Universitatea
Transilvania
din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

# Universitatea Transilvania din Brașov

## Facultatea de Inginerie Electrică și Știința Calculatoarelor

# Proiect de software pentru telecomunicații

Student: Cătălin Daniel Bradac, Andreea Negru, Simona Grama

Specializarea: TST

Anul III

# Cuprins

## 1. Introducere

Scopul aplicației **Admin Zone** este administrarea unui sistem educațional simplificat: gestionarea studenților, cursurilor, înscrierilor, prezențelor, utilizatorilor și auditului operațiilor. Soluția este implementată în Java (Spring Boot) pentru backend și React (Vite + TypeScript) pentru frontend, folosind MySQL ca bază de date relațională.

## 2. Alegerea și descrierea modului de dezvoltare software

Pentru dezvoltare este potrivit un model iterativ Agile (Scrum/Kanban), deoarece funcționalitățile pot fi livrate incremental (MVP → module suplimentare), iar feedback-ul utilizatorului poate fi integrat rapid.

## 3. Roluri și responsabilități în cadrul echipei

❖ Product Owner (PO) – Management cerințe & validare livrabile

Responsabilități principale:

1. Definirea obiectivelor aplicației și a cerințelor funcționale/nefuncționale.
2. Prioritizarea backlog-ului și stabilirea criteriilor de acceptanță.
3. Validarea livrabilelor (funcționalități, UX, documentație) și feedback iterativ.
4. Alinierea echipei pe scope, termene și versiuni (release management).

Persoană: Bradac Cătălin Daniel

❖ Team Lead / Backend Developer – Arhitectură backend, API & securitate

Responsabilități principale:

1. Proiectarea arhitecturii backend și a contractului API (REST endpoints, DTO-uri, validări).
2. Implementarea logicii de business (services), persistență (JPA/Hibernate) și integrare MySQL.
3. Configurarea autentificării și autorizării (Spring Security, roluri/permisiuni).
4. Tratarea erorilor prin excepții custom + handler global și asigurarea logging-ului relevant.

Persoane:

Bradac Cătălin Daniel (coordonare tehnică, integrare backend, decizii arhitecturale)

Negru Andreea (implementare backend pe module, ajustări API/servicii, contribuții la securitate)

❖ Frontend Developer – Interfață, fluxuri UI & integrare cu API

Responsabilități principale:

1. Implementarea interfeței în **React + TypeScript** (pagini, componente reutilizabile, layout).
2. Definirea rutelor și a fluxurilor aplicației (navigare, protecție rute, stări).
3. Integrarea cu backend-ul (Axios / client API), tratarea erorilor și feedback UI.
4. Validări în formulare și îmbunătățiri UX (loading states, mesaje de eroare/succes).

Persoană: Negru Andreea

❖ QA / DevOps – Testare, livrare, rulare în Docker/EC2 & monitorizare

Responsabilități principale:

1. Definirea scenariilor de test (funcționale + negative) și verificări end-to-end.
2. Verificarea consistenței API ↔ UI (status codes, payload-uri, edge cases).
3. Livrare și rulare în **Docker Compose** (servicii: API, MySQL, reverse proxy), configurări runtime.
4. Suport operare/mentenanță: loguri containere, monitorizare basic, debug rapid în producție.

Persoane:

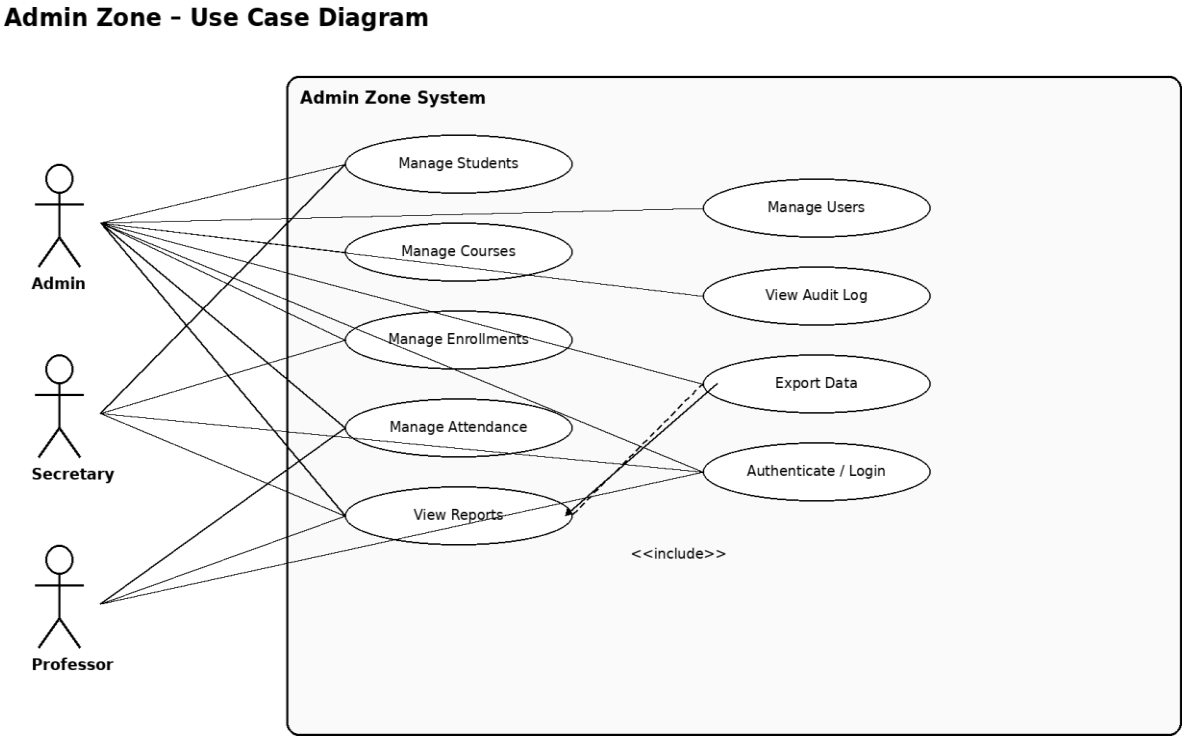
Negru Andreea (testare UI/E2E, verificări integrare)

Bradac Cătălin Daniel (deploy, networking, configurare infrastructură, troubleshooting)

Grama Simona (scenarii de test, validări funcționale, verificări regresie)

# 4. Modelarea cerințelor – Use Case Diagram

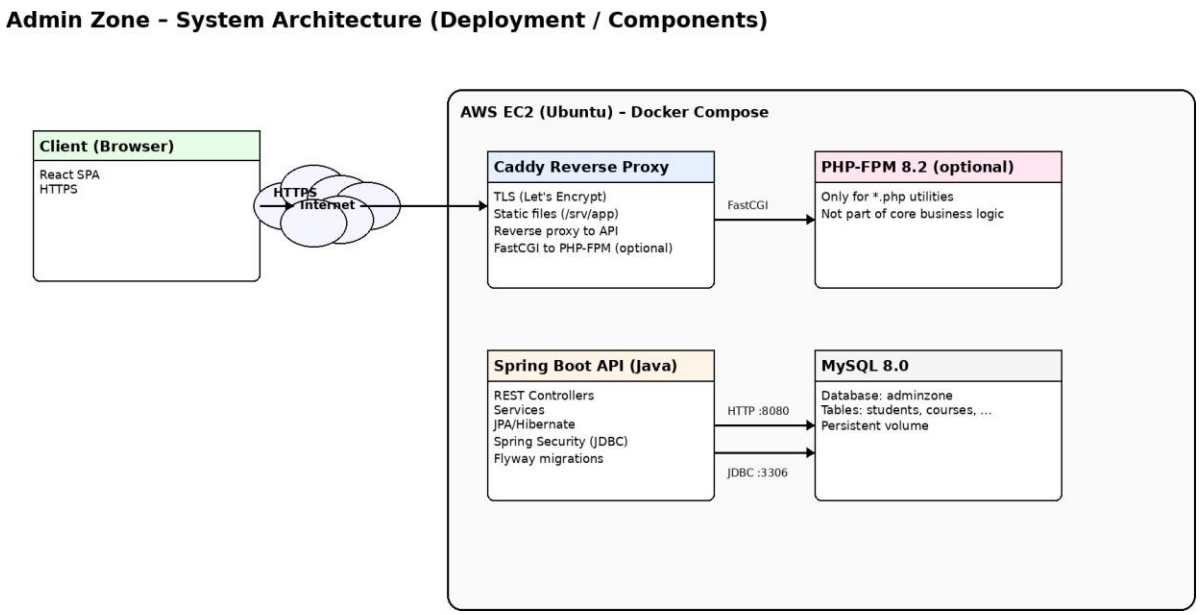
Diagrama use case descrie actorii și funcționalitățile principale. Actorii tipici sunt: Admin, Secretar și Profesor. Adminul gestionează utilizatori/roluri și audit; Secretarul gestionează studenți/cursuri/înscrieri; Profesorul gestionează prezențe.




*Figură 1 - Use Case Diagram*

# 5. Modelarea arhitecturii sistemului

Sistemul are o arhitectură pe straturi (layered) și separă clar prezentarea (frontend) de logica de business (backend). Comunicarea se face prin API REST peste HTTPS.



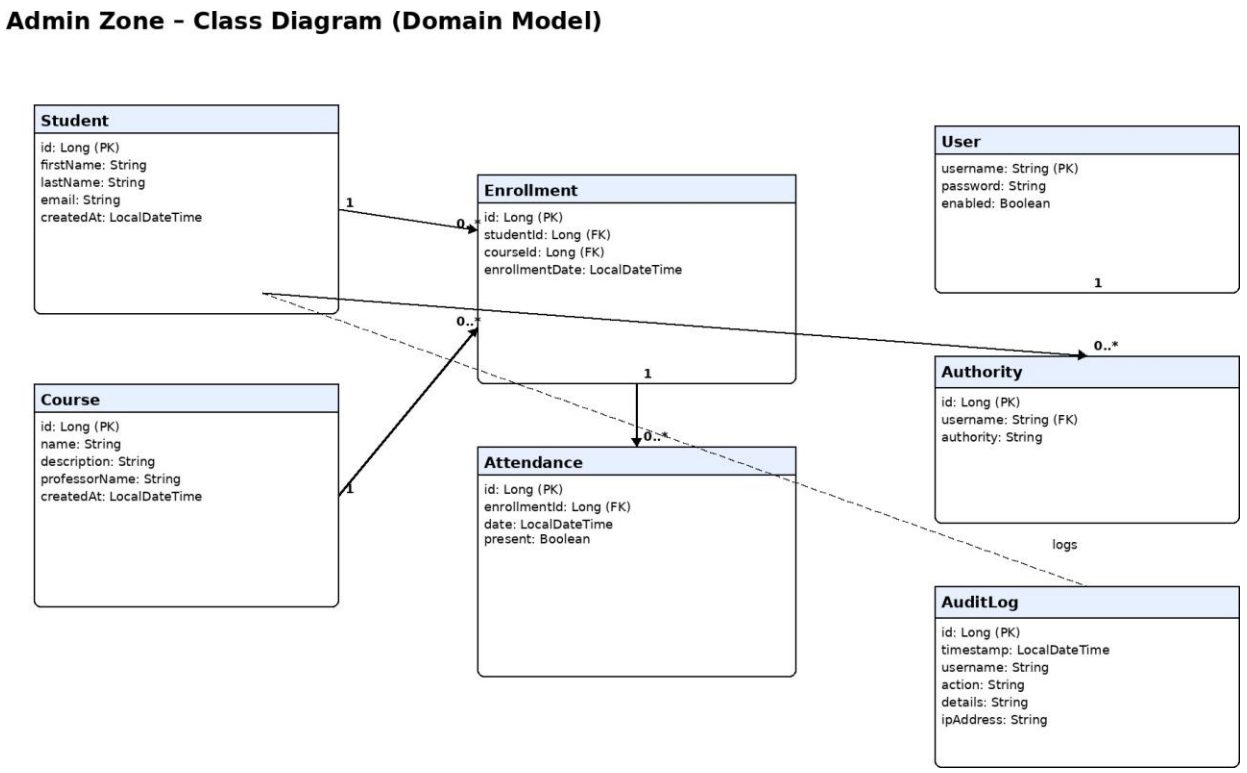

*Figură 2 - Diagrama arhitecturii (frontend-API-DB)*

## 5.1. Straturi și responsabilități

- Frontend (React + TypeScript): UI, rutare, formulare, validări, consum API.
- Backend (Spring Boot): controlere REST, servicii, reguli de business, audit, securitate.
- Persistență (Spring Data JPA): repositories, mapare entități, tranzacții.
- Baza de date (MySQL): stocare relațională a entităților și a relațiilor.
- Securitate (Spring Security): autentificare și autorizare pe roluri.

# 6. Diagrame UML – Class Diagram

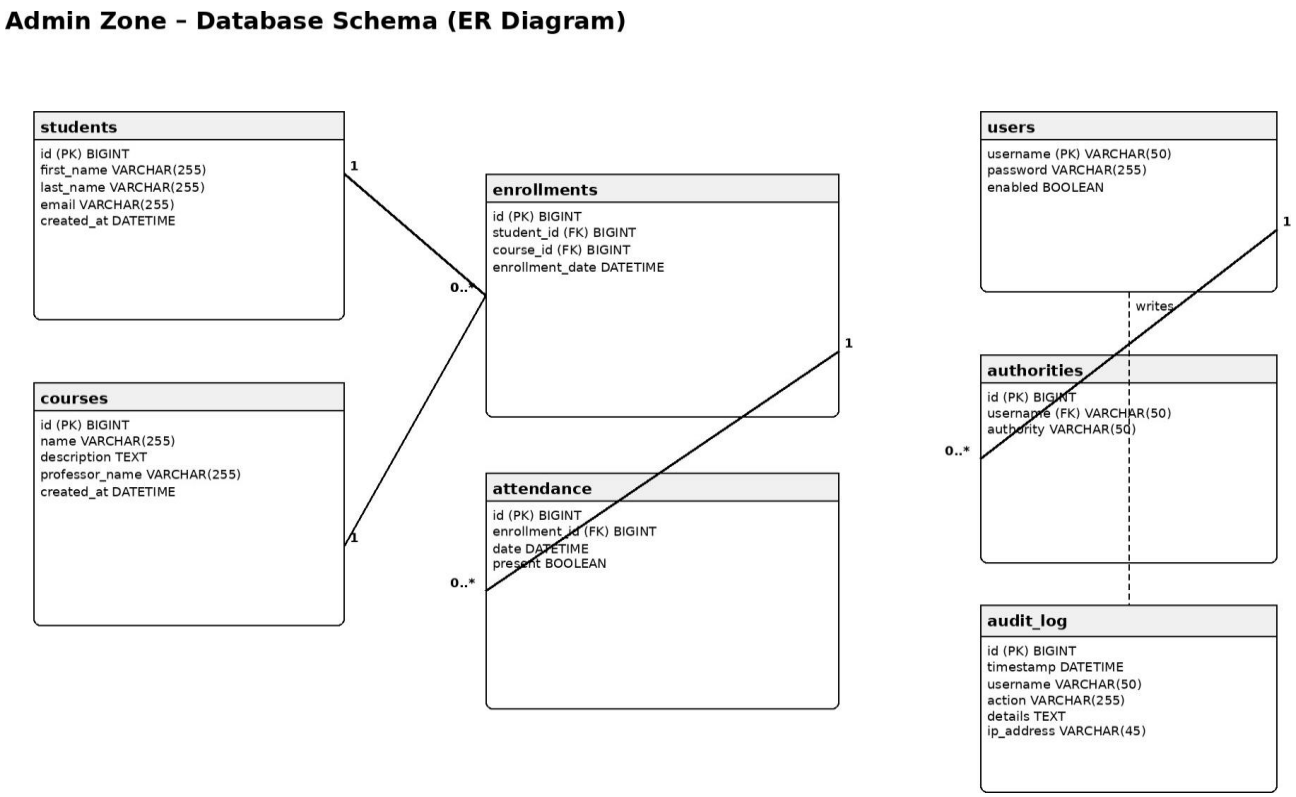
Diagrama de clasă surprinde modelul domeniului (entități și relații). În implementare, aceste relații sunt mapate în JPA (OneToMany, ManyToOne etc.).




*Figură 3 - Class Diagram (UML)*

# 7. Structura bazei de date

Baza de date este relațională (MySQL). Structura include tabele pentru entitățile principale și pentru securitate. Pentru Spring Security sunt prezente tabele/entități dedicate: users și authorities.




*Figură 4 - ER Diagram (tabele și relații)*

## 7.1. Migrații și inițializare

Schema este gestionată prin migrații (Flyway) aflate în **`src/main/resources/db/migration`** Acest mecanism asigură versionarea și reproducerea bazei de date între medii (local, docker, producție).

# 8. Structura proiectului și descrierea codului

Proiectul este organizat într-un monorepo: backend (Spring Boot) în rădăcină și frontend (React) în directorul **`frontend/`**. Mai jos este o vedere sintetică a structurii de fișiere relevante.

### 8.1. Structura la nivel de repository

```
admin-zone-api_prod_v1.0.4/
|-- docs/
|     |-- class-diagram.puml
|     |-- er-diagram.puml
|     |-- postman-collection.json
|     |-- use-case-diagram.puml
|-- frontend/
|     |-- public/
|     |     |-- vite.svg
|     |-- src/
|     |     |-- api/
|     |     |-- auth/
|     |     |-- components/
|     |     |-- pages/
|     |     |-- types/
|     |     |-- App.tsx
|     |     |-- index.css
|     |     |-- main.tsx
|     |     |-- theme.ts
|     |-- .env.example
|     |-- index.html
|     |-- package.json
|     |-- package-lock.json
|     |-- tsconfig.json
|     |-- tsconfig.node.json
|     |-- vite.config.ts
|-- src/
|     |-- main/
|     |     |-- java/
|     |     |     |-- com/adminzone/
|     |     |     |-- AdminZoneApiApplication.java
|     |     |     |-- config/
|     |     |     |-- controller/
|     |     |     |-- dto/
|     |     |     |-- entity/
|     |     |     |-- exception/
|     |     |     |-- repository/
|     |     |     |-- security/
|     |     |     |-- service/
|     |     |     |-- util/
|     |     |-- resources/
|     |     |-- application.yml
|     |     |-- application-docker.yml
|     |     |-- db/migration/ (Flyway SQL scripts)
|     |-- test/
|-- Dockerfile
|-- docker-compose.yml
|-- pom.xml
|-- README.md
```

## 8.2. Frontend – organizare și flux

Frontend-ul este o aplicație React (Vite + TypeScript). Organizarea tipică:

- **`frontend/src/pages/`** – pagini (Dashboard, Studenți, Cursuri, Înscrieri, Prezențe, Utilizatori, Audit Log).
- **`frontend/src/components/`** – componente reutilizabile (layout, tabele, formulare, dialoguri).
- **`frontend/src/api/`** – client HTTP (Axios) și funcții de acces la backend.
- **`frontend/src/auth/`** – logică de autentificare (token/session, protecție rute).

Structură (fragment):

```
frontend/
├── public/
│   └── vite.svg
├── src/
│   ├── api/
│   │   └── client.ts
│   ├── auth/
│   │   ├── AuthContext.tsx
│   │   ├── index.ts
│   │   └── RequireAuth.tsx
│   ├── components/
│   │   ├── index.ts
│   │   └── Layout.tsx
│   ├── pages/
│   │   ├── AttendancePage.tsx
│   │   ├── AuditPage.tsx
│   │   ├── CoursesPage.tsx
│   │   ├── DashboardPage.tsx
│   │   ├── EnrollmentsPage.tsx
│   │   ├── index.ts
│   │   ├── LoginPage.tsx
│   │   ├── StudentsPage.tsx
│   │   └── UsersPage.tsx
│   ├── types/
│   │   └── index.ts
│   ├── App.tsx
│   ├── index.css
│   ├── main.tsx
│   └── theme.ts
├── .env.example
├── index.html
├── package-lock.json
├── package.json
├── tsconfig.json
├── tsconfig.node.json
└── vite.config.ts
```

Comunicația cu backend-ul se face printr-un client Axios configurat cu **`baseURL: '/api'`**, astfel încât, în producție, reverse proxy-ul (Caddy) poate ruta atât UI cât și API pe același domeniu.

### 8.3. Backend – organizare și flux

Backend-ul este o aplicație Spring Boot (Maven) cu pachete separate pe responsabilități:

- **`controller/`** – expune endpoint-uri REST.
- **`service/`** – implementă logica de business și orchestrarea operațiilor.
- **`repository/`** – acces la baza de date (Spring Data JPA).
- **`entity/`** – modelul JPA (inclusiv `User` și `Authority` pentru securitate).
- **`security/`** – configurare Spring Security și mecanisme de autentificare/autorizare.
- **`config/`** și **`util/`** – configurări și utilitare.

Structură (fragment):

```
main/
├── java/
│   └── com/
│       └── adminzone/
│       ├── config/
│       │   ├── DataInitializer.java
│       │   ├── OpenApiConfig.java
│       │   ├── SecurityConfig.java
│       │   └── WebConfig.java
│       ├── controller/
│       │   ├── AttendanceController.java
│       │   ├── AuditController.java
│       │   ├── AuthController.java
│       │   ├── CourseController.java
│       │   ├── EnrollmentController.java
│       │   ├── ExportController.java
│       │   ├── StudentController.java
│       │   └── UserController.java
│       ├── dto/
│       │   ├── AttendanceRequest.java
│       │   ├── AttendanceResponse.java
│       │   ├── AttendanceStatsResponse.java
│       │   ├── AuditLogResponse.java
│       │   ├── CourseRequest.java
│       │   ├── CourseResponse.java
│       │   ├── EnrollmentRequest.java
│       │   ├── EnrollmentResponse.java
│       │   ├── LoginRequest.java
│       │   ├── LoginResponse.java
│       │   ├── PageResponse.java
│       │   ├── StudentRequest.java
│       │   ├── StudentResponse.java
│       │   ├── UserRequest.java
```

```
│       │   └── UserResponse.java
│       ├── entity/
│       │   ├── Attendance.java
│       │   ├── AuditLog.java
│       │   ├── Authority.java
│       │   ├── Course.java
│       │   ├── Enrollment.java
│       │   ├── EnrollmentId.java
│       │   ├── Student.java
│       │   └── User.java
│       ├── exception/
│       │   ├── BadRequestException.java
│       │   ├── BusinessRuleException.java
│       │   ├── DuplicateResourceException.java
│       │   ├── ErrorResponse.java
│       │   ├── GlobalExceptionHandler.java
│       │   └── ResourceNotFoundException.java
│       ├── repository/
│       │   ├── AttendanceRepository.java
│       │   ├── AuditLogRepository.java
```

```
│       │       ├── CourseRepository.java
│       │       ├── EnrollmentRepository.java
│       │       ├── StudentRepository.java
│       │       └── UserRepository.java
│       ├── security/
│       │   ├── CustomAccessDeniedHandler.java
│       │   ├── CustomAuthenticationFailureHandler.java
│       │   ├── CustomAuthenticationSuccessHandler.java
│       │   └── CustomLogoutSuccessHandler.java
│       ├── service/
│       │   ├── AttendanceService.java
│       │   ├── AuditService.java
│       │   ├── CourseService.java
│       │   ├── EnrollmentService.java
│       │   ├── ExportService.java
│       │   ├── StudentService.java
│       │   └── UserService.java
│       └── AdminZoneApplication.java
└── resources/
    ├── db/
    │   └── migration/
    │       ├── V1__create_security_tables.sql
    │       ├── V2__create_domain_tables.sql
    │       └── V3__seed_data.sql
    ├── application-docker.yml
    └── application.yml
```

Controlere identificate în cod:

`AttendanceController.java`, `AuditController.java`, `AuthController.java`, `CourseController.java`, `EnrollmentController.java`, `ExportController.java`, `StudentController.java`, `UserController.java`

Entități principale identificate în cod:
`Attendance.java, AuditLog.java, Authority.java, Course.java, Enrollment.java, EnrollmentId.java, Student.java, User.java`

### 8.4. Cum funcționează o cerere tipică

Exemplu: listarea studenților. Clientul (UI) solicită **`GET /api/students`**. Cererea ajunge în controller, care deleagă către un service. Service-ul apelează repository-ul JPA, care execută interogarea în MySQL și întoarce rezultatul mapat în DTO-uri. Pe parcurs, filtrele Spring Security aplică regulile de acces (**`ROLE_ADMIN/ROLE_SECRETARY/ROLE_PROFESSOR`**), iar operația poate fi înregistrată în Audit Log.

## 9. Autentificarea și autorizarea (Spring Security)

Sistemul folosește Spring Security pentru controlul accesului. Modelul este bazat pe utilizatori și roluri (authorities). În baza de date există entitățile/tabelele **`users`** și **`authorities`**, iar endpoint-urile sunt protejate pe roluri.

- Autentificare: validarea credențialelor utilizatorului.
- Autorizare: restricționarea operațiilor în funcție de rol (de ex. Admin poate gestiona utilizatori).
- Audit: înregistrarea operațiilor importante (cine, ce, când).

## 10. Concluzii

Raportul a prezentat metodologia de dezvoltare, rolurile, structura bazei de date, modelarea arhitecturii, diagramele UML (use case și class), precum și o descriere practică a structurii proiectului și a fluxului de execuție. Soluția finală este scalabilă pentru extindere (statistici, rapoarte, exporturi) și poate fi livrată containerizat (Docker).

Notă privind PHP

Implementarea aplicației (logică de business, API, securitate, persistență) este realizată în Java. Dacă în mediul de deployment există fișiere/containere PHP pentru testare sau compatibilitate cu anumite instrumente de backup/hosting, acestea nu fac parte din funcționalitatea cerută de proiect și pot fi eliminate.

Release v1.0.4: https://github.com/dbradacc/PST/releases/tag/v1.0.4
Continut: https://github.com/dbradacc/PST/tree/v1.0.4
Live demo: https://proiect.bradac.ro/