Sistem Programări Ghișeu

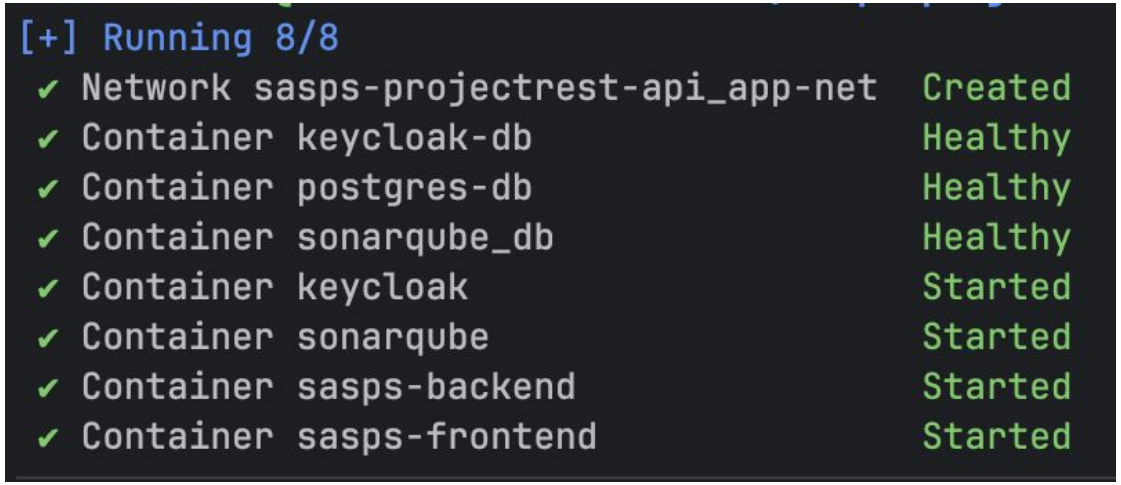
MILESTONE 3: Frontend, Security & Design Patterns

Echipa: Mihalcea Marius • Lebada Daria-Cristiana • Dragomir Andrei-Mihai

Arhitectura

Sistemul este complet operațional într-un mediu containerizat.

- **Docker Orchestration:** Toate serviciile (Frontend, Backend, DB) pornesc printr-o singură comandă.
- **Backend:** Spring Boot REST API conectat la PostgreSQL.
- **Frontend:** Angular.
- **Security:** Keycloak ca server de autorizare centralizat.



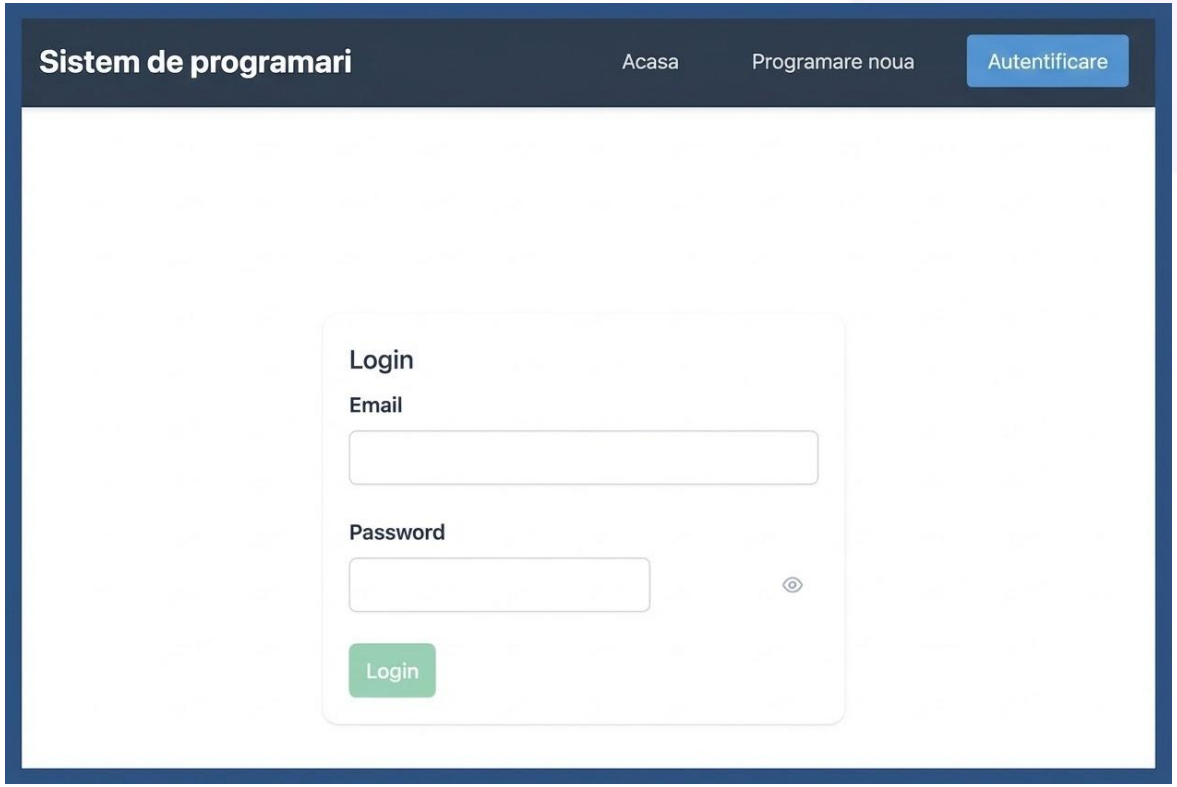
Securitate: Keycloak Integration

STATUS: IMPLEMENTAT

Autentificare si Roluri

Am finalizat configurarea securității folosind protocolul OpenID Connect.

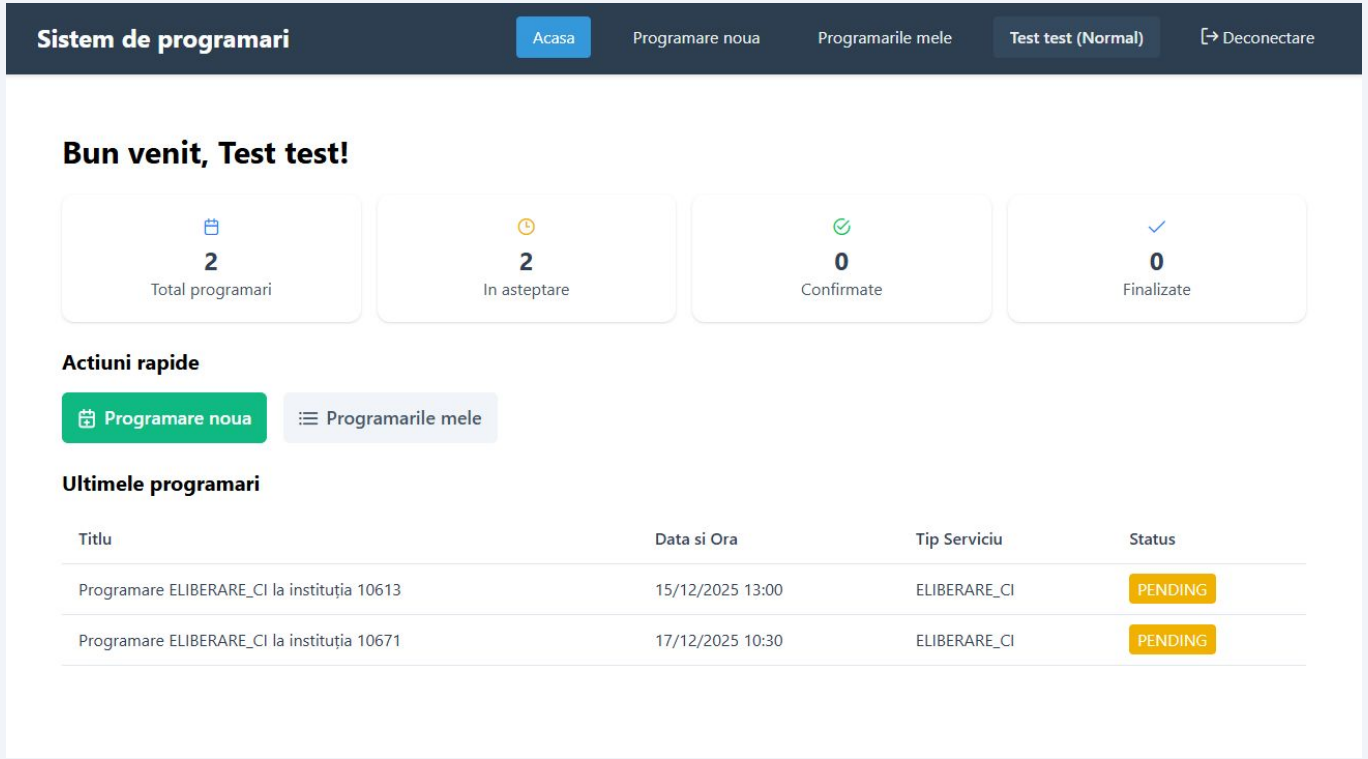
- **Realm "sasps-realm":** Configurat automat prin scripturi bash la pornirea containerului.
- **Role-Based Access Control (RBAC):**
  ADMIN - Gestionare instituții și utilizatori.
  USER - Creare și vizualizare programări proprii.
- **Securizarea Endpoint-urilor:** Backend-ul validează token-urile JWT emise de Keycloak.



Frontend Implementat

STATUS: FUNCȚIONAL

- **User Dashboard:** Vizualizare istoric programări și creare programare nouă.
- **Admin Dashboard:** Administrare utilizatori și vizualizare statistici.
- **Validare Formulare:** Feedback vizual imediat pentru utilizator.

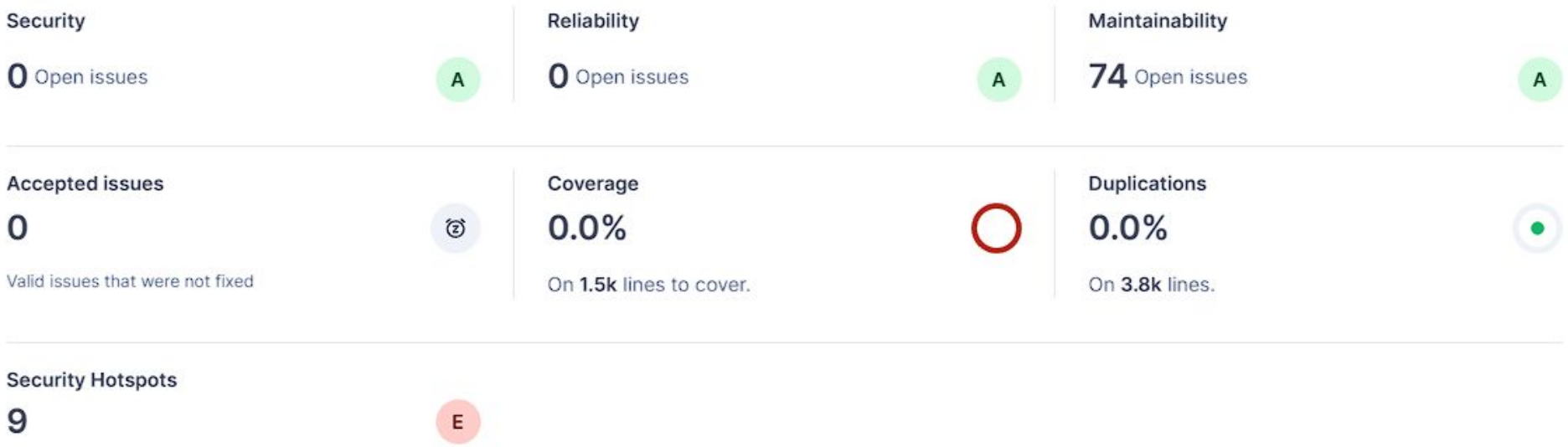


Probleme Fără Design Patterns

Lipsa pattern-urilor cauzează cod rigid și dificil de extins/ testat.

Probleme majore includ: Duplicare cod, tight coupling, încălcarea principiilor SOLID.

SonarQube a indicat probleme de mentenanță și securitate.



Design patterns

Analiza lipsei de design patterns

Am analizat manual lipsa folosirii design patternurilor.

- **27 locații unde lipsesc design patterns**
- **43 clase analizate - toate cu probleme structurale**
- **~970 linii de cod afectat de lipsa patterns**
- **8 tipuri de design patterns critice absente**

```
Institution.InstitutionType institutionType = institution.getType();
if (institutionType == Institution.InstitutionType.ANAF) {
    sendViaGovEmailProvider(recipientEmail, emailContent);
} else if (institutionType == Institution.InstitutionType.PRIMARIA) {
    sendViaLocalGovProvider(recipientEmail, emailContent);
} else {
    sendViaGenericProvider(recipientEmail, emailContent);
}
```

```
Appointment appointment = new Appointment();
appointment.setInstitutionId(request.getInstitutionId());
appointment.setUserId(userId);
appointment.setInstitutionType(request.getInstitutionType());

String title = "Programare " + request.getServiceType() + " la instituția " + request.getInstitutionId();
appointment.setTitle(title);

appointment.setNotes(request.getNotes());
appointment.setAppointmentTime(request.getAppointmentTime());
```

Design patterns

Analiza lipsei de design patterns

Am analizat manual lipsa folosirii design patternurilor.

- **Probleme de extindere și cuplare strictă**

```
private void validateAppointmentRequest(AppointmentRequest request) {
    if (request.getInstitutionId() == null) {
        throw new IllegalArgumentException(s: "Institution ID is required");
    }
    if (request.getAppointmentTime() == null) {
        throw new IllegalArgumentException(s: "Appointment time is required");
    }
    if (request.getCustomerName() == null || request.getCustomerName().isBlank()) {
        throw new IllegalArgumentException(s: "Customer name is required");
    }
    if (request.getCustomerEmail() == null || !request.getCustomerEmail().contains(s: "@")) {
        throw new IllegalArgumentException(s: "Valid email is required");
    }
    if (request.getServiceType() == null || request.getServiceType().isBlank()) {
        throw new IllegalArgumentException(s: "Service type is required");
    }
}
```

```
// Tightly coupled email sending
try {
    User user = userRepository.findById(appointment.getUserId()).orElse(null);
    if (user != null && user.getEmailNotificationsEnabled()) {
        Institution institution = institutionRepository.findById(...).orElse(null);
        emailService.sendAppointmentConfirmationEmail(user, appointment, institutionName);
    }
} catch (Exception e) {
    System.err.println("Failed to send confirmation email: " + e.getMessage());
}
```

Refactorizări identificate

- **Strategy** - Export programări (CSV/PDF) 220+
- **Adapter** - Email/SMS provideri 180+
- **Factory** - Creare Appointment/Institution 150+
- **Strategy** - Business rules & validări 100+
- **Chain of Responsibility** - Security checks 40+
- **Template Method** - Email templates 200+
- **Observer** - Event notifications 50+
- **Builder** - Obiecte complexe 120+
- **Facade** - Service orchestration 30+
- **Singleton** - Cache management 30+
- **Decorator** - Logging 40+

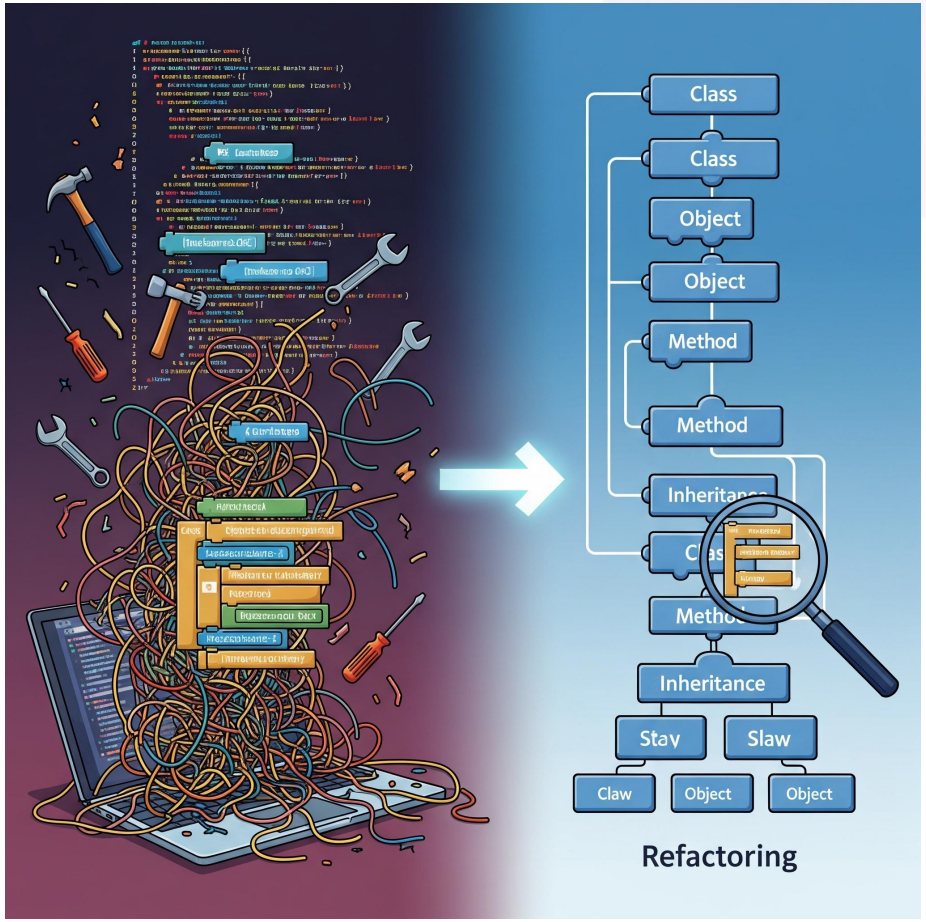
Refactorizare – Primele Rezultate

STATUS: ÎN DESFĂȘURARE

Analiză și Direcții Noi

După implementarea funcționalităților de bază și rularea analizei SonarQube, am identificat zone critice care necesită refactorizare.

- **Problema Identificată:** controlere cu logică excesivă și reguli de business duplicate în servicii.
- **Obiectiv:** Trecerea de la o abordare procedurală la una orientată pe obiecte și evenimente.
- **Acțiune:** Am început izolarea logicii de creare a obiectelor și a algoritmilor de validare.



Design Patterns în curs de implementare

Creational & Behavioral

Planul de restructurare a codului backend:

- **Factory Method:** Pentru standardizarea creării programărilor.
- **Strategy:** Pentru a gestiona flexibil regulile diferite de validare per instituție.
- **Observer:** Pentru a decupla sistemul de notificări de fluxul principal.

STATUS: ÎN DESFĂȘURARE

Beneficii Anticipate

Prin aplicarea acestor tipare, urmărim:

- **Reducerea complexității** (eliminarea if/else imbricate).
- **Cuplare mai slabă** (ceea ce duce la testarea mai facilă).
- **Extensibilitate** (adăugarea unei instituții noi fără modificarea codului existent).

Next Steps: Milestone 4

Finalizare Implementare

PLANIFICARE

Următorul pas este finalizarea completă a tranziției către arhitectura bazată pe Design Patterns.

- Implementarea completă a interfețelor **Factory** și **Strategy**.
- Rescrierea completă a **NotificationService** folosind **Observer**.

Analiză Comparativă

OBIECTIV M4

Vom realiza o comparație directă între versiunea inițială și cea refactorizată.

- **SonarQube:** Comparare Complexitate (Before vs After).
- **Maintainability Index:** Evaluarea efortului necesar pentru extindere.
- **Linii de Cod:** Reducerea codului redundant.

Vă mulțumim!