RECYCLING PLATFORM

A Web Application for Better Recycling

Autori

Strimbei Daria

Musatoiu Iulian

Flinta Ovidiu

ABSTRACT

This thesis presents the design and implementation of a distributed web application for managing recycling locations and user-generated reports. The system was developed using Spring Boot, Hibernate and MySQL on the backend, and React with Leaflet for the frontend.

The application allows users to visualize recycling points on an interactive map, filter them based on accepted waste categories, submit reports regarding issues such as full or damaged containers, and propose new recycling locations. A role-based access control mechanism was implemented using JWT authentication, supporting both regular users and administrators.

A gamification mechanism was integrated through a point-based reward system. Users receive points when their reports are approved by an administrator, and a leaderboard dynamically displays the ranking of active contributors.

The system architecture follows a layered design pattern, separating presentation, business logic, persistence and security concerns. The application demonstrates the integration of modern Java technologies in the development of distributed systems and highlights practical aspects such as REST API design, database modeling, authentication, and frontend-backend communication.

The main contributions of this work include the implementation of a complete report lifecycle (submission, moderation, approval), automatic creation of recycling points based on validated proposals, and the integration of an interactive geospatial visualization interface.

Future improvements may include real-time notifications, scalability optimizations, and mobile platform integration.

Cuprins

Capitolul 1

Introducere

1.1 Context și motivație

Creșterea cantității de deșeuri urbane și necesitatea reciclării eficiente impun dezvoltarea unor soluții informatice moderne care să faciliteze accesul cetățenilor la informații despre punctele de colectare.

Lucrarea prezintă implementarea unei aplicații web distribuite pentru gestionarea punctelor de reciclare, realizată folosind tehnologiile Spring Boot, Hibernate, MySQL și React.

1.2 Obiectivele proiectului

Obiectivele principale sunt:

- Implementarea unei aplicații distribuite client-server
- Integrarea unei baze de date relaționale
- Implementarea autentificării JWT
- Gestionarea rolurilor USER și ADMIN
- Implementarea unui sistem de puncte (gamification)
- Afișarea interactivă a locațiilor pe hartă

1.3 Structura lucrării

Capitolul 2 prezintă arhitectura sistemului. Capitolul 3 descrie implementarea aplicației. Capitolul 4 prezintă testarea și problemele întâmpinate. Capitolul 5 prezintă procesul de deployment. Capitolul final conține concluziile.

Capitolul 2

Arhitectura aplicației

2.1 Arhitectura generală

Aplicația dezvoltată urmează o arhitectură distribuită de tip client–server, în care componenta de prezentare (frontend) comunică prin intermediul unui API REST cu componenta de procesare (backend), iar datele sunt persistate într-o bază de date relațională.

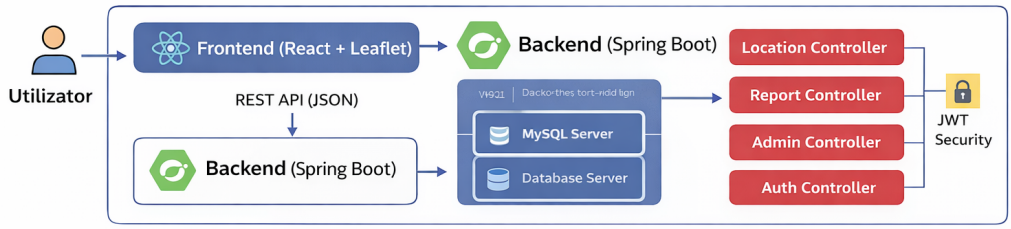

Figura 2.1: Arhitectura generală a sistemului

Sistemul este compus din trei componente principale:

- **Frontend (React + Leaflet)** – responsabil pentru interfața utilizator și interacțiunea cu utilizatorul;

- **Backend (Spring Boot)** – responsabil pentru logica de business, validarea datelor și expunerea serviciilor REST;
- **Baza de date (MySQL)** – responsabilă pentru persistența datelor.

Comunicarea între frontend și backend se realizează prin cereri HTTP de tip REST (GET, POST), iar securizarea endpoint-urilor este realizată prin autentificare bazată pe JWT (JSON Web Tokens).

Această arhitectură permite separarea clară a responsabilităților și facilitează mentenanța și extinderea aplicației.

2.2 Justificarea alegerii tehnologiilor

Tehnologiile utilizate au fost selectate pe baza următoarelor criterii:

- **Spring Boot** – oferă un framework robust pentru dezvoltarea aplicațiilor enterprise, suport nativ pentru REST și integrare cu Spring Security.
- **Hibernate (JPA)** – simplifică maparea obiect-relațională și gestionarea persistenței datelor.
- **MySQL** – sistem de gestiune a bazelor de date relaționale stabil și performant.
- **React** – permite dezvoltarea unei interfețe dinamice bazate pe componente reutilizabile.
- **Leaflet** – bibliotecă specializată pentru afișarea și manipularea hărților interactive.

Alegerea acestor tehnologii reflectă integrarea modernă a ecosistemului Java cu tehnologii frontend actuale.

2.3 Arhitectura backend

Backend-ul este organizat conform unei arhitecturi stratificate (layered architecture), respectând principiul separării responsabilităților.

Straturile principale sunt:

- **Controller Layer** – gestionează cererile HTTP și expune endpoint-uri REST.
- **Service Layer** – conține logica de business.
- **Repository Layer** – gestionează interacțiunea cu baza de date prin JPA.

- **Entity Layer** – definește modelul de date și maparea ORM.
- **Security Layer** – implementează autentificarea și autorizarea utilizatorilor.

Această organizare permite izolarea logicii aplicației și facilitează testarea unitară a fiecărui strat.

2.4 Modelul de date

Modelul de date este construit pe baza unei baze de date relaționale.

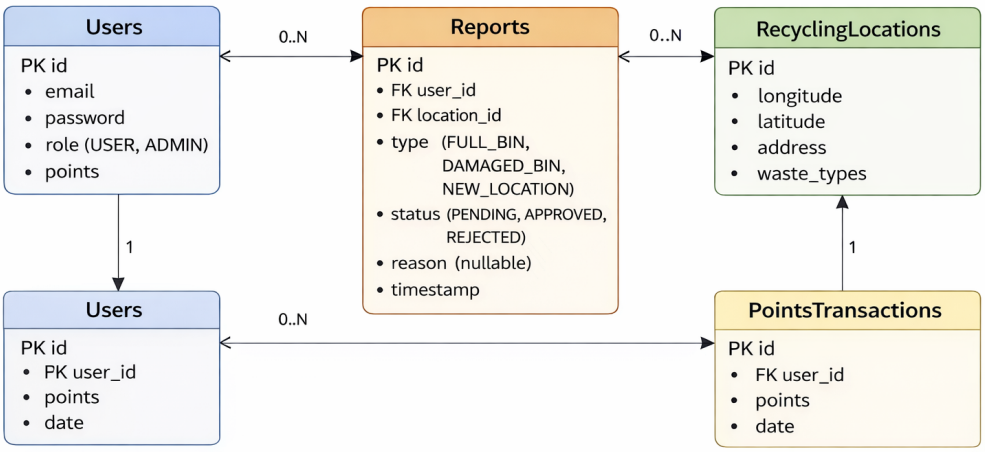

Figura 2.2: Diagrama ER a bazei de date

2.4.1 Entitatea User

Entitatea *User* reprezintă utilizatorii aplicației și conține informații despre autentificare (email, parolă), rol (USER sau ADMIN) și numărul total de puncte acumulate.

2.4.2 Entitatea RecyclingLocation

Entitatea *RecyclingLocation* reprezintă punctele de reciclare afișate pe hartă. Aceasta include coordonate geografice (latitudine, longitudine), adresă și tipurile de deșeuri acceptate.

2.4.3 Entitatea Report

Entitatea *Report* modelează raportările trimise de utilizatori. Un raport poate avea diferite tipuri (FULL_BIN, DAMAGED_BIN, NEW_LOCATION) și un status (PENDING, APPROVED, REJECTED).

2.4.4 Entitatea PointsTransaction

Entitatea *PointsTransaction* păstrează istoricul acordării punctelor utilizatorilor, asigurând trasabilitatea sistemului de gamification.

2.5 Securitatea aplicației

Securitatea este implementată utilizând Spring Security și JWT.

Fluxul de autentificare este următorul:

1. Utilizatorul trimite credențialele către endpoint-ul de login.
2. Serverul validează datele și generează un JWT.
3. Token-ul este transmis clientului.
4. Clientul atașează token-ul în header-ul Authorization pentru cererile ulterioare.

Controlul accesului este realizat pe baza rolurilor definite în sistem.

2.6 Diagrama de secvență – Flux propunere locație

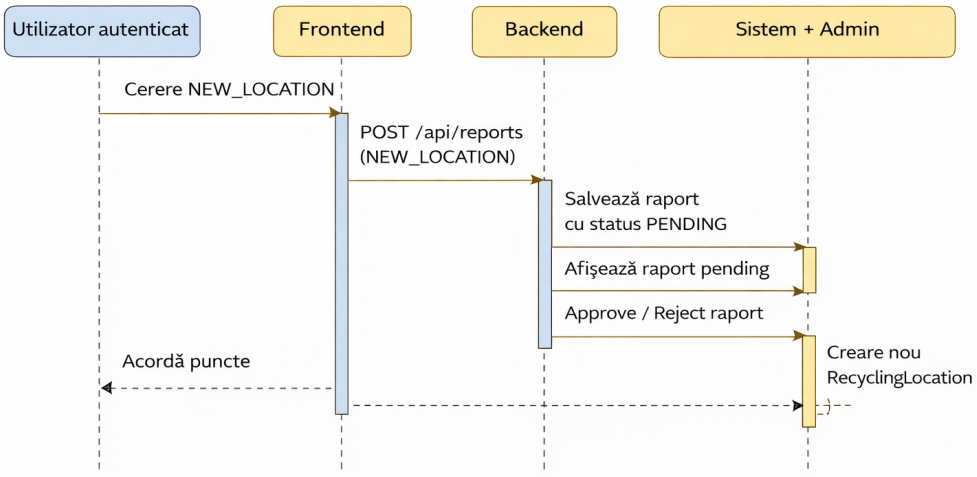

Figura 2.3: Diagramă de secvență pentru propunerea unei locații

Fluxul detaliat este:

1. Utilizatorul autenticat trimite o cerere de tip NEW_LOCATION.
2. Sistemul creează un obiect Report cu status PENDING.
3. Administratorul vizualizează raportul în panoul de administrare.
4. Administratorul aprobă raportul.
5. Sistemul creează automat o nouă entitate RecyclingLocation.
6. Utilizatorului i se acordă puncte.
7. Leaderboard-ul este actualizat.

2.7 Considerații privind scalabilitatea

Arhitectura aplicației permite scalarea independentă a componentelor:

- Backend-ul poate fi replicat pe mai multe instanțe.
- Baza de date poate fi optimizată prin indexare și replicare.
- Frontend-ul este complet decuplat de backend.

Structura modulară permite extinderea ulterioară a aplicației, de exemplu prin introducerea notificărilor în timp real sau integrarea cu servicii externe.

2.8 Interfața REST (endpoint-uri)

Backend-ul expune un set de endpoint-uri REST organizate pe resurse. În implementarea curentă, endpoint-urile sunt:

- `POST /api/auth/register` – creare cont utilizator.
- `POST /api/auth/login` – autentificare și emitere token JWT.
- `GET /api/locations` – listare / căutare locații (filtru după `wasteType` și bounding box: `minLat`, `maxLat`, `minLng`, `maxLng`).
- `GET /api/waste-types` – listarea categoriilor de deșeu.
- `POST /api/reports` – creare raport (necesită autentificare).
- `GET /api/reports/mine` – raportările utilizatorului curent.
- `GET /api/leaderboard?limit=N` – top utilizatori după puncte.

- `GET /api/admin/reports?status=PENDING|APPROVED|REJECTED` – listare rapoarte (doar ADMIN).
- `POST /api/admin/reports/{id}/approve` – aprobare raport (doar ADMIN).
- `POST /api/admin/reports/{id}/reject` – respingere raport (doar ADMIN).

2.9 Observații despre modelul de autorizare

Configurarea Spring Security permite acces public la resursele necesare încărcării inițiale a aplicației (autentificare, listarea locațiilor, tipurile de deșeuri și documentația OpenAPI). Restul endpoint-urilor necesită token JWT, iar operațiile administrative sunt protejate suplimentar prin verificarea rolului `ADMIN` (anotarea `@PreAuthorize`).

Capitolul 3

Implementarea aplicației

3.1 Considerații generale

Implementarea aplicației a fost realizată incremental, pornind de la definirea modelului de date și continuând cu dezvoltarea serviciilor REST, integrarea mecanismului de securitate și implementarea interfeței grafice.

Backend-ul este implementat în Spring Boot și respectă o arhitectură stratificată, iar frontend-ul este realizat în React utilizând componente reutilizabile.

3.2 Autentificare JWT

Autentificarea utilizatorilor este realizată prin JSON Web Tokens (JWT). Sistemul este configurat în mod stateless, ceea ce înseamnă că serverul nu păstrează sesiuni active, iar fiecare cerere autenticată conține token-ul în header-ul HTTP.

3.2.1 Fluxul de autentificare

1. Utilizatorul trimite email și parolă către endpoint-ul de login.
2. Backend-ul validează credențialele folosind *UserDetailsService*.
3. Se generează un token JWT semnat.
4. Token-ul este returnat clientului.
5. Clientul atașează token-ul în header-ul *Authorization* pentru cererile ulterioare.

3.2.2 Metoda login

```
1 public AuthResponse login(LoginRequest request) {
2     authenticationManager.authenticate(
```

```
3           new UsernamePasswordAuthenticationToken(
4               request.getEmail(),
5               request.getPassword()
6           )
7       );
8
9       User user = userRepository.findByEmail(request.getEmail
            ())
10              .orElseThrow();
11
12      String token = jwtService.generateToken(user);
13
14      return new AuthResponse(token, user.getRole(), user.
            getDisplayName());
15  }
```

Intrare: email și parolă.

Ieșire: token JWT, rol utilizator și nume afișat.

Validarea parolei este realizată folosind extitBCryptPasswordEncoder. Token-ul conține identitatea utilizatorului (extttemail) și este validat la fiecare cerere printr-un filtru dedicat (extttJwtAuthenticationFilter).

3.3 Gestionarea locațiilor

Locațiile sunt afișate pe hartă și pot fi filtrate în funcție de tipul de deșeu și zona geografică (bounding box).

3.3.1 Metoda findLocations

Metoda primește coordonatele hărții și tipul de deșeu selectat și returnează doar locațiile active care corespund criteriilor.

```
1  public List<LocationDto> findLocations(
2      Double minLat,
3      Double maxLat,
4      Double minLng,
5      Double maxLng,
6      String wasteType
7  ) {
8      return locationRepository
9          .findFiltered(minLat, maxLat, minLng, maxLng,
               wasteType)
```

```
10          .stream()
11          .map(LocationMapper::toDto)
12          .toList();
13  }
```

Filtrarea se face la nivel de repository folosind interogări personalizate.

3.4 Raportarea problemelor

Utilizatorii autentificați pot trimite raportări asociate unei locații existente sau pot propune o locație nouă.

3.4.1 Metoda createReport

```
1  public Report createReport(CreateReportRequest req, User
       user) {
2
3       Report report = new Report();
4       report.setType(req.getType());
5       report.setDescription(req.getDescription());
6       report.setStatus(ReportStatus.PENDING);
7       report.setCreatedBy(user);
8       report.setCreatedAt(Instant.now());
9
10      return reportRepository.save(report);
11  }
```

La crearea raportului:

- Statusul este setat implicit la PENDING.
- Nu se acordă puncte în această etapă.
- Validarea datelor este realizată înainte de salvare.

3.5 Aprobarea raportului

Aprobarea este realizată exclusiv de utilizatori cu rol ADMIN.

3.5.1 Metoda approveReport

```
@Transactional
public Report approveReport(Long id, Long pointsOverride,
    String comment) {

    Report report = reportRepository.findById(id)
        .orElseThrow();

    if (report.getStatus() != ReportStatus.PENDING) {
        throw new IllegalStateException("Already resolved");
    }

    if (report.getType() == ReportType.NEW_LOCATION) {
        RecyclingLocation location = new RecyclingLocation()
            ;
        location.setLatitude(report.getProposedLat());
        location.setLongitude(report.getProposedLng());
        locationRepository.save(location);
        report.setLocation(location);
    }

    report.setStatus(ReportStatus.APPROVED);
    report.setResolvedAt(Instant.now());

    awardPoints(report);

    return report;
}
```

Metoda este marcată cu *@Transactional*, pentru a asigura consistența datelor în cazul apariției unei erori.

3.5.2 Acordarea punctelor

Punctele sunt calculate prin intermediul *PointsRulesService*. La aprobarea raportului:

- Se actualizează totalul de puncte al utilizatorului.
- Se creează o entitate PointsTransaction.
- Leaderboard-ul reflectă noul clasament.

3.6 Frontend

Frontend-ul este implementat în React și utilizează biblioteca Leaflet pentru afișarea hărții.

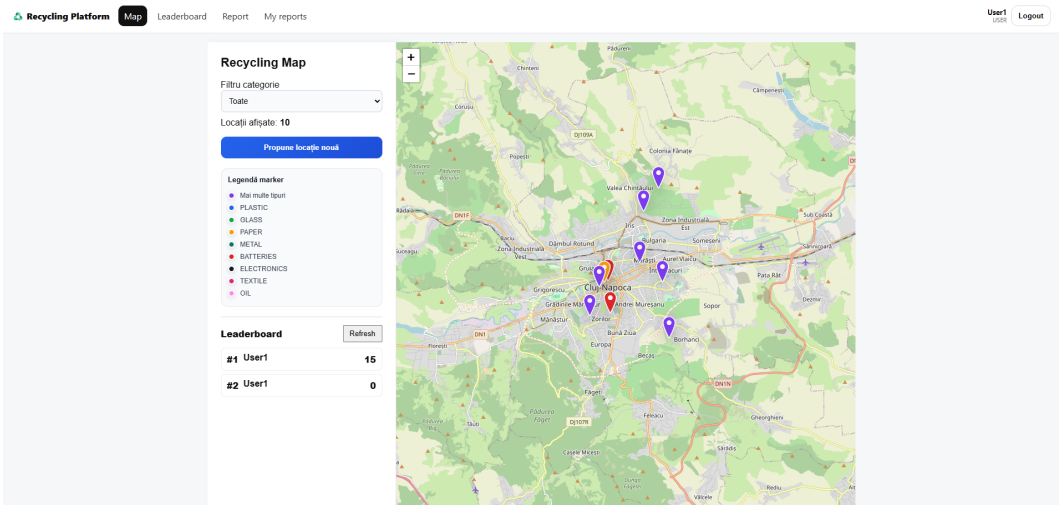

Figura 3.1: Interfața principală a aplicației

3.6.1 Structura componentelor

Aplicația este organizată pe componente reutilizabile:

- **MapPage** – componentă principală ce gestionează harta.
- **Leaderboard** – afișează clasamentul utilizatorilor.
- **AdminPanel** – permite aprobarea sau respingerea raportărilor.
- **ReportForm** – formular pentru raportarea problemelor.
- **NewLocationForm** – formular pentru propunerea unei locații noi.

3.6.2 Integrarea cu backend-ul

Comunicarea cu backend-ul se realizează printr-un modul dedicat (*api.js*), care utilizează axios pentru trimiterea cererilor HTTP.

Token-ul JWT este atașat automat în header-ul cererilor protejate.

3.7 Structura directoarelor

3.7.1 Backend

`backend/`

```
src/main/java/ro/ubb/recyclingplatform/
  controller/
  service/
  repository/
  entity/
  security/
src/main/resources/
  application.properties
```

3.7.2 Frontend

```
frontend/
  src/
    pages/
    components/
    auth/
    api.js
    main.jsx
```

3.8 Considerații privind extensibilitatea

Structura modulară permite adăugarea facilă a unor funcționalități suplimentare, precum:

- Notificări în timp real
- Sistem de badge-uri
- Export de rapoarte statistice
- Integrare cu aplicații mobile

3.9 Gestionarea stării de autentificare în frontend

Token-ul JWT și datele minime despre utilizator sunt salvate în *LocalStorage*. Modulul `api.js` configurează interceptori Axios pentru:

- atașarea automată a header-ului `Authorization: Bearer <token>`;
- curățarea sesiunii locale la răspuns `401 Unauthorized`.

3.10 Observație de mentenanță

În fișierul `api.js` există două interceptoare de tip *request* care fac același lucru (atașarea token-ului). Pentru claritate și evitarea comportamentelor greu de diagnosticat, se recomandă păstrarea unuia singur.

Capitolul 4

Testare și validare

4.1 Strategia de testare

Testarea aplicației a fost realizată pe mai multe niveluri:

- Testare unitară (Unit Testing)
- Testare de integrare (Integration Testing)
- Testare API utilizând Postman
- Testare funcțională la nivel de interfață grafică

4.2 Testare unitară

Testele unitare au fost implementate folosind framework-ul JUnit 5. Scopul acestora a fost validarea logicii de business din stratul de servicii.

4.2.1 Test pentru aprobarea unui raport

Testul verifică următoarele:

- raportul este găsit în baza de date
- statusul se schimbă în APPROVED
- utilizatorul primește puncte
- este creată o tranzacție de tip PointsTransaction

```
@Test
void shouldApproveReportAndAwardPoints() {
    Report report = new Report();
    report.setStatus(ReportStatus.PENDING);

    when(reportRepository.findById(1L))
        .thenReturn(Optional.of(report));

    adminService.approveReport(1L, null, "OK");

    assertEquals(ReportStatus.APPROVED, report.getStatus());
    verify(pointsTransactionRepository, times(1))
        .save(any(PointsTransaction.class));
}
```

4.3 Testare de integrare

Testele de integrare au verificat funcționarea completă a endpoint-urilor REST.

4.3.1 Test autentificare

```
@Test
void shouldAuthenticateUser() throws Exception {
    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content("""
        {
            "email": "user@test.com",
            "password": "password123"
        }
        """))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.token").exists());
}
```

4.4 Testare API (Postman)

API-ul a fost testat utilizând colecții Postman pentru următoarele scenarii:

- Login utilizator
- Creare raport
- Aprobare raport
- Filtrare locații

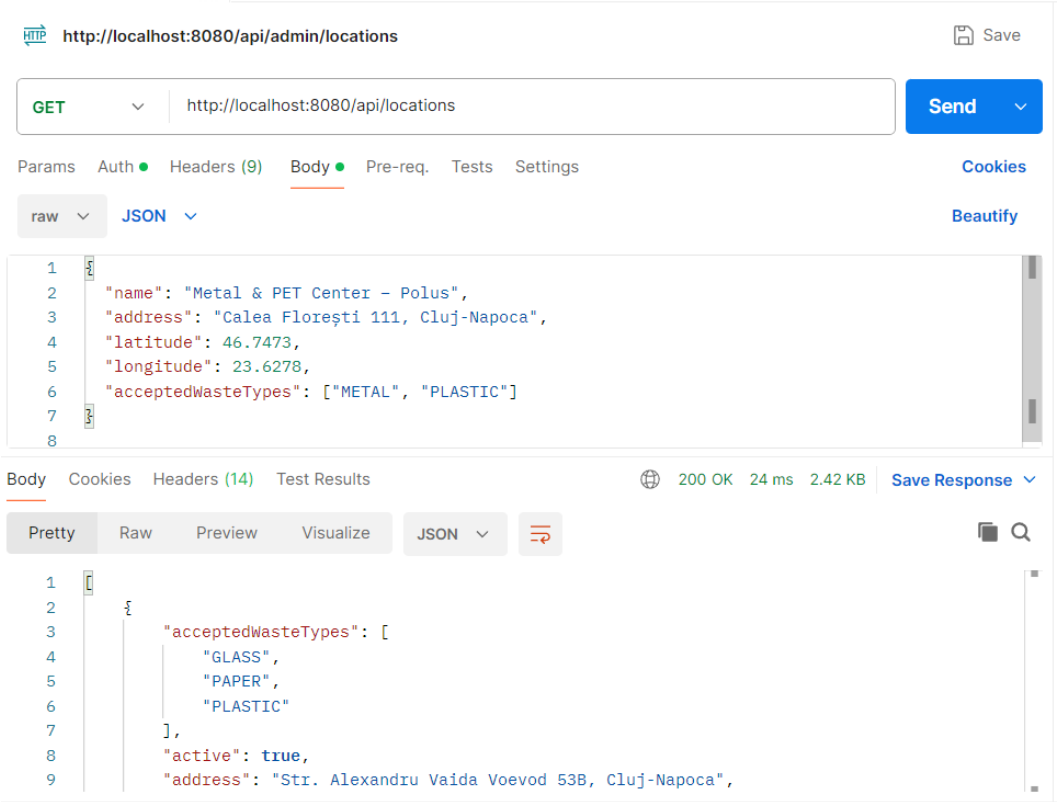

Figura 4.1: Testarea endpoint-urilor REST în Postman

4.5 Testare funcțională frontend

Testarea funcțională a componentei frontend a avut ca obiectiv verificarea corectitudinii interacțiunii utilizatorului cu interfața grafică și validarea integrării cu serviciile backend.

În primul rând, s-a verificat afișarea corectă a hărții interactive realizate cu Leaflet. Au fost testate încărcarea markerelor existente, poziționarea acestora pe baza coordonatelor geografice și afișarea informațiilor asociate fiecărei locații.

De asemenea, a fost testat fluxul de propunere a unei locații noi. Utilizatorul autenticat completează formularul corespunzător, iar aplicația transmite cererea către backend. S-a verificat afișarea mesajelor de confirmare și gestionarea corectă a eventualelor erori.

Pentru rolul de administrator, s-a validat afișarea raportărilor în panoul de administrare, inclusiv posibilitatea de aprobare sau respingere a acestora. Interfața reflectă în timp real schimbarea statusului raportului după procesare.

În plus, s-a testat actualizarea leaderboard-ului, verificând că modificarea punctajului utilizatorului este reflectată corect în clasament după aprobarea unui raport.

Testarea a fost realizată manual, urmărind scenarii reale de utilizare, pentru a valida coerența experienței utilizatorului și consistența datelor afișate.

4.6 Validare finală

Validarea finală a avut ca scop confirmarea funcționării complete a fluxului principal al aplicației, de la inițierea unei acțiuni de către utilizator până la actualizarea sistemului.

Fluxul validat a fost următorul: un utilizator autenticat transmite un raport prin intermediul aplicației. Raportul este salvat în baza de date cu status *PENDING* și devine vizibil în panoul de administrare. Administratorul analizează raportul și îl aprobă.

În cazul aprobării, sistemul execută automat operațiile asociate: acordarea punctelor utilizatorului, actualizarea clasamentului (leaderboard) și, în cazul unui raport de tip *NEW_LOCATION*, crearea unei noi entități *RecyclingLocation*, care devine vizibilă pe hartă.

Toate etapele fluxului au fost testate succesiv, iar aplicația a trecut scenariile definite fără erori funcționale. Rezultatele obținute confirmă coerența implementării și integrarea corectă între frontend, backend și baza de date.

4.7 Setul de teste implementate

În cadrul proiectului au fost implementate teste unitare pentru servicii și teste de integrare pentru autentificare. Exemple de clase de test:

- `AuthServiceTest`, `AuthControllerTest`
- `LocationServiceTest`
- `ReportServiceTest`, `AdminServiceTest`
- `LeaderboardServiceTest`
- `PointsRulesServiceTest`

Capitolul 5

Rulare, configurare și livrare (Deployment)

5.1 Configurare backend

Backend-ul este o aplicație Spring Boot configurată prin fișierul `application.properties`. Parametrii esențiali sunt:

- **Conexiunea la baza de date**: URL, utilizator, parolă.
- **Politica Hibernate DDL**: `spring.jpa.hibernate.ddl-auto=update` (util în dezvoltare; pentru producție se recomandă migrații controlate).
- **JWT**: cheia secretă și durata de valabilitate a token-ului.
- **Portul aplicației**: `server.port`.

Observație de securitate: valoarea `app.jwt.secret` trebuie înlocuită cu un secret lung (minim 32 caractere) și nu trebuie comisă în repository.

5.2 Baza de date și Docker

Pentru a simplifica rularea locală, proiectul conține un fișier `docker-compose.yml` care pornește o instanță MySQL 8.0.

```
1 # din directorul backend/docker
2 docker compose up -d
```

După pornirea containerului, backend-ul poate fi rulat local (de exemplu din IDE). Baza de date este expusă pe portul `3306`, iar datele sunt persistate într-un volum Docker.

5.3 Pornirea aplicației

5.3.1 Backend

```
# exemplu generic
./mvnw spring-boot:run
```

(Completează aici comanda reală folosită în proiectul tău: Maven/Gradle, profiluri, etc.)

5.3.2 Frontend

Frontend-ul este o aplicație React (Vite). API-ul este configurat prin variabila de mediu `VITE_API_URL` (fallback la `http://localhost:8080`).

```
npm install
npm run dev
```

5.4 Documentarea API-ului (OpenAPI/Swagger)

Aplicația expune documentație OpenAPI. Endpoint-urile publice includ:

- `/v3/api-docs`
- `/swagger-ui/index.html`

5.5 Recomandări pentru producție

Pentru o livrare în producție sunt recomandate următoarele îmbunătățiri:

- Migrații de schemă cu Flyway/Liquibase în loc de `ddl-auto=update`.
- Secret management (Vault, variabile de mediu, Docker secrets).
- Configurare CORS strictă și rate limiting.
- Observabilitate: logging structurat, metrics (Micrometer), tracing.
- Pipeline CI/CD (build, test, lint, deploy).

Capitolul 6

Concluzii

Proiectul *Recycling Platform* a avut ca obiectiv dezvoltarea unei aplicații web complete pentru raportarea și gestionarea locațiilor de reciclare, utilizând tehnologii moderne din ecosistemul Java pentru backend și tehnologii web moderne pentru frontend.

În urma implementării, aplicația demonstrează o arhitectură stratificată clar definită, separând responsabilitățile între nivelul de prezentare, nivelul de business și nivelul de persistență. Utilizarea Spring Boot a permis organizarea coerentă a componentelor aplicației, iar integrarea mecanismului de autentificare bazat pe JWT asigură un control securizat al accesului la resurse.

Gestionarea rolurilor (utilizator și administrator) contribuie la delimitarea clară a responsabilităților în cadrul sistemului. De asemenea, sistemul de puncte implementat introduce un mecanism de gamification care stimulează implicarea utilizatorilor în procesul de raportare a locațiilor de reciclare.

Un element important al aplicației îl reprezintă fluxul complet de procesare a unui raport: utilizatorul transmite o propunere, administratorul o analizează și, în cazul aprobării, locația este creată și integrată automat în sistem. Acest flux reflectă o implementare coerentă a regulilor de business și a controlului administrativ.

Integrarea hărții interactive prin biblioteca Leaflet oferă o experiență vizuală intuitivă și facilitează identificarea rapidă a punctelor de reciclare. Comunicarea dintre frontend și backend se realizează prin API REST documentat cu OpenAPI/Swagger, ceea ce asigură claritate și extensibilitate.

Din punct de vedere tehnic, proiectul evidențiază:

- utilizarea unei arhitecturi scalabile și ușor de extins;
- implementarea autentificării stateless prin token JWT;
- separarea logicii de business de logica de prezentare;
- integrarea unei baze de date relaționale pentru persistența datelor;

- testarea funcționalităților principale prin teste automate.

În ceea ce privește direcțiile viitoare de dezvoltare, aplicația poate fi extinsă prin introducerea notificărilor în timp real, dezvoltarea unei aplicații mobile dedicate, precum și prin migrarea către o arhitectură bazată pe microservicii pentru a îmbunătăți scalabilitatea și modularitatea sistemului.

În concluzie, proiectul demonstrează aplicarea coerentă a conceptelor studiate în cadrul programului de formare și oferă o bază solidă pentru dezvoltări ulterioare într-un context real de producție.

Bibliografie

[hib] Hibernate orm documentation. Online. Accessed: 2026-02-17.

[jun] Junit 5 user guide. Online. Accessed: 2026-02-17.

[jwt] Json web token (jwt). RFC 7519. Internet Engineering Task Force.

[lea] Leaflet documentation. Online. Accessed: 2026-02-17.

[mys] Mysql 8.0 reference manual. Online. Accessed: 2026-02-17.

[ope] Openapi specification. Online. Accessed: 2026-02-17.

[rea] React documentation. Online. Accessed: 2026-02-17.

[spra] Spring boot reference documentation. Online. Accessed: 2026-02-17.

[sprb] Spring security reference. Online. Accessed: 2026-02-17.