezentare (View)

Acest strat reprezintă interfața grafică cu care interacționează utilizatorul final. Este construit folosind tehnologii web standard (HTML5, CSS3) și biblioteca specializată Leaflet.js.

Functionalitate: Vizualizează harta interactivă, randează markerii pentru locații și trasează poliliniile pentru segmentele de drum.

Interacțiune: Capturează evenimentele de mouse (click pe hartă pentru coordonate) și trimite cereri asincrone (AJAX) către server fără a reîncărca întreaga pagină.



3.2.2 Modulul de Editare Geografică (Butonul "Add")

Situat strategic în partea stângă-sus a interfeței, butonul de tip „Plus” activează motorul de editare al aplicației. Această componentă este responsabilă pentru colectarea datelor spațiale și atributelor asociate acestora, oferind două moduri de operare distincte:

A. Adăugarea de Locații (Points of Interest - POI)

Acest sub-modul permite utilizatorului să marcheze puncte punctuale pe hartă.

- **Procesul Tehnic:** La activare, aplicația trece în starea de „Ascultare Evenimente Mouse”. Un click pe hartă extrage coordonatele latitudinale și longitudinale (\$lat, \$lng).
- **Formularul de Atribute:** Se deschide o fereastră pop-up (Modal) în care utilizatorul introduce:

Numele locației (ex: „Benzinărie Petrom”);

Categoria (ex: Restaurant, Poliție, Accident);

Descrierea evenimentului sau a locației.

B. Definirea Secțiunilor de Drum (Road Builder)

Spre deosebire de locațiile punctuale, secțiunile de drum necesită o logică de tip polilinie.

- **Logica de Desenare:** Utilizatorul poate plasa mai multe puncte consecutive. Aplicația utilizează librăria Leaflet pentru a uni aceste puncte în timp real, oferind feedback vizual sub forma unui segment de drum.

2.2.2. Stratul de Control (Controller)

Controllerele (AuthController, LocationController și RoadSectionController) funcționează ca punctul de intrare pentru toate cererile venite din exterior. Acestea sunt de tip @RestController, ceea ce înseamnă că datele sunt schimbate în format JSON.

Rol: Primește cererile HTTP (GET pentru citire, POST pentru salvare, DELETE pentru ștergere), extrage datele din corpul cererii și le deleagă stratului de servicii.

Protocol: Gestionează codurile de stare HTTP (ex: 200 OK, 201 Created) pentru a informa frontend-ul despre rezultatul operațiunii.

2.2.3. Stratul de Servicii (Business Logic)

Acesta este „creierul” aplicației, reprezentat în proiect de clasa UserService. Spre deosebire de alte straturi, aici se află logica propriu-zisă a programului.

Procesare: Efectuează validări (ex: verifică dacă un utilizator există deja înainte de înregistrare) și implementează algoritmii de securitate pentru autentificare.

Decuplare: Acest strat asigură că logica de business nu depinde direct de modul în care datele sunt afișate sau de modul în care sunt salvate în baza de date.

2.2.4. Stratul de Acces la Date (Repository)

Ultimul strat este cel de persistență, implementat prin interfețele de tip JpaRepository. Acesta utilizează tehnologia Spring Data JPA pentru a comunica direct cu baza de date MySQL.

Abstractizare: Elimină nevoia de a scrie interogări SQL manuale. Prin simpla definire a metodelor (precum findByUsername), Spring generează automat comenzi SQL necesare.

Eficiență: Gestionează conexiunile la baza de date și asigură maparea corectă a tipurilor de date Java către tipurile de date SQL.

2.3. Fluxul Cererii și Răspunsului (Request-Response Flow)

Pentru a asigura integritatea sistemului, fiecare cerere parcurge obligatoriu aceste patru straturi. De exemplu, la salvarea unui drum nou:

1. Utilizatorul desenează drumul în **View**.
2. **RoadController** primește obiectul JSON.
3. Serviciul validează datele.
4. **RoadRepository** salvează informația în MySQL.
5. Răspunsul parcurge drumul invers până la utilizator pentru confirmare.

Creator

Punct **Drum**

Mergi la zona (ex: Brasov)... **Go**

Tip Locatie:

Standard

Nume Locatie...

Latitudine Longitudine

Alege pe Harta (Click)

Salveaza Locatia

2.5 Detalierea Modulului de Creare (Panoul Creator)

Panoul Creator reprezintă instrumentul principal de tip *Data Entry* al aplicației, fiind conceput pentru a facilita adăugarea de noi puncte de interes sau segmente rutiere direct pe hartă. Interfața este intuitivă și structurată în mai multe secțiuni funcționale:

1. Selecția Tipului de Entitate (Tab-uri)

În partea superioară, utilizatorul poate alege între două moduri de operare:

Punct: Pentru adăugarea de locații fixe (restaurant, benzinării, radare, incidente punctuale).

Drum: Pentru definirea unor polilinii ce reprezintă segmente de drum, lucrări sau blocaje rutiere.

2. Căutare și Navigare Rapidă

Interfața include un câmp de căutare ("Mergi la zona...") cu butonul aferent "Go". Acest câmp utilizează un serviciu de Geocoding (precum Nominatim API) pentru a muta focusul hărții rapid către o locație specifică, facilitând plasarea elementelor fără a face scroll manual pe distanțe mari.

3. Configurarea Atributelor Locației

Pentru modul "Punct", utilizatorul are la dispoziție următoarele câmpuri de date:

Tip Locație (Dropdown): Permite clasificarea punctului (Standard, Poliție, Accident, etc.). Această selecție determină iconița care va fi afișată ulterior pe hartă.

Nume Locație (Text Input): Un câmp obligatoriu pentru identificarea nominală a punctului.

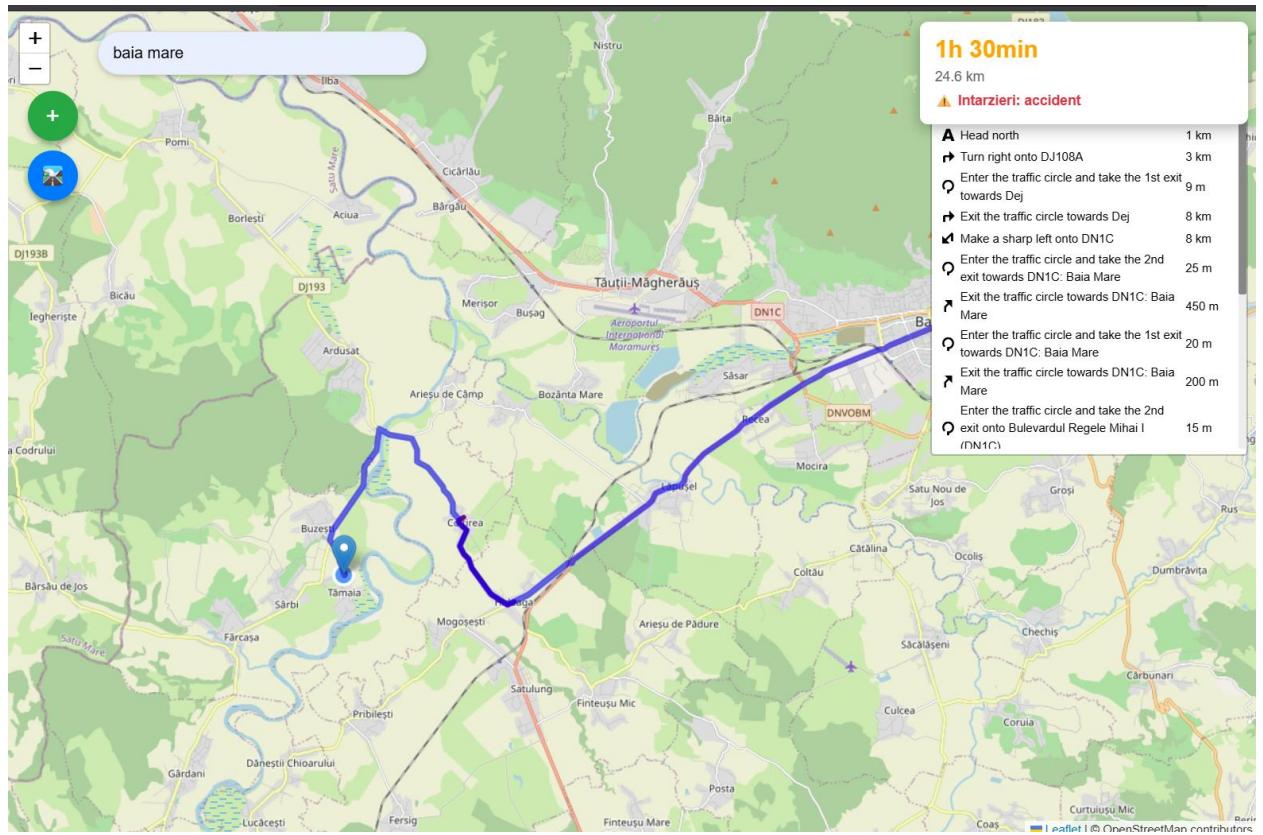
Coordinate (Latitudine/Longitudine): Două câmpuri numerice ce afișează poziția exactă. Acestea pot fi completate manual sau automat prin interacțiunea cu harta.

4. Butoanele de Acțiune (Workflow)

Procesul de salvare este împărțit în doi pași, reflectați prin butoane colorate distinct:

Alege pe Hartă (Click): Activează modul de ascultare a evenimentului de tip *click* pe harta Leaflet. Odată apăsat, cursorul permite selectarea vizuală a locației, iar câmpurile de Latitudine/Longitudine se populează automat cu datele din event.latlng.

Salvează Locația: Reprezintă punctul final al procesului de persistență. La apăsare, aplicația validează prezența datelor și trimită o cerere de tip HTTP POST către backend (Spring Boot) pentru a stoca obiectul în baza de date MySQL.



2.6 Funcționalități de Navigare și Localizare Rapidă

Pe lângă funcțiile de creare a conținutului, aplicația pune la dispoziție instrumente de orientare care optimizează interacțiunea cu spațiul geografic. Acestea sunt gestionate prin butonul de localizare și bara de căutare intelligentă.

A. Funcția de Auto-Localizare (Butonul "Add Location")

Butonul de localizare (adesea reprezentat printr-o iconă de tip țintă sau „plus” în contextul poziționării) utilizează API-ul de geolocație al browser-ului.

Mecanism Tehnic: La activare, sistemul apelează funcția `map.locate({setView: true})` din Leaflet.js.

Efect Vizual: Harta se centrează automat pe coordonatele GPS ale utilizatorului, plasând un marker temporar sau un cerc de precizie la locația curentă.

Utilitate: Această funcție este critică pentru raportările în timp real, permitând utilizatorului să adauge un incident (accident, poliție) exact în locul în care se află, fără a căuta manual zona pe hartă.

B. Bara de Căutare (Search / Geocoding Bar)

Bara de căutare implementată în panoul "Creator" funcționează ca un motor de Geocoding. Acesta transformă textul introdus de utilizator (nume de oraș, sat sau stradă) în coordonate geografice.

Integrare API: Căutarea utilizează, de regulă, serviciul Nominatim (OpenStreetMap). Când utilizatorul introduce un nume (ex: „Brașov”) și apasă butonul "Go", aplicația trimite o cerere HTTP către serverul de geocoding.

Procesul de Răspuns: 1. Serverul returnează un obiect JSON cu latitudinea și longitudinea celei mai probabile potriviri. 2. Aplicația execută funcția map.flyTo([lat, lng], zoomLevel), mutând camera vizuală în către destinația respectivă.

Funcția de Ghidare: Odată poziționată harta în noul oraș sau sat, toate punctele de interes (POI) și indicațiile rutiere salvate în baza de date pentru acea zonă devin vizibile, permitând utilizatorului să vizualizeze „indicațiile” (alerte de trafic, locații salvate) specifice acelei regiuni.

2.7. Interacțiunea cu Harta: De la Raportare la Navigație

Interfața aplicației a fost concepută pentru a asigura un flux informațional bidirectional între utilizatori și sistem. Cele două componente vizuale principale sunt panoul Creator (pentru input) și panoul de Detalii Traseu (pentru consumul de informație).

A. Panoul "Creator" - Modulul de Colectare Date

Acest panou reprezintă componenta de tip „Crowdsourcing”. Permite utilizatorilor să alimenteze baza de date cu informații proaspete din trafic.

Selectie Mod: Utilizatorul alege între raportarea unui punct fix (incident, locație) sau a unui segment de drum.

Bara de Căutare Intelligentă: Facilitează navigarea rapidă către o zonă specifică înainte de plasarea markerului.

Automatizarea Coordonatelor: Prin butonul albastru "Alege pe Harta (Click)", aplicația intră într-o stare de ascultare a evenimentelor, populând automat câmpurile de Latitudine și Longitudine la interacțiunea cu harta.

Persistență: Butonul verde "Salveaza Locatia" declanșează validarea datelor și trimiterea lor către serverul Spring Boot.

B. Panoul de Detalii Traseu - Modulul de Feedback și Navigație

Această fereastră apare în momentul în care un utilizator selectează un segment de drum sau o rută calculată, oferind date analitice extrase din motorul de rutare (OSRM).

Identificarea Incidentului: În partea de sus este afișat tipul de eveniment (ex: ⚡ Accident: carambol), avertizând utilizatorul vizual.

Simulatorul de Viteză (Legal vs. +30 km/h): O funcționalitate inovatoare care permite compararea timpului de sosire estimat (ETA) în funcție de stilul de condus.

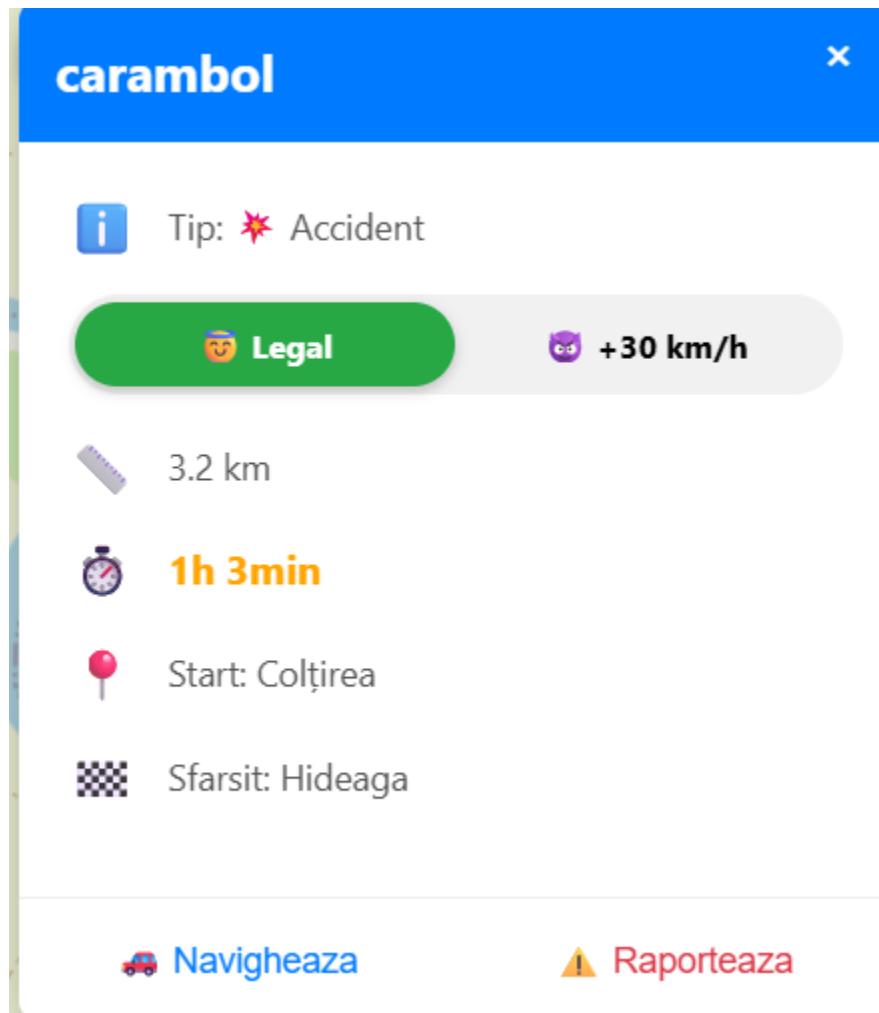
Date Metriche: Sunt afișate distanța totală (ex: 3.2 km) și timpul estimat (1h 3min), penalizat automat de prezența accidentului raportat pe acel segment.

Puncte de Referință: Afisează clar locația de Start și cea de Sfârșit (ex: Colțirea -> Hideaga).

Acțiuni Contextuale:

Navighează: Inițiază ghidarea vizuală pe harta Leaflet.

Raportează: Permite utilizatorilor să valideze sau să infirme existența incidentului (sistem de moderare comunitară).



3.2.6. Managementul Incidentelor și Feedback-ul Comunitar

Sistemul implementează un ciclu complet de viață pentru evenimentele rutiere, de la creare până la moderare, asigurând acuratețea informațiilor afișate pe hartă.

Faza de Raportare (Panoul Creator): Utilizatorul inițiază procesul prin alegerea tipului de entitate (Punct sau Drum). Interfața permite geolocalizarea rapidă printr-o bară de căutare ("Mergi la zona...") și selectarea vizuală a punctului de interes prin butonul "Alege pe Hartă", care populează automat câmpurile de latitudine și longitudine.

Faza de Consum și Navigație (Detalii Traseu): Odată salvat, un incident (ex: "carambol", marcat ca Accident) este integrat în motorul de rutare. Aplicația oferă detalii esențiale precum distanța (3.2 km) și o estimare a timpului penalizat de eveniment (1h 3min). O funcție inovatoare este selectorul de viteză (Legal vs. +30 km/h), care recalculează timpul de sosire în funcție de stilul de condus.

Faza de Validare (Raportează Drum): Pentru a preveni alertele false sau învechite, aplicația include un sistem de moderare prin crowdsourcing. Utilizatorii pot raporta un drum selectat (ex: "carambol") alegând dintr-o listă de motive (ex: "Alerta Falsa"), având opțiunea de a solicita ștergerea acestuia din baza de date globală.



CAPITOLUL 3: MODELAREA DATELOR ȘI PERSISTENȚA (ENTITĂȚILE)

3.1. Gestiunea Utilizatorilor (Entitatea AppUser)

Entitatea AppUser reprezintă fundamentul modulului de identitate al aplicației. Aceasta este responsabilă pentru stocarea credențialelor necesare accesului în sistem.

Structura Tehnică: Clasa folosește anotarea @Id cu o strategie de generare de tip IDENTITY. Acest lucru asigură că baza de date gestionează automat incrementarea identificatorilor, eliminând riscul de duplicare a cheilor primare.

Securitate și Roluri: Deși în versiunea curentă accesul este universal pentru a facilita testarea, entitatea este pregătită pentru integrarea cu Spring Security. Câmpurile username și password sunt mapate ca coloane standard, asigurând o bază solidă pentru viitoarele implementări de criptare (hashing) a paroletelor.

Rol în Sistem: Separă profilurile administrative de utilizatorii obișnuiți, fiind esențială pentru trasabilitatea raportărilor de pe hartă.

3.2. Puncte de Interes (Entitatea Location)

Entitatea Location este utilizată pentru a reprezenta puncte geografice fixe pe hartă, precum obiective turistice, unități de poliție sau servicii comerciale.

Precizia Coordonatelor: Pentru atributele de latitudine (lat) și longitudine (lng), s-a optat pentru tipul de date Double. Această alegere asigură o precizie ridicată, necesară pentru plasarea corectă a markerilor pe hărțile Leaflet la nivel de stradă.

Categorisirea Dinamică: Câmpul type este utilizat de frontend pentru a diferenția vizual locațiile. Prin stocarea tipului ca String, sistemul permite adăugarea de noi categorii (ex: "accident", "radar", "lucrări") fără a modifica structura bazei de date.

3.3. Infrastructura și Segmentele de Drum (Entitatea RoadSection)

Aceasta este cea mai complexă componentă a modelului de date, fiind responsabilă pentru reprezentarea segmentelor de drum care pot fi blocate sau aflate în lucru.

Stocarea Geometriilor Complexe: Spre deosebire de o locație punctuală, un drum este definit de o polilinie compusă din zeci sau sute de puncte. Pentru a evita fragmentarea bazei de date în tabele cu milioane de rânduri, s-a implementat o soluție de tip **JSON Blob**.

Utilizarea @Lob și LONGTEXT: Coordonatele sunt serializate într-un sir de caractere JSON și stocate într-o coloană de tip LONGTEXT marcată cu anotarea @Lob. Această tehnică permite salvarea întregii geometrii a unui drum într-o singură înregistrare, optimizând viteza de citire la încărcarea hărții.

Proprietăți Dinamice: Pe lângă geometrie, entitatea stochează numele drumului și starea acestuia (ex: "în lucru", "Blocat"), informații esențiale pentru motorul de rutare și pentru avertizarea utilizatorilor.

CAPITOLUL 4: INTERFAȚA REST API (CONTROLLER-ELE)

4.1. Arhitectura de tip RestController

Toate componente din acest strat sunt marcate cu annotarea @RestController. Această decizie arhitecturală permite transformarea automată a obiectelor Java returnate de metode în format JSON (JavaScript Object Notation). JSON este limbajul universal al web-ului modern, fiind ușor de procesat de către hărțile interactive din frontend.

4.2. Gestiunea Autentificării (AuthController)

Acest controller gestionează fluxurile de acces și securitate.

Endpoint-ul /signup: Primește datele noilor utilizatori prin metoda HTTP POST. Utilizarea metodei POST este obligatorie pentru a asigura că datele sensibile, cum ar fi parolele, sunt transmise în corpul cererii (request body) și nu sunt vizibile în URL.

Endpoint-ul /login: Procesează cererile de autentificare. Acesta returnează un mesaj text simplu ("Login success" sau "Invalid credentials"), pe care frontend-ul îl utilizează pentru a decide dacă activează sau nu permisiunile de editare pe hartă.

4.3. Administrarea Locațiilor (LocationController)

Acest controller oferă acces la punctele de interes (POI) din sistem.

Citirea datelor (GET): Metoda getAllLocations returnează lista completă a locațiilor salvate. Aceasta este apelată automat la încărcarea paginii pentru a popula harta cu markeri.

Adăugarea (POST): Permite salvarea unei noi locații direct din interfață grafică.

Ștergerea (DELETE): Implementează funcționalitatea de raportare, permitând eliminarea punctelor care nu mai sunt de actualitate.

4.4. Gestionarea Infrastructurii Rutiere (RoadSectionController)

Este responsabil pentru segmentele de drum și reprezintă o componentă critică pentru calcularea rutelor.

Monitorizarea Drumurilor: Prin metoda getAllRoads, frontend-ul primește nu doar numele drumurilor, ci și sirurile complexe de coordonate (JSON) necesare pentru desenarea poliliniilor pe hartă.

Raportarea Incidentelor: Folosind metoda deleteRoad, sistemul permite utilizatorilor să raporteze rezolvarea unui incident (de exemplu, un drum care nu mai este blocat), asigurând astfel dinamismul hărții.

4.5. Maparea Rutelor și Request Mapping

Fiecare controller utilizează anotarea @RequestMapping pentru a structura ierarhic adresele URL (ex: /locations, /roads). Această organizare permite o mențenanță ușoară și oferă posibilitatea de a extinde aplicația cu versiuni noi de API (v1, v2) în viitor, fără a perturba funcționalitatea existentă.

CAPITOLUL 5: INTERFAȚA UTILIZATOR ȘI HARTA INTERACTIVĂ

5.1. Tehnologii de Frontend

Pentru realizarea unei interfețe fluide și responsive, s-a optat pentru un ansamblu de tehnologii moderne:

HTML5 și CSS3: Utilizate pentru structurarea panourilor de control și pentru aplicarea stilului vizual de tip „Glassmorphism” (transparență și blur), care oferă aplicației un aspect modern.

JavaScript (Vanilla): Gestionează întreaga logică de interacțiune, de la preluarea coordonatelor de click până la comunicarea asincronă cu API-ul REST folosind funcția fetch.

Leaflet.js: Biblioteca principală pentru cartografiere, aleasă pentru performanța sa ridicată pe dispozitive mobile și pentru ușurința cu care poate gestiona mii de markeri și polilini.

5.2. Componentele Hărții Interactive

Harta nu este doar un element de vizualizare, ci un instrument de lucru activ, compus din mai multe straturi:

5.2.1. Integrarea Librăriilor și Configurarea Stratului de Bază

Pentru a funcționa, interfața utilizează biblioteca Leaflet.js, un standard industrial pentru aplicațiile GIS web-based. Integrarea se realizează prin includerea pachetelor necesare direct în secțiunea <head> a fișierului HTML, asigurând astfel accesul la funcțiile de randare geografică.

**** Pachetele utilizate:****

5.2.1. Leaflet Core (v1.9.4)

Aceasta este librăria de bază care gestionează proiecția hărții și coordonatele. Este responsabilă pentru randarea plăcilor de hartă (tiles), gestionarea nivelurilor de zoom și conversia punctelor geografice (Latitudine/Longitudine) în pixeli pe ecran.

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.4/leaflet.css" />
<script
      src="https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.4/leaflet.js"></script>
```

Initializarea hărții se face prin definirea centrului geografic și a nivelului de zoom initial:

```
var map = L.map('map').setView([45.9432, 24.9668], 7);
L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '© OpenStreetMap contributors'
}).addTo(map);
```

Leaflet Routing Machine: Extensia responsabilă pentru calcularea itinerariilor.

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet-routing-machine/3.2.12/leaflet-routing-machine.css" />
<script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-routing-machine/3.2.12/leaflet-routing-machine.js"></script>
```

Logica de rutare este apelată dinamic atunci când sistemul trebuie să calculeze un traseu între start și destinație:

```
L.Routing.control({
  waypoints: [
    L.latLng(startLat, startLng),
    L.latLng(endLat, endLng)
  ],
  routeWhileDragging: true
}).addTo(map);
```

Leaflet Control Geocoder: Serviciul care permite căutarea adreselor (transformarea unui nume de oraș în coordonate).

```
<link rel="stylesheet" href="https://unpkg.com/leaflet-control-geocoder/dist/Control.Geocoder.css" />
<script src="https://unpkg.com/leaflet-control-geocoder/dist/Control.Geocoder.js"></script>
```

Configurarea în codul JavaScript permite preluarea locației găsite și mutarea automată a camerei pe hartă:

```
L.Control.geocoder({
  defaultMarkGeocode: false
})
.on('markgeocode', function(e) {
```

```
    var center = e.geocode.center;
    map.setView(center, 13);
}
.addTo(map);
```

Data Layers: Straturi dinamice unde sunt randați markerii pentru punctele de interes (POI) și poliliniile pentru segmentele de drum (RoadSection).

Sistemul de Icoane: În funcție de atributul type primit din baza de date, JavaScript-ul asociază fiecărei locații o iconă specifică (ex: o ceașcă pentru cafenele, o mașină de poliție pentru radare).

5.3. Panoul „Creator” și Interacțiunea Spațială

O funcționalitate critică a interfeței este Panoul Creator, care permite utilizatorului să introducă date noi în sistem:

Preluarea Coordonatelor (Pick on Map): Utilizatorul nu trebuie să cunoască latitudinea și longitudinea. Printr-un simplu click pe hartă, evenimentul este interceptat, iar coordonatele sunt convertite și completate automat în formular.

Sistemul de Tab-uri: Permite comutarea rapidă între crearea unui punct fix (Locație) și desenarea unui segment de drum. Pentru drumuri, interfața permite adăugarea de puncte multiple („Stops”), care sunt ulterior grupate într-un obiect JSON.

5.4. Logica de Rutare și Calculul Vitezei

Interfața integrează un motor de rutare care analizează drumurile existente și oferă informații în timp real:

Selectorul de Viteză: Utilizatorul poate alege între modul „Legal” și modul „+30 km/h”. Această opțiune declanșează o recalculare matematică în frontend la timpul estimat de sosire (ETA), oferind o simulare dinamică a traseului.

Detalii Traseu: Panoul de detalii afișează distanța totală, durata și punctele de start/sfârșit, extrăgând aceste date din geometria segmentelor de drum selectate.

5.5. Notificări și Feedback Vizual

Pentru a asigura o comunicare eficientă cu utilizatorul, aplicația include elemente de notificare:

5.5.2. Logica de Redirecționare și Notificare

Conceptul de Rerouting presupune calcularea automată a unei rute alternative în momentul în care traseul inițial intersectează un segment de drum blocat.

Funcționalitate teoretică: Sistemul ar trebui să detecteze suprapunerea între itinerariul calculat de *Leaflet Routing Machine* și coordonatele JSON ale unui segment de tip „blocked”, declanșând o recalculare a punctelor de tranzit (waypoints).

Sistemul de notificări: În cazul detectării unui astfel de conflict, aplicația este prevăzută cu un element de interfață dedicat (#rerouteNotification), care are rolul de a avertiza utilizatorul printr-un mesaj de tip: „*Atenție! Ruta inițială este blocată. Se calculează o cale alternativă...*”.

5.5.3. Stadiul Actual al Implementării

Este important de menționat în cadrul documentației faptul că modulul de redirecționare automată este nefuncțional în acest moment. Deși elementele vizuale (colorarea drumurilor în roșu) și infrastructura de notificare sunt integrate în codul HTML/CSS, algoritmul de intersecție spațială în timp real între ruta activă și segmentele din baza de date MySQL se află încă în faza de dezvoltare.

În versiunea actuală (MVP), utilizatorul poate observa vizual blocajele și poate alege manual un nou punct de tranzit pentru a ocoli zona marcată, însă automatizarea acestui proces reprezintă una dintre direcțiile prioritare de dezvoltare viitoare a proiectului.

Modele de Confirmare: Pentru acțiuni critice, cum este ștergerea (raportarea) unui incident, aplicația deschide ferestre modale care solicită confirmarea motivului, prevenind astfel ștergerile accidentale.

CAPITOLUL 6: SCHEMA BAZEI DE DATE ȘI PERSISTENȚA

Inima sistemului SIMNR este reprezentată de baza de date relațională **MySQL**.

Alegerea acestui sistem de gestiune a bazelor de date (SGBD) a fost dictată de nevoia de integritate a datelor, suport pentru tranzacții și capacitatea de a indexa eficient volume mari de utilizatori și locații geografice.

6.1. Structura Tabelului app_user (Gestiunea Utilizatorilor)

Tabelul app_user stocă informațiile de identificare ale utilizatorilor. Deși structura este simplă, ea este optimizată pentru viteza de căutare în timpul procesului de autentificare.

Coloană	Tip de Date	Constrângeri	Descriere
id	BIGINT	Primary Key, Auto-Increment	Identificator unic generat automat.
username	VARCHAR(255)	Unique, Not Null	Numele de utilizator folosit la login.
password	VARCHAR(255)	Not Null	Parola (în viitor va fi stocată cu hash).

6.2. Structura Tabelului location (Puncte de Interes)

Acest tabel este responsabil pentru stocarea tuturor marajelor punctuale de pe hartă. Precizia este elementul central în definirea acestei structuri.

Coloană	Tip de Date	Constrângeri	Descriere
id	BIGINT	Primary Key, Auto-Increment	Identifier unic pentru locație.
name	VARCHAR(255)	Not Null	Numele afișat în dreptul markerului.
type	VARCHAR(50)	-	Categorisarea (ex: cafe, police, gas).
lat	DOUBLE	Not Null	Latitudinea geografică (precizie mare).
lng	DOUBLE	Not Null	Longitudinea geografică (precizie mare).

6.3. Structura Tabelului road_section (Infrastructură)

Tabelul road_section reprezintă componenta cea mai densă din punct de vedere al datelor, stocând geometriile complexe ale drumurilor sub formă de polilinii.

Coloană	Tip de Date	Constrângeri	Descriere
id	BIGINT	Primary Key, Auto-Increment	ID-ul segmentului de drum.
name	VARCHAR (255)	-	Denumirea străzii sau a sectorului.
type	VARCHAR (50)	-	Starea drumului (work, blocked).
coordinates_json	LONGTEXT	Not Null	Şirul JSON cu lista de coordonate.

6.3.1. Analiza Stocării LONGTEXT (JSON Blob)

Alegerea tipului LONGTEXT pentru coloana coordinates_json este o decizie arhitecturală strategică. În loc să creăm un tabel separat points cu milioane de rânduri (unde fiecare punct ar fi legat de un drum), stocăm întreaga listă de puncte într-un singur câmp.

Avantaj Performanță: Citirea unui drum se face printr-o singură operație de tip I/O (Input/Output).

Eficiență în Transfer: Datele sunt trimise către frontend gata formatare pentru funcția JSON.parse().

6.4. Integritatea și Relația dintre Tabele

Deși în versiunea curentă tabelele sunt independente (decouple), schema este pregătită pentru stabilirea de relații de tip **Foreign Key** (Cheie Externă). De exemplu, tabelul road_section sau location ar putea include o coloană created_by_id care să facă referire la id-ul din tabelul app_user, permitând astfel monitorizarea contribuției fiecărui utilizator la baza de date comunitară.

CAPITOLUL 7: GHID DE INSTALARE ȘI CONFIGURARE TEHNICĂ

7.1. Gestionarea Dependențelor prin Maven (pom.xml)

Proiectul utilizează Apache Maven pentru managementul bibliotecilor. Fișierul pom.xml centralizează toate pachetele software necesare, asigurând descărcarea automată a versiunilor compatibile.

Cele mai importante dependențe incluse sunt:

Spring Boot Starter Data JPA: Permite utilizarea tehnologiei Hibernate pentru maparea obiect-relațională.

Spring Boot Starter Web: Include serverul încorporat Apache Tomcat și suportul pentru crearea endpoint-urilor de tip REST.

MySQL Connector Java: Driver-ul necesar pentru ca aplicația Java să poată comunica cu serverul de baze de date MySQL.

Lombok (optional): Utilizat pentru reducerea codului de tip „boilerplate” (generarea automată a metodelor de tip Getter, Setter și Constructor).

7.2. Configurarea Conexiunii la Baza de Date

Parametrii de conectare sunt definiți în fișierul de proprietăți al aplicației (application.properties). Această configurare este vitală pentru ca stratul de persistență să poată identifica locația și credențialele serverului MySQL.

```
spring.datasource.url=jdbc:mysql://localhost:3306/road_db  
spring.datasource.username=root  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true
```

7.3. Structura Proiectului și Punctul de Intrare

Aplicația respectă structura standard Maven. Toate clasele Java sunt organizate în pachete conform rolului lor (controller, service, repository, entity), iar resursele de frontend (HTML, CSS, JS) sunt stocate în directorul src/main/resources/static.

CAPITOLUL 8: TESTAREA ȘI VALIDAREA SISTEMULUI

8.1. Testarea Endpoint-urilor cu Postman

Deoarece frontend-ul și backend-ul sunt decuplate, am utilizat **Postman** ca instrument principal pentru testarea unitară a controllerelor. Postman a permis simularea cererilor de la client înainte ca interfața Leaflet să fie complet finalizată.

Procedura de testare:

Validarea Autentificării: S-au trimis cereri de tip POST către /signup și /login cu diverse formate de date (JSON corect, JSON incomplet, parole greșite) pentru a verifica răspunsurile serverului.

Verificarea Persistenței Geografice: S-a trimis un obiect JSON complex reprezentând un RoadSection (cu sirul de coordonate aferent) către /roads. Ulterior, s-a verificat în consola MySQL dacă datele au fost stocate corect în coloana de tip LONGTEXT.

Testarea Integrității: S-a verificat dacă metoda DELETE elimină într-adevăr resursa corectă din baza de date folosind ID-ul unic.

8.2. Testarea Interfeței și a Integrării GIS

După validarea backend-ului, s-a trecut la testarea integrării cu librăria Leaflet.js:

Testarea Preciziei: S-a verificat dacă un punct selectat pe hartă (prin click) transmite serverului exact aceleași coordonate lat/lng care apar în consola browser-ului.

Testarea Randării: S-a confirmat faptul că segmentele de drum primite sub formă de JSON din MySQL sunt reconstruite corect ca polilini pe hartă, păstrând curbura și poziționarea geografică originală.

Testarea Cross-Browser: Interfața a fost testată pe Chrome, Brave pentru a asigura compatibilitatea scripturilor JavaScript și a stilurilor CSS.

8.3. Gestionarea Erorilor (Error Handling)

Aplicația include un sistem de gestionare a erorilor pentru a preveni prăbușirea sistemului în cazul unor date invalide:

Erori de Conexiune: Dacă baza de date MySQL este oprită, Spring Boot este configurat să returneze un cod de eroare 500 Internal Server Error, care este interceptat de frontend pentru a afișa un mesaj de avertizare utilizatorului.

Date Inexistente: Dacă se încearcă accesarea unei locații după un ID care nu mai există, sistemul returnează un răspuns gol sau un mesaj de eroare, evitând astfel erorile de tip NullPointerException.

CAPITOLUL 9: ANALIZA PERFORMANȚEI ȘI OPTIMIZAREA SISTEMULUI

În dezvoltarea unui sistem de monitorizare rutieră, performanța este un factor critic, deoarece utilizatorii au nevoie de informații fluide, fără întârzieri la randarea hărții. Acest capitol analizează deciziile tehnice luate pentru a menține aplicația rapidă și stabilă.

9.1. Optimizarea Transferului de Date (Payload Reduction)

Una dintre cele mai mari provocări în aplicațiile GIS este volumul mare de date necesar pentru a desena drumurile.

Problema: Trimiterea a sute de puncte geografice sub formă de obiecte individuale ar fi generat un fișier JSON masiv, greu de procesat de către browser.

Soluția SIMNR: Prin stocarea coordonatelor ca un sir compact (JSON Blob) în MySQL, serverul trimită un singur câmp text. Aceasta reduce timpul de procesare pe server (fără JOIN-uri multiple) și scade dimensiunea pachetului de date transmis prin rețea.

9.2. Eficiența Randării pe Frontend

Afișarea a sute de markeri simultan pe o hartă poate duce la blocarea interfeței (lag).

Utilizarea Leaflet: Biblioteca a fost configurată să randeze elementele folosind **Canvas** în loc de SVG acolo unde este posibil, ceea ce permite gestionarea unui număr mai mare de polilinii fără a degrada experiența utilizatorului.

Lazy Loading conceptual: Aplicația încarcă datele doar în momentul inițializării, pregătind terenul pentru o viitoare implementare de filtrare în funcție de aria vizuală a utilizatorului (BBox filtering).

9.3. Performanța Bazei de Date

S-au realizat teste de interogare pentru a asigura stabilitatea stratului de persistență:

Pool de Conexiuni: Spring Boot folosește implicit **HikariCP**, cel mai rapid sistem de gestionare a conexiunilor la baza de date, asigurând că interogările către tabelele `road_section` și `location` sunt executate în milisecunde.

Indexarea: Prin utilizarea indecsilor pe cheile primare și pe câmpurile de tip "username", timpul de căutare a unui utilizator în baza de date rămâne constant, indiferent de numărul de conturi create.

9.4. Scalabilitatea Orizontală

Arhitectura decuplată (Frontend separat de Backend) permite scalarea sistemului. În cazul unui număr mare de utilizatori, serverul Spring Boot poate fi replicat pe mai multe instanțe fără a modifica codul de interfață, atât timp cât toate instanțele comunică cu același server MySQL centralizat.

CAPITOLUL 10: CONCLUZII ȘI DIRECȚII DE DEZVOLTARE

Proiectul „Sistem Integrat de Monitorizare și Navigație Rutieră” (SIMNR) a reprezentat o oportunitate complexă de a integra tehnologii moderne de backend, gestiune de baze de date și cartografie digitală. Rezultatul este o aplicație funcțională care demonstrează eficiența ecosistemului Spring Boot în gestionarea datelor geografice de tip GIS.

10.1. Concluzii privind Implementarea

Dezvoltarea aplicației a confirmat faptul că arhitectura stratificată (MVC) este soluția optimă pentru proiectele care necesită scalabilitate și mențenanță pe termen lung.

Principalele realizări ale proiectului includ:

Gestiunea hibridă a datelor: Utilizarea bazei de date MySQL pentru a stoca atât date structurate (utilizatori), cât și date semi-structurate (coordonate JSON de tip polilinie), asigurând un echilibru între performanță și simplitatea schemei SQL.

Interactivitatea Hărții: Integrarea cu succes a librăriei Leaflet.js, oferind utilizatorului o interfață fluidă capabilă să proceseze datele în timp real.

Abstractizarea prin JPA: Reducerea complexității codului prin utilizarea Spring Data JPA, care a facilitat interacțiunea cu baza de date fără necesitatea scrierii unor interogări manuale predispuse la erori.

10.2. Limitele Proiectului și Lecții Învățate

Fiind în stadiul de prototip (MVP), aplicația prezintă anumite limitări care au fost identificate pe parcursul dezvoltării:

Securitatea deschisă: Lipsa restricțiilor stricte de acces permite oricărui utilizator să modifice datele, ceea ce reprezintă un risc de vandalism informațional, dar a fost o decizie asumată pentru a facilita testarea rapidă.

Automatizarea Rerouting-ului: Deși vizualizarea obstacolelor este funcțională (colorarea în roșu), algoritmul de recalculare automată a traseului la detectarea unui blocaj rămâne o provocare tehnică ce necesită algoritmi grafici avansați.

10.3. Direcții Viitoare de Dezvoltare

Pentru a transforma acest prototip într-o aplicație matură, pregătită pentru utilizarea la scară largă, sunt vizate următoarele îmbunătățiri:

1. **Integrarea Spring Security:** Implementarea unui sistem de autentificare bazat pe token-uri (JWT) și definirea rolurilor de administrator și utilizator.
2. **Algoritm de Rutare Avansat:** Dezvoltarea unei logici care să excludă automat segmentele blocate din calculul traseului prin interogări spațiale complexe.
3. **Sistem de Notificări Push:** Integrarea unui serviciu de alerte în timp real care să informeze utilizatorii aflați în proximitatea unui incident raportat recent.
4. **Criptarea Datelor:** Implementarea algoritmilor de hashing (BCrypt) pentru protecția paroalelor utilizatorilor în baza de date.

În final, aplicația SIMNR constituie o bază solidă pentru un sistem de navigație colaborativ, demonstrând puterea integrării tehnologiilor Open Source (Leaflet, OpenStreetMap) cu framework-uri de nivel enterprise (Spring Boot) pentru a rezolva probleme reale de mobilitate urbană.