Raport Tehnic

Sistem Informatic de Gestiune a Secției de Poliție

Ivan Vlad-Daniel

Ianuarie 2026

Grupa: 331AB

Documentație completă incluzând proiectarea bazei de date, arhitectura backend și interfața utilizator.

Cuprins

1 Descrierea Cerințelor și Funcționalităților

Aplicația **"Police DB"** este un sistem informatic complex destinat digitalizării activității operative a unei secții de poliție. Proiectul răspunde nevoii de a gestiona eficient resursele umane, evidența populației și istoricul infracțional, înlocuind registrele fizice cu o soluție digitală securizată.

1.1 Cerințe Funcționale

Sistemul a fost proiectat pentru a îndeplini următoarele scenarii de utilizare:

- **Gestiunea Resurselor Umane (Polițiști):** Administrarea personalului, inclusiv gradul, funcția și datele de autentificare.
- **Evidența Populației:** Stocarea datelor cetățenilor cu validarea strictă a unicității (CNP).
- **Managementul Operativ (Incidente):** Înregistrarea faptelor cu detalii despre locație, timp și participanți, având posibilitatea de a schimba statusul dosarului (Activ $\rightarrow$ Închis).
- **Sistemul Fiscal (Amenzi):** Emiterea sancțiunilor cu calcul automat și validare a sumelor.
- **Raportare Analitică:** Generarea de statistici complexe (zone de risc, eficiența agenților) bazate pe date istorice.

1.2 Funcționalități Avansate (Elemente de Distincție)

Pentru a asigura robustețea și utilitatea practică, au fost implementate mecanisme superioare cerințelor standard:

1. **Smart Delete (Integritate Referențială Activă):** Sistem care blochează ștergerea accidentală a datelor. Utilizatorul primește feedback vizual (Roșu/Portocaliu/Verde) în funcție de dependențele entității (ex: nu poți șterge un polițist care are un dosar activ în lucru).
2. **Live Search:** Căutare instantanee insensibilă la diacritice și majuscule (`COLLATE Latin1_General_100_CI_AI`), care interoghează simultan multiple coloane.
3. **Validări Matematice:** Verificarea cifrei de control a CNP-ului și validarea sumelor amenzilor (multiplu de 25 RON).

2 Etapa de Proiectare a Bazei de Date

Proiectarea bazei de date a urmat modelul relațional, asigurând normalizarea datelor până la Forma Normală 3 (3NF) pentru a elimina redundanța și anomaliile de actualizare.

2.1 Diagrama Entitate-Relație (ERD)

Structura bazei de date este centrată pe entitățile operaționale (Incident, Amenda) care leagă entitățile nomenclator (Politist, Persoana, Adresa).

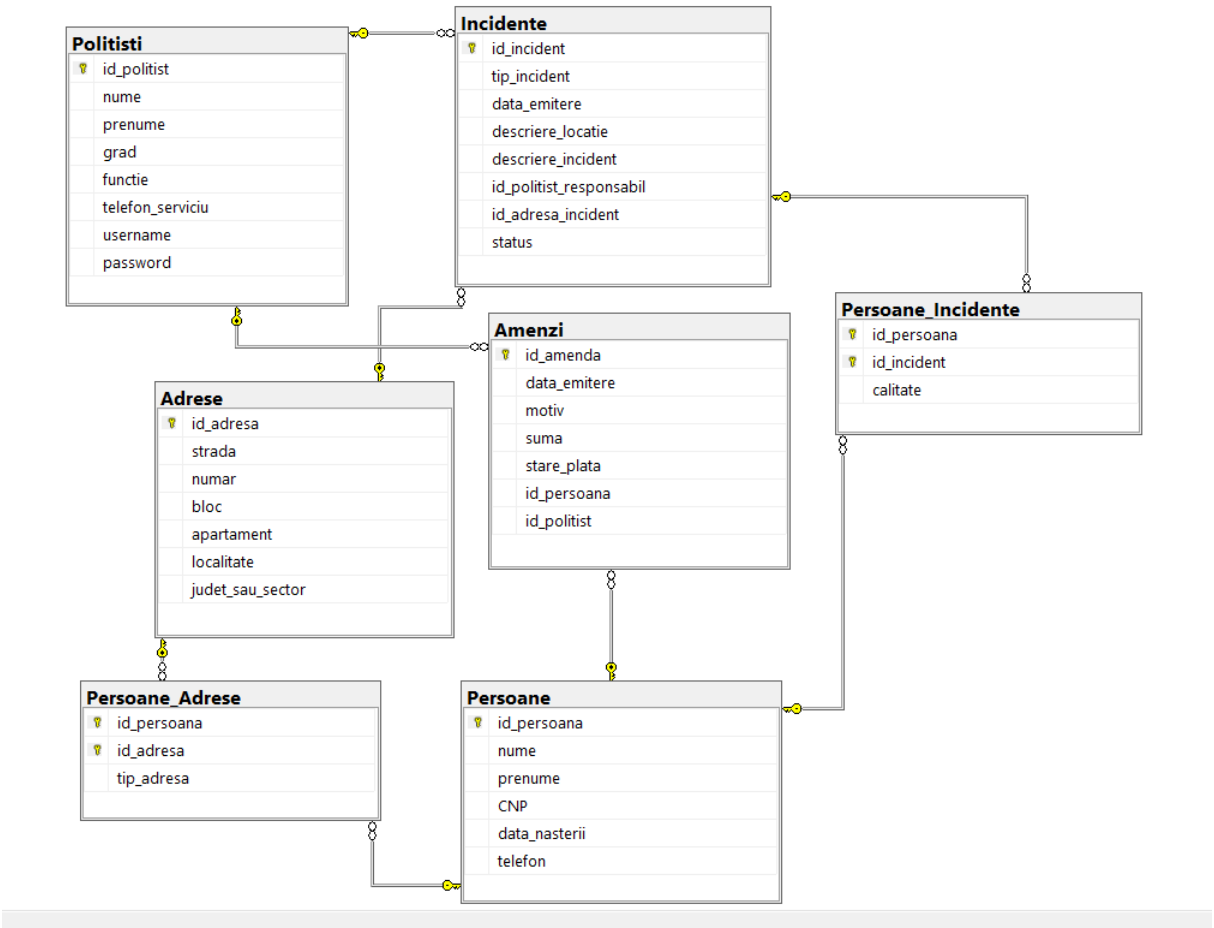


Figura 1: Diagrama Entitate-Relație a bazei de date Police DB, evidențiind cheile primare, cheile străine și cardinalitățile.

2.2 Descrierea Tabelelor și a Relațiilor

2.2.1 Tabele Principale

- **Politisti:** Tabela personalului operativ.
  - *PK:* `id_politist` (int, NOT NULL, Identity).
  - *Coloane:*
    - `nume`: nvarchar(50), NOT NULL.
    - `prenume`: nvarchar(50), NOT NULL.

        * `grad`: nvarchar(50), ALLOW NULL.
        * `functie`: nvarchar(100), ALLOW NULL.
        * `telefon_serviciu`: nvarchar(15), ALLOW NULL.
        * `username`: varchar(50), ALLOW NULL (Constrângere: UNIQUE).
        * `password`: varchar(100), ALLOW NULL

- **Persoane:** Tabela de evidență a cetățenilor.
    - *PK:* `id_persoana` (int, NOT NULL, Identity).
    - *Coloane:*
        * `nume`: nvarchar(50), NOT NULL.
        * `prenume`: nvarchar(50), NOT NULL.
        * `CNP`: nvarchar(13), NOT NULL (Constrângere: UNIQUE).
        * `data_nasterii`: date, ALLOW NULL.
        * `telefon`: nvarchar(15), ALLOW NULL.

- **Adrese:** Registur adrese.
    - *PK:* `id_adresa` (int, NOT NULL, Identity).
    - *Coloane:*
        * `strada`: nvarchar(100), NOT NULL.
        * `numar`: nvarchar(10), NOT NULL.
        * `bloc`: nvarchar(10), ALLOW NULL.
        * `apartament`: int, ALLOW NULL.
        * `localitate`: nvarchar(50), NOT NULL.
        * `judet_sau_sector`: nvarchar(50), NOT NULL.

- **Incidente:** Tabela operațională centrală.
    - *PK:* `id_incident` (int, NOT NULL, Identity).
    - *Coloane:*
        * `tip_incident`: nvarchar(100), NOT NULL.
        * `data_emitere`: datetime2(7), NOT NULL.
        * `descriere_locatie`: nvarchar(255), ALLOW NULL (Folosită dacă adresa nu e standardizată).
        * `descriere_incident`: nvarchar(MAX), NOT NULL (Stochează text lung).
        * `status`: nvarchar(50), ALLOW NULL.
        * `id_politist_responsabil`: int, ALLOW NULL (Foreign Key → Politisti).
        * `id_adresa_incident`: int, ALLOW NULL (Foreign Key → Adrese).

- **Amenzi:** Tabela sancțiunilor contravenționale.
    - *PK:* `id_amenda` (int, NOT NULL, Identity).
    - *Coloane:*

* `data_emitere`: datetime2(7), NOT NULL.
* `motiv`: nvarchar(255), NOT NULL.
* `suma`: decimal(10, 2), NOT NULL (Precizie financiară).
* `stare_plata`: nvarchar(20), NOT NULL.
* `id_persoana`: int, ALLOW NULL (Foreign Key → Persoane).
* `id_politist`: int, ALLOW NULL (Foreign Key → Politisti).

2.2.2 Tabele de Asociere (Junction Tables)

Aceste tabele descompun relațiile Many-to-Many (N:M) în relații 1:N, asigurând integritatea referențială prin chei primare compuse (Composite Keys).

- **Persoane_Incidente:**
  - *PK Compusă:* `id_persoana` (FK) + `id_incident` (FK).
  - *Coloane:*
    * `calitate`: nvarchar(20), NOT NULL (Specifică rolul: Martor, Suspect, Victimă).
- **Persoane_Adrese:**
  - *PK Compusă:* `id_persoana` (FK) + `id_adresa` (FK).
  - *Coloane:*
    * `tip_adresa`: nvarchar(20), NOT NULL (Specifică rolul: Domiciliu, Reședință).

2.3 Constrângeri de Integritate Impuse

1. **Chei Primare (PK):** Toate tabelele folosesc coloane `IDENTITY(1,1)` pentru generarea automată a ID-urilor unice.
2. **Chei Străine (FK):** Relațiile dintre tabele asigură integritatea referențială. Nu se poate introduce un Incident fără a specifica un Polițist existent.
3. **Validare la nivel de Aplicație:** Deși baza de date permite ștergerea (ON DELETE CASCADE în anumite tabele de legătură), aplicația impune o constrângere logică superioară ("Smart Delete") care previne ștergerea înregistrărilor părinte dacă acestea au copii activi (stare "Activ" sau "Neplătită").

3 Arhitectura Backend și Logica Aplicativă

Aplicația este construită pe platforma **Spring Boot**, utilizând o arhitectură stratificată MVC. Diferențiatorul major al acestui proiect este utilizarea exclusivă a **Native SQL** în stratul de persistență, oferind performanță maximă și control direct asupra interogărilor.

3.1 Pachetul Repository (Acces Date)

Acest pachet constituie stratul de persistență al aplicației și asigură interfațarea directă cu baza de date SQL Server. Pentru a depăși limitările impuse de metodele standard generate de Spring Data JPA și pentru a optimiza performanța, am implementat majoritatea operațiilor utilizând **Native SQL**.

3.1.1 Căutare Avansată și Live Search

Una dintre funcționalitățile centrale este motorul de căutare "Live Search", care permite utilizatorilor să găsească rapid înregistrări tastând doar câteva litere. Aceasta a fost implementată printr-o interogare SQL care concatenează mai multe coloane relevante (Nume, Prenume, CNP, Telefon etc.) și verifică potrivirea cu termenul introdus.

Detaliu tehnic: Utilizarea clauzei `COLLATE Latin1_General_100_CI_AI` este crucială pentru a asigura insensibilitatea la majuscule și diacritice (ex: căutarea "ion" va găsi și "Ion", și "ION", și "Ionuț").

```
59
60      // === SEARCH INTELIGENT (_100_CI_AI) ===
61      // Cautare dupa inceputul numelui, prenumelui sau CNP-ului
62      @Query(value = "SELECT * FROM Persoane WHERE " +  1 usage  vladiv5
63              "CONCAT(COALESCE(nume, ''), ' ', COALESCE(prenume, ''), ' ', COALESCE(cnp, ''), ' ', COALESCE(telefon, '')) " +
64              "COLLATE Latin1_General_100_CI_AI " +
65              "LIKE CONCAT(:termen, '%')", nativeQuery = true)
66      List<Persoana> cautaDupaInceput(@Param("termen") String termen);
67
```

Figura 2: Implementarea metodei `cautaDupaInceput` în `PersoanaRepository`, demonstrând utilizarea CONCAT și COLLATE pentru căutare flexibilă.

3.1.2 Rapoarte Statistice și Agregare

Pentru a genera rapoartele din dashboard-ul aplicației (ex: "Top Zone de Risc", "Top Polițiști"), am scris interogări complexe care utilizează funcții de agregare (`COUNT`, `SUM`, `AVG`) și clauze `GROUP BY`. Aceste calcule sunt realizate direct în baza de date, returnând doar rezultatul final către aplicație, ceea ce reduce semnificativ latența.

```
67
68      // 1. Top străzi periculoase (Cele mai multe incidente)
69      @Query(value = "" +  1 usage  new *
70              "SELECT adr.strada, adr.localitate, COUNT(i.id_incident) as nr_incidente " +
71              "FROM adrese adr " +
72              "JOIN incidente i ON adr.id_adresa = i.id_adresa_incident " +
73              "WHERE (:startDate IS NULL OR i.data_emitere >= :startDate) " +
74              "  AND (:endDate IS NULL OR i.data_emitere <= :endDate) " +
75              "GROUP BY adr.strada, adr.localitate " +
76              "ORDER BY nr_incidente DESC",
77              nativeQuery = true)
78      List<Map<String, Object>> getTopStraziIncidente(
79              @Param("startDate") LocalDateTime startDate,
80              @Param("endDate") LocalDateTime endDate
81      );
82
```

Figura 3: Interogare complexă în `IncidentRepository` pentru identificarea zonelor cu risc ridicat, folosind agregare și join-uri multiple.

3.1.3 Analiză Comparativă (Subcereri)

Pentru rapoartele avansate, precum identificarea "Zilelor Critice" (zile cu activitate peste medie) sau a "Agenților Severi", am utilizat subcereri SQL (Subqueries). Acestea permit compararea unei valori agregate a unui grup cu o medie globală calculată dinamic.

```
115     // 4. Zile Critice (Zile cu activitate infracțională peste medie)
116     // Utilizează Subquery pentru calculul mediei dinamice
117     @Query(value = "" +  1 usage  new *
118             "SELECT CAST(i.data_emitere AS DATE) as ziua, COUNT(*) as nr_incidente " +
119             "FROM Incidente i " +
120             "WHERE (:startDate IS NULL OR i.data_emitere >= :startDate) " +
121             "  AND (:endDate IS NULL OR i.data_emitere <= :endDate) " +
122             "GROUP BY CAST(i.data_emitere AS DATE) " +
123             "HAVING COUNT(*) > (" +
124             "    SELECT AVG(sub.zilnic) FROM (" +
125             "        SELECT COUNT(*) as zilnic " +
126             "        FROM Incidente i2 " +
127             "        WHERE (:startDate IS NULL OR i2.data_emitere >= :startDate) " +
128             "          AND (:endDate IS NULL OR i2.data_emitere <= :endDate) " +
129             "        GROUP BY CAST(i2.data_emitere AS DATE)" +
130             "    ) as sub" +
131             ") " +
132             "ORDER BY nr_incidente DESC",
133             nativeQuery = true)
134     List<Map<String, Object>> getZileCritice(
135             @Param("startDate") LocalDateTime startDate,
136             @Param("endDate") LocalDateTime endDate
137     );
```

Figura 4: Utilizarea subcererilor SQL în `IncidentRepository` pentru detectarea anomaliilor statistice (zile cu activitate peste medie).

3.1.4 Operații CRUD Native

Pentru a avea un control granular asupra tranzacțiilor și a evita comportamentele implicite ale Hibernate care pot fi ineficiente în anumite scenarii, operațiile de `INSERT`, `UPDATE` și `DELETE` au fost implementate manual.

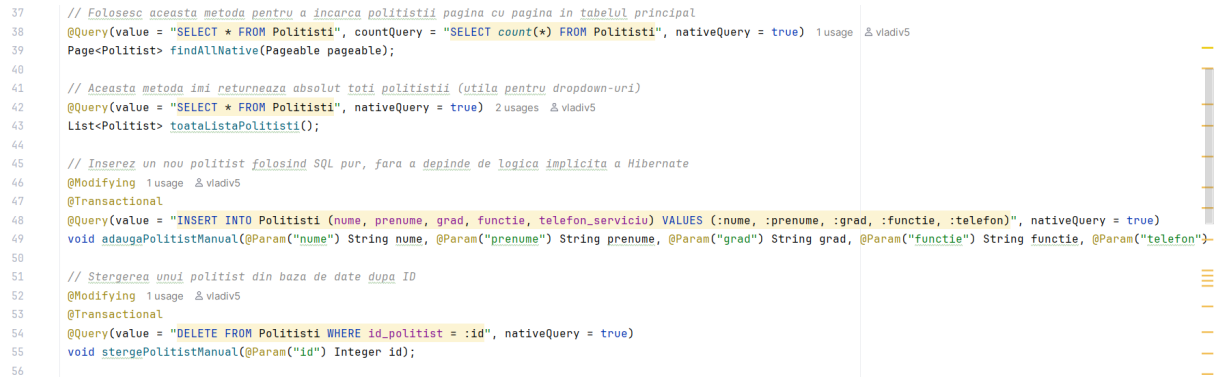


Figura 5: Metode tranzacționale native pentru modificarea datelor în `PolitistRepository`.

3.2 Pachetul Controller

Acest pachet reprezintă "creierul" aplicației, orchestrând fluxul de date între interfața utilizator (Frontend) și baza de date (Repository). Controllerele expun un API REST bine structurat și implementează validările și securitatea.

3.2.1 Validarea Datelor

Securitatea și integritatea datelor sunt prioritare. Înainte ca orice dată să fie trimisă către baza de date, aceasta trece printr-un proces riguros de validare implementat în metode private. Dacă validarea eșuează, controller-ul returnează un cod HTTP 400 (Bad Request).

Criterii de validare implementate:

- **Sintactice:** Regex pentru telefon, nume (doar litere).
- **Logice:** Data nașterii nu poate fi în viitor; suma amenzii trebuie să fie pozitivă.
- **Matematice:** Algoritmul de control al CNP-ului și regula ca amenzile să fie multiplu de 25 RON.

```java
// --- HELPER VALIDARE (LOGICA MATEMATICĂ) ---
private Map<String, String> valideazaAmenda(AmendaRequest req) {  2 usages  vladiv5
    Map<String, String> errors = new HashMap<>();

    // 1. Validare Motiv
    if (req.motiv == null || req.motiv.trim().isEmpty()) {
        errors.put("motiv", "Motivul amenzii este obligatoriu!");
    } else if (req.motiv.length() > 255) {
        errors.put("motiv", "Maxim 255 de caractere!");
    } else if (req.motiv.matches( regex: ".*\\d.*")) {
        errors.put("motiv", "Motivul nu poate conține cifre!");
    }

    // 2. Validare Suma (Interval 50-3000, Multiplu de 25)
    if (req.suma == null) {
        errors.put("suma", "Suma este obligatorie!");
    } else {
        BigDecimal min = new BigDecimal( val: "50");
        BigDecimal max = new BigDecimal( val: "3000");
        BigDecimal multiplu = new BigDecimal( val: "25");

        boolean inRange = req.suma.compareTo(min) >= 0 && req.suma.compareTo(max) <= 0;
        boolean isMultiplu = req.suma.remainder(multiplu).compareTo(BigDecimal.ZERO) == 0;

        if (!inRange || !isMultiplu) {
            errors.put("suma", "Suma trebuie să fie multiplu de 25 și între 50 și 3000 RON.");
        }
    }
```

Figura 6: Metoda privată de validare din `AmendaController` care impune reguli stricte asupra datelor de intrare.

```java
// --- VALIDARI LOGICE MATEMATICE ---
private boolean isCnpValidMatematic(String cnp) {  1 usage  vladiv5
    String constanta = "279146358279";
    int suma = 0;
    for (int i = 0; i < 12; i++) {
        int cifraCNP = Character.getNumericValue(cnp.charAt(i));
        int cifraConst = Character.getNumericValue(constanta.charAt(i));
        suma += cifraCNP * cifraConst;
    }
    int rest = suma % 11;
    int cifraControl = (rest == 10) ? 1 : rest;
    int cifraControlReala = Character.getNumericValue(cnp.charAt(12));
    return cifraControl == cifraControlReala;
}
```

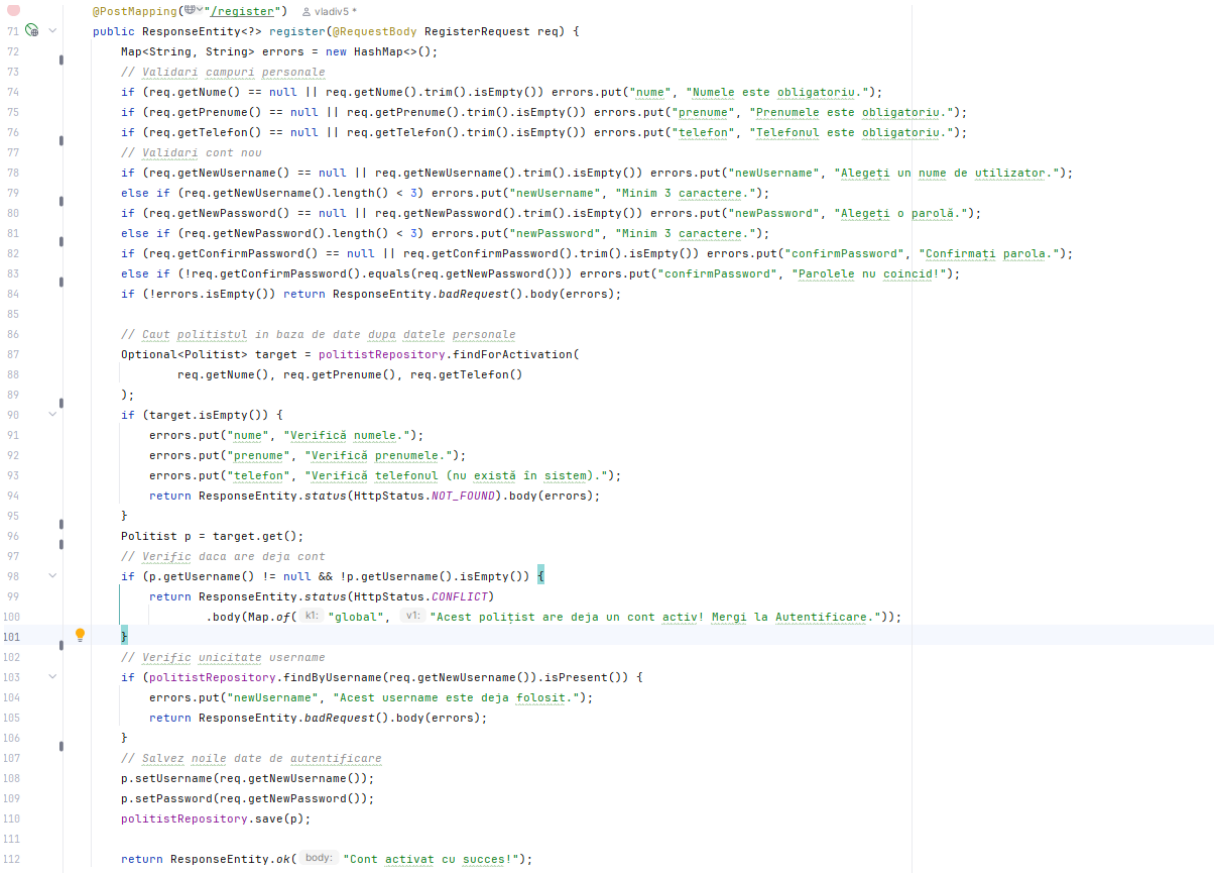
Figura 7: Algoritmul de verificare a validității matematice a CNP-ului implementat în `PersoanaController`.

3.2.2 Sistemul de Autentificare și Securitate

`AuthController` gestionează accesul în aplicație. Metoda de Login nu se bazează doar pe interogarea bazei de date, ci implementează o verificare suplimentară în Java folosind `.equals()` pentru a asigura **Case Sensitivity**. Metoda de Register permite activarea conturilor pentru personalul deja existent, verificând identitatea prin corelarea a trei factori: Nume, Prenume și Telefon.

```java
    // === LOGIN (Case Sensitive) ===
    @PostMapping("/login")  vladiv5
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
        Map<String, String> errors = new HashMap<>();

        if (loginRequest.getNume() == null || loginRequest.getNume().trim().isEmpty()) {
            errors.put("nume", "Introduceți numele de utilizator!");
        }
        if (loginRequest.getParola() == null || loginRequest.getParola().trim().isEmpty()) {
            errors.put("parola", "Introduceți parola!");
        }

        if (!errors.isEmpty()) return ResponseEntity.badRequest().body(errors);

        // Verificare BACKDOOR ADMIN pentru testare rapida
        if ("admin".equals(loginRequest.getNume()) && "admin".equals(loginRequest.getParola())) {
            Politist adminFals = new Politist();
            adminFals.setIdPolitist(-1);
            adminFals.setNume("Developer");
            adminFals.setPrenume("Admin");
            adminFals.setGrad("SuperUser");
            adminFals.setFunctie("Developer");
            return ResponseEntity.ok(adminFals);
        }

        // Caut user in baza de date
        Optional<Politist> politistOpt = politistRepository.findByUsername(loginRequest.getNume());

        if (politistOpt.isPresent()) {
            Politist p = politistOpt.get();
            // Verificare stricta Java pentru litere mari/mici
            if (p.getUsername().equals(loginRequest.getNume()) &&
                    p.getPassword().equals(loginRequest.getParola())) {
                return ResponseEntity.ok(p);
            }
        }

        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body(Map.of( k1: "global",  v1: "Date incorecte (atenție la majuscule)!"));
    }
```

Figura 8: Logica de autentificare securizată (Case Sensitive) din `AuthController`.



```java
@PostMapping("/register")  vladiv5 *
public ResponseEntity<?> register(@RequestBody RegisterRequest req) {
    Map<String, String> errors = new HashMap<>();
    // Validari campuri personale
    if (req.getNume() == null || req.getNume().trim().isEmpty()) errors.put("nume", "Numele este obligatoriu.");
    if (req.getPrenume() == null || req.getPrenume().trim().isEmpty()) errors.put("prenume", "Prenumele este obligatoriu.");
    if (req.getTelefon() == null || req.getTelefon().trim().isEmpty()) errors.put("telefon", "Telefonul este obligatoriu.");
    // Validari cont nou
    if (req.getNewUsername() == null || req.getNewUsername().trim().isEmpty()) errors.put("newUsername", "Alegeți un nume de utilizator.");
    else if (req.getNewUsername().length() < 3) errors.put("newUsername", "Minim 3 caractere.");
    if (req.getNewPassword() == null || req.getNewPassword().trim().isEmpty()) errors.put("newPassword", "Alegeți o parolă.");
    else if (req.getNewPassword().length() < 3) errors.put("newPassword", "Minim 3 caractere.");
    if (req.getConfirmPassword() == null || req.getConfirmPassword().trim().isEmpty()) errors.put("confirmPassword", "Confirmați parola.");
    else if (!req.getConfirmPassword().equals(req.getNewPassword())) errors.put("confirmPassword", "Parolele nu coincid!");
    if (!errors.isEmpty()) return ResponseEntity.badRequest().body(errors);

    // Caut politistul in baza de date dupa datele personale
    Optional<Politist> target = politistRepository.findForActivation(
            req.getNume(), req.getPrenume(), req.getTelefon()
    );
    if (target.isEmpty()) {
        errors.put("nume", "Verifică numele.");
        errors.put("prenume", "Verifică prenumele.");
        errors.put("telefon", "Verifică telefonul (nu există în sistem).");
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(errors);
    }
    Politist p = target.get();
    // Verific daca are deja cont
    if (p.getUsername() != null && !p.getUsername().isEmpty()) {
        return ResponseEntity.status(HttpStatus.CONFLICT)
                .body(Map.of( k1: "global",  v1: "Acest polițist are deja un cont activ! Mergi la Autentificare."));
    }
    // Verific unicitate username
    if (politistRepository.findByUsername(req.getNewUsername()).isPresent()) {
        errors.put("newUsername", "Acest username este deja folosit.");
        return ResponseEntity.badRequest().body(errors);
    }
    // Salvez noile date de autentificare
    p.setUsername(req.getNewUsername());
    p.setPassword(req.getNewPassword());
    politistRepository.save(p);

    return ResponseEntity.ok( body: "Cont activat cu succes!");
```

Figura 9: Procesul de activare a conturilor și verificare a identității în `AuthController`.

3.2.3 Logica "Smart Delete"

Una dintre cele mai complexe funcționalități este sistemul de ștergere inteligentă. Înainte de a permite ștergerea unei entități, controller-ul apelează endpoint-ul dedicat `/verifica-stergere`. Acesta analizează dependențele și returnează un status:

- **BLOCKED:** Dacă există elemente active (incidente, amenzi neplătite).
- **WARNING:** Dacă există istoric (incidente închise).
- **SAFE:** Dacă ștergerea este sigură.

```java
// --- VERIFICARE STERGERE (LOGICA SMART DELETE REVIZUITĂ) ---
@GetMapping("/verifica-stergere/{id}")  vladiv5 *
public DeleteConfirmation verificaStergere(@PathVariable Integer id) {
    List<BlockingItem> listaTotala = new ArrayList<>();
    boolean hasRed = false;    // Blochează ștergerea (Active/Neplătite)
    boolean hasOrange = false; // Avertizează (Închise/Plătite) -> Arhivatele vor fi ignorate (Verde)

    // 1. Verificăm Amenzile
    List<Amenda> toateAmenzile = amendaRepository.findAllNativeByPolitist(id);
    for (Amenda a : toateAmenzile) {
        String status = a.getStarePlata();
        String desc = status + " - " + a.getSuma() + " RON";

        // Adaug în listă doar ca info
        listaTotala.add(new BlockingItem( tip: "Amendă", a.getIdAmenda(), desc));

        // Logica de decizie
        if ("Neplatita".equalsIgnoreCase(status)) {
            hasRed = true;
        } else if ("Platita".equalsIgnoreCase(status)) {
            hasOrange = true;
        }
        // "Anulata" este ignorată -> Rămâne Verde (Safe)
    }

    // 2. Verificăm Incidentele
    List<Incident> toateIncidentele = incidentRepository.findAllNativeByPolitist(id);
    for (Incident i : toateIncidentele) {
        String status = i.getStatus();
        String desc = i.getTipIncident() + " (" + status + ")";

        listaTotala.add(new BlockingItem( tip: "Incident", i.getIdIncident(), desc));

        // Logica de decizie
        if ("Activ".equalsIgnoreCase(status)) {
            hasRed = true;
        } else if ("Închis".equalsIgnoreCase(status)) {
            hasOrange = true;
        }
        // "Arhivat" este ignorat -> Rămâne Verde (Safe)
    }

    // 3. Returnăm rezultatul pe baza priorității (Roșu > Portocaliu > Verde)
    if (hasRed) {
        return new DeleteConfirmation(
                permisiuneStergere: false,
                severitate: "BLOCKED",
                titlu: "Ștergere Blocată",
                mesaj: "Polițistul are elemente ACTIVE (cazuri în lucru sau amenzi neplătite). Rezolvați-le înainte de ștergere!",
                listaTotala
        );
    } else if (hasOrange) {
        return new DeleteConfirmation(
                permisiuneStergere: true,
                severitate: "WARNING",
                titlu: "Atenție - Ștergere Istoric",
                mesaj: "Polițistul nu mai are cazuri active, dar există un istoric (dosare închise/amenzi plătite). Ștergerea este permisă, dar ireversibilă.",
                listaTotala
        );
    } else {
        // Aici ajungem dacă lista e goală SAU dacă are doar "Arhivat" / "Anulata"
        return new DeleteConfirmation(
                permisiuneStergere: true,
                severitate: "SAFE",
                titlu: "Ștergere Sigură",
                mesaj: "Nu există date critice asociate. Dosarele arhivate sau anulate vor fi șterse automat.",
                listaTotala
        );
    }
}
```

Figura 10: Algoritmul de decizie pentru Smart Delete în `PolitistController`, care analizează starea dosarelor asociate.

3.2.4 Generarea Rapoartelor și Filtrarea

`StatisticiController` acționează ca un agregator de date pentru dashboard. Acesta primește parametrii de filtrare (intervale de timp), îi validează și apelează metodele specializate din repository-uri.

```
59      // --- ENDPOINTS ---
60
61      @GetMapping("/top-politisti")  vladiv5
62      public List<Map<String, Object>> getTopPolitisti(@RequestParam(required=false) String start, @RequestParam(required=false) String end) {
63          return politistRepo.getTopPolitistiAmenzi(parseDate(start), parseDateEnd(end));
64      }
65
66      @GetMapping("/top-strazi")  vladiv5
67      public List<Map<String, Object>> getTopStrazi(@RequestParam(required=false) String start, @RequestParam(required=false) String end) {
68          return incidentRepo.getTopStraziIncidente(parseDate(start), parseDateEnd(end));
69      }
70
71      @GetMapping("/rau-platnici")  vladiv5
72      public List<Map<String, Object>> getRauPlatnici(@RequestParam(required=false) String start, @RequestParam(required=false) String end) {
73          return persoanaRepo.getRauPlatnici(parseDate(start), parseDateEnd(end));
74      }
75
76      @GetMapping("/amenzi-grad")  vladiv5
77      public List<Map<String, Object>> getAmenziGrad(@RequestParam(required=false) String start, @RequestParam(required=false) String end) {
78          return politistRepo.getStatisticiPerGrad(parseDate(start), parseDateEnd(end));
79      }
```

Figura 11: Endpoint-urile din `StatisticiController` care expun date agregate pentru graficele din frontend.

4 Interfața Grafică și Funcționarea Aplicației

Frontend-ul este realizat în **React.js**, oferind o experiență SPA (Single Page Application). Design-ul urmează tematica "Tactical Navy and Gold".

4.1 Dashboard (Panou Operativ)

Punctul de intrare în aplicație oferă o imagine de ansamblu prin grafice interactive (Pie Charts, Bar Charts) generate pe baza interogărilor complexe din backend. Utilizatorul poate filtra datele calendaristic.

4.2 Fluxuri de Lucru (Workflows)

4.2.1 1. Gestionarea Listelor (Paginare și Căutare)

Tabelele de date (Polițiști, Adrese, etc.) implementează **Server-Side Pagination**. Căutarea (Live Search) resetează automat paginația și interoghează endpoint-urile de căutare din API, oferind rezultate instantanee.

4.2.2 2. Fluxul "Boomerang" (Rezolvarea Conflictelor)

Acesta este un mecanism UX inovator pentru gestionarea erorilor de ștergere:

1. Utilizatorul încearcă să șteargă un Polițist.
2. Sistemul detectează un incident "Activ" și blochează ștergerea (Modal Roșu).
3. Utilizatorul apasă "Rezolvă" și este dus automat la pagina Incidentului.
4. După închiderea dosarului, sistemul îl readuce automat (*Boomerang*) la polițistul inițial pentru a finaliza ștergerea.

5 Aspecte privind Testarea și Îmbunătățiri

5.1 Strategia de Testare

- **Testare Backend (White Box):** Verificarea constrângerilor SQL și a logicii matematice (CNP/Amenzi) prin request-uri POSTMAN malițioase.
- **Testare Frontend (Black Box):** Validarea fluxurilor de utilizator, în special a mecanismului "Boomerang" și a feedback-ului vizual (Toast Notifications).
- **User Acceptance Testing:** Testarea cu utilizatori externi a dus la implementarea mesajelor de eroare prietenoase ("Nu există date pentru perioada selectată").

5.2 Direcții de Dezvoltare

1. **Integrare Google Maps API:** Vizualizarea "Hotspots" (zone cu infracționalitate ridicată) pe hartă.
2. **Generare PDF:** Exportarea automată a proceselor verbale de amendă.
3. **Roluri (RBAC):** Implementarea distincției între Admin și Ofițer pentru drepturi de ștergere.
4. **Arhivare Automată:** Job-uri programate pentru mutarea incidentelor vechi în tabele de arhivă.
5. **Audit Logs:** Urmărirea istorică a tuturor modificărilor pentru transparență.