WEB CRAWLER

PROIECT PROIECTAREA APLICAȚIILOR WEB

Autor:
Pop-Cristof Marian

2025

Cuprins

1. INTRODUCERE

1.1 Context General

Un web crawler este un program automat care navighează pe internet pentru a colecta și indexa informații din paginile web. Este esențial în funcționarea motoarelor de căutare, facilitând accesul rapid la conținut relevant. De asemenea, este util în analiza de date și monitorizarea conținutului online. Un crawler personalizat permite control asupra modului de colectare, frecvenței și tipului de date extrase. Acesta se poate adapta nevoilor specifice ale unui proiect.

Notă specifică asupra arhitecturii: Acest web crawler a fost conceput cu o logică similară unui motor de căutare global. Astfel, baza de date de pagini indexate este comună pentru toți utilizatorii. Dacă un utilizator nou creează o configurație de scanare pentru un site care a fost deja vizitat de un utilizator anterior, sistemul va identifica paginile existente în baza de date. Această abordare eficientizează procesul de indexare, evitând descărcarea redundantă a aceluiași conținut și permițând accesul rapid la informații deja colectate.

1.2 Obiective

Dezvoltarea acestei aplicații web vizează atingerea următoarelor obiective:

- **Funcționalitate de bază:** Navigarea și extragerea link-urilor din paginile web, pornind de la o listă inițială de URL-uri (seed URLs).
- **Deep Scan:** Capacitatea de a naviga recursiv prin link-urile găsite pentru a explora în profunzime site-urile.
- **Căutare în conținut:** Implementarea unei funcționalități de căutare a unor cuvinte cheie sau expresii în conținutul paginilor web scanate.
- **Persistența datelor:** Stocarea URL-urilor vizitate, a conținutului paginilor și a rezultatelor căutării într-o bază de date.
- **Interfață prietenoasă:** O interfață web intuitivă pentru configurarea crawler-ului și vizualizarea rezultatelor.

1.3 Lista MoSCoW

Prioritizarea cerințelor proiectului a fost realizată utilizând metoda MoSCoW, conform tabelului de mai jos:

Tabela 1.1: Lista MoSCoW pentru Web Crawler

Categorie	**Cerință**
Must-haves	• Pornirea crawler-ului de la o listă de URL-uri inițiale • Extracția link-urilor din paginile vizitate • Salvarea URL-urilor vizitate în baza de date • Funcționalitate de căutare a textului în conținutul paginilor • Afișarea rezultatelor căutării în interfața web
Should-haves	• Navigare recursivă (deep scan) cu o limită de adâncime configurabilă • Evitarea buclelor infinite (gestionarea URL-urilor deja vizitate) • Opțiuni pentru configurarea numărului de thread-uri (concurrency) • Excluderea anumitor tipuri de fișiere (ex: imagini, PDF-uri)
Could-haves	• Filtrare domenii (a permite sau a bloca anumite domenii) • Planificare rulări (scheduling) ale crawler-ului • Exportul rezultatelor în formate diferite (CSV, JSON) • Autentificare și autorizare pentru interfața de administrare
Won't-haves	• Integrare cu servicii cloud complexe (ex: AWS Lambda) • Analiză semantică avansată a conținutului • Suport pentru JavaScript dinamic (renderizare) la această etapă

1.4 Cazuri de Utilizare

.

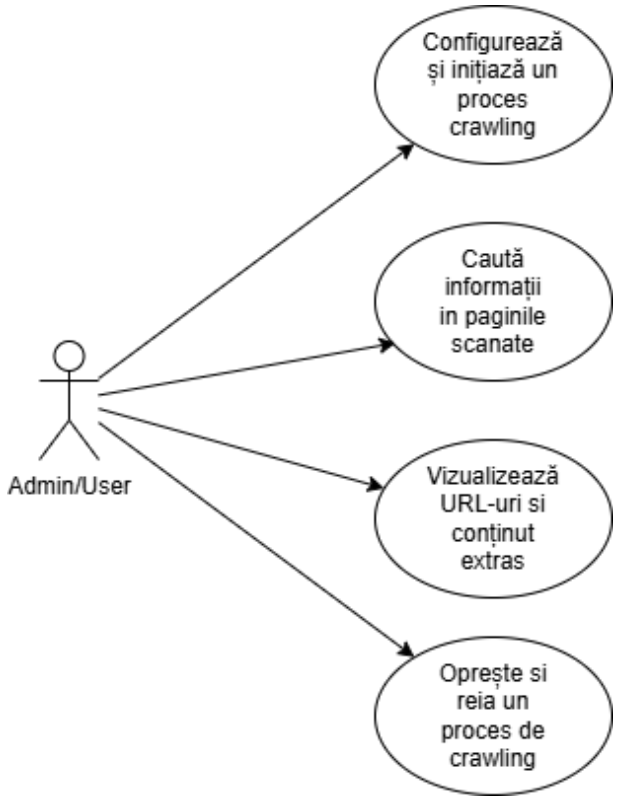

Figura 1.1: Cazuri de utilizare

2. ARHITECTURA

Arhitectura aplicației este concepută pentru a asigura o performanță de calitate, scalabilitate și o interfață utilizator prietenoasă. Bazată pe ecosistemul Spring Boot cu motorul de template-uri Thymeleaf, această arhitectură modernă îmbină tehnologii de tip server-side pentru a oferi o experiență coerentă și sigură utilizatorilor.

Spre deosebire de aplicațiile decuplate complet, această arhitectură utilizează randarea pe server pentru a asigura o integrare strânsă între logica de business și prezentare, optimizând în același timp securitatea datelor.

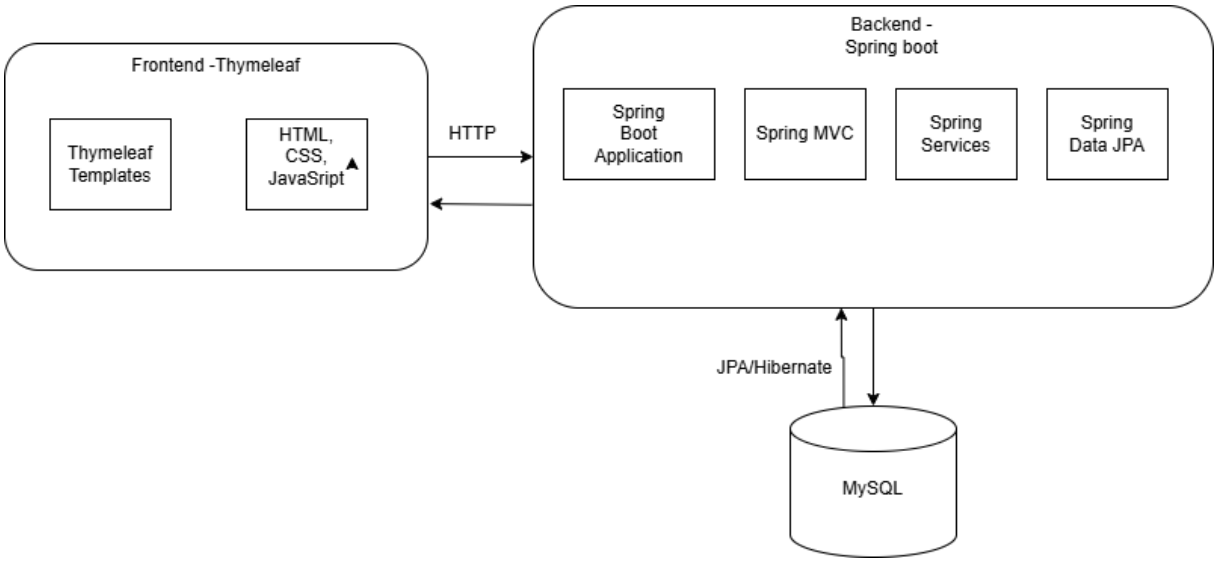

Figura 2.1: Arhitectura aplicației

2.1 Front-End Thymeleaf

Thymeleaf a fost ales pentru partea de front-end datorită integrării sale native cu Spring Boot și a capacității de a randa HTML natural. Această tehnologie permite crearea de pagini dinamice direct pe server, reducând complexitatea necesară gestionării stării pe client. Utilizarea fragmentelor Thymeleaf oferă o modularitate ridicată, rezultând într-un cod ușor de întreținut și extins, în timp ce protecția automată împotriva atacurilor de tip Cross-Site Scripting (XSS) asigură un nivel ridicat de securitate a interfeței.

2.2 Back-End Spring Boot

Spring Boot gestionează back-end-ul, furnizând un cadru ușor de utilizat pentru dezvoltarea serviciilor. Utilizând Spring Data JPA, interacțiunea cu baza de date devine transparentă și eficientă, eliminând necesitatea scrierii de cod SQL repetitiv. Spring Boot oferă, de asemenea, facilități pentru gestionarea dependențelor, configurarea automată a contextului și implementarea securității prin Spring Security, permițând dezvoltarea rapidă a unei logici de business complexe pentru procesul de crawling.

2.3 Comunicare între Front-End și Back-End

Comunicarea între front-end și back-end este gestionată prin intermediul arhitecturii Spring MVC. Aceasta permite transferul eficient de date prin obiectul *Model*, facilitând afișarea rezultatelor indexate în template-urile HTML. Pentru monitorizarea progresului de scanare în timp real, aplicația utilizează protocolul **WebSocket** (STOMP), care permite serverului să trimită notificări asincrone către browser fără a fi necesară reîncărcarea paginii.

2.4 Baza de Date MySQL

MySQL a fost ales ca sistem de gestionare a bazelor de date datorită fiabilității și performanței demonstrate în gestionarea volumelor mari de text. Arhitectura bazei de date este una centralizată, funcționând ca un index global (motor de căutare); odată ce o pagină este scanată de un utilizator, conținutul acesteia devine disponibil în baza de date comună pentru interogările viitoare ale oricărui alt utilizator. Interfața Spring Data JPA simplifică manipularea datelor, oferind o abordare elegantă pentru stocarea eficientă a URL-urilor și a conținutului extras.

2.5 Beneficii ale acestei arhitecturii

- **Eficiență în indexare:** Utilizarea unei baze de date partajate permite acumularea rapidă de informații, evitând scanarea redundantă a acelorași domenii de către utilizatori diferiți.
- **Securitate Server-Side:** Randarea paginilor pe server prin Thymeleaf asigură faptul că logica de validare și datele sensibile nu sunt expuse direct în browser.
- **Performanță optimizată:** Utilizarea WebSockets pentru actualizarea statusului crawler-ului reduce traficul HTTP și oferă o experiență de utilizare fluidă (real-time).
- **Scalabilitate și extensibilitate:** Modularitatea serviciilor Spring permite adăugarea facilă de noi funcționalități, precum exportul datelor sau analize statistice avansate asupra textului colectat.

3. IMPLEMENTARE FRONTEND

În dezvoltarea front-end-ului pentru aplicația Web Crawler, am adoptat o abordare orientată către eficiența randării pe server și interactivitate în timp real. Front-end-ul reprezintă interfața principală prin care utilizatorul configurează procesele de crawling și interoghează baza de date globală. Alegerea tehnologiilor potrivite a fost esențială pentru a asigura o experiență fluidă, în special în monitorizarea live a proceselor de scanare.

3.1 Tehnologii frontend

Pentru această aplicație, am optat pentru **Thymeleaf** ca motor principal de template-uri, datorită integrării simple cu Spring Boot și a capacității de a genera HTML valid care poate fi previzualizat ușor.

Pe lângă limbajele fundamentale ale web-ului, am utilizat următoarele resurse:

- **HTML5 și CSS3:** Pentru definirea structurii și a stilizării de bază a paginilor.
- **Bootstrap 5:** Framework-ul CSS utilizat pentru a crea o interfață responsivă, modernă și bazată pe componente predefinite (card-uri, tabele, butoane).
- **JavaScript:** Utilizat pentru logica de client, în special pentru gestionarea conexiunilor asincrone.

- **SockJS și Stomp.js:** Biblioteci folosite pentru implementarea protocolului WebSocket, permițând actualizarea automată a listelor de URL-uri în pagina de status fără reîncărcarea acesteia.

3.2 Structura componentelor

Spre deosebire de framework-urile de tip Single Page Application, structura frontendului în Spring Boot cu Thymeleaf este organizată în template-uri HTML modulare. Am utilizat conceptul de *fragments* pentru a extrage elementele comune (precum header-ul și bara de navigare), favorizând astfel coerența vizuală și ușurința în întreținere.

3.2.1 Componentele folosite

Principalele pagini (view-uri) care alcătuiesc interfața aplicației sunt centralizate în tabelul următor:

Tabela 3.1: Componentele Frontend pentru Web Crawler

Nume fișier	**Rolul componentei**	**Endpoint**
index.html	Dashboard central. Permite configurarea parametrilor de crawling și vizualizarea istoricului.	/
status.html	Panou de monitorizare live. Afișează progresul scanării curente prin WebSockets.	/status
searchResults.html	Interfața motorului de căutare. Permite interogarea bazei de date și filtrarea pe domenii.	/search
login.html	Formular securizat pentru autentificarea utilizatorilor.	/login
register.html	Formular pentru crearea unui cont nou în sistem.	/register

3.3 Utilizarea aplicației din perspectiva guest-ului

3.3.1 Pagina de login

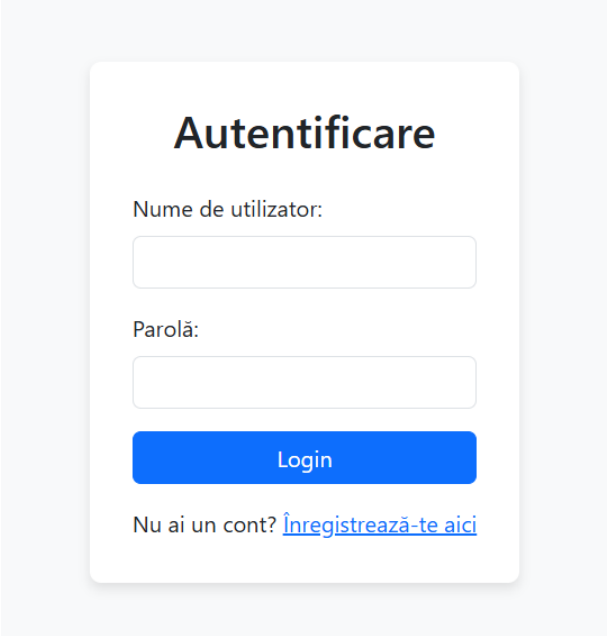

Figura 3.1: Conectare

3.3.2 Pagina de creare a unui cont nou

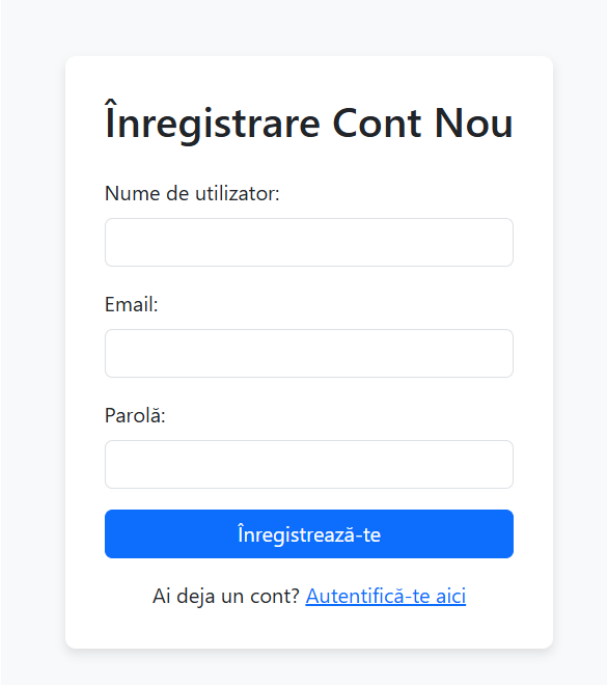

Figura 3.2: Creează cont

3.3.3 Pagina de pornire

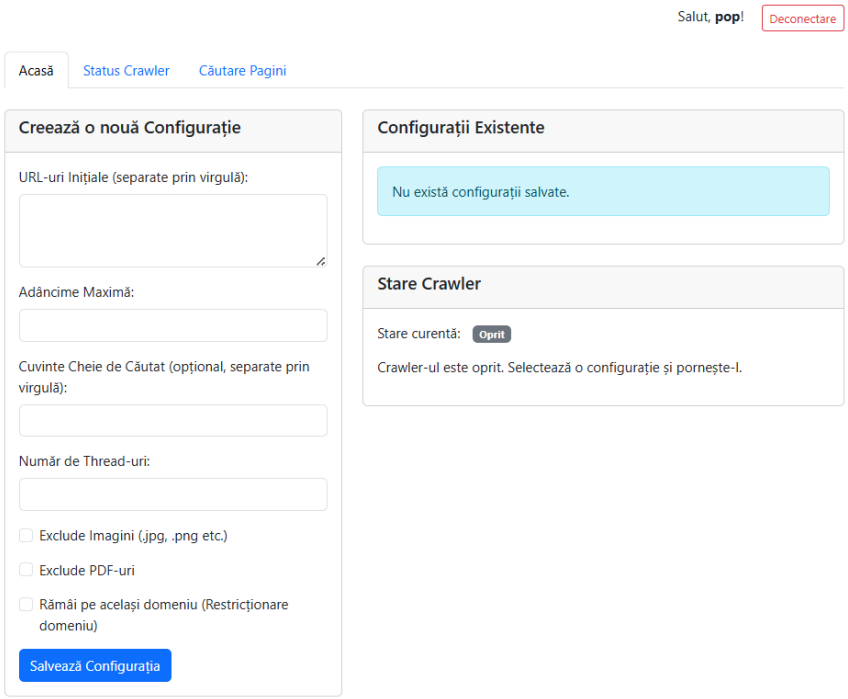

Figura 3.3: Dashboard

3.3.4 Pagina de monitorizare

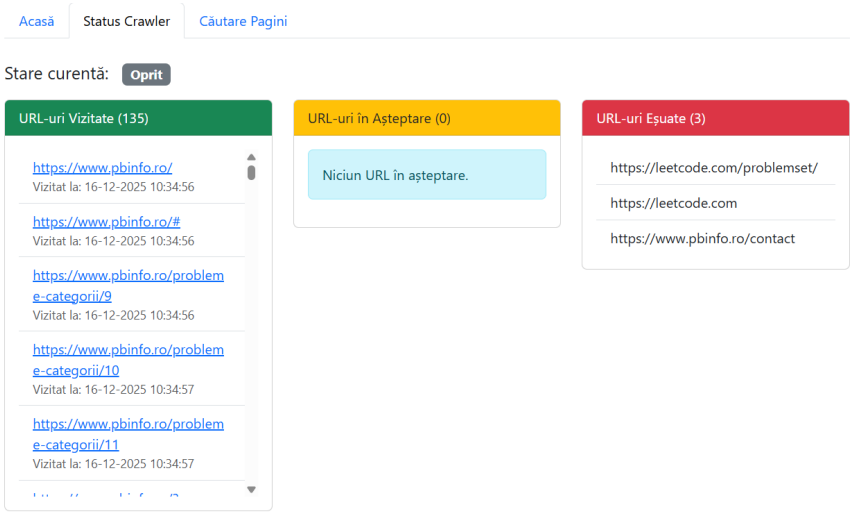

Figura 3.4: Monitorizare

3.3.5 Pagina de căutare

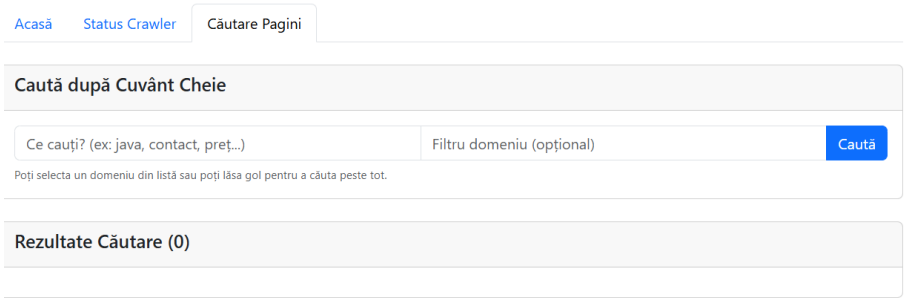

Figura 3.5: Căutare pagini

4. IMPLEMENTARE BACKEND

4.1 Tehnologii backend

Backend-ul aplicației de Web Crawler gestionează logica de explorare recursivă, procesarea paralelă a datelor și furnizează suportul necesar pentru funcționalitățile de indexare și căutare. Alegerea tehnologiilor a fost orientată spre eficiență în multithreading, scalabilitatea bazei de date și ușurința în gestionarea dependențelor.

4.1.1 Spring Boot

Spring Boot este framework-ul principal care facilitează dezvoltarea rapidă a aplicației prin abordarea „convention-over-configuration”. Acesta gestionează automat contextul aplicației și oferă un server Tomcat încorporat, eliminând necesitatea configurărilor manuale complexe.

Spring Data JPA

Java Persistence API (JPA) reprezintă standardul pentru gestionarea datelor relaționale. Integrarea în proiect permite manipularea informațiilor despre URL-uri și conținut sub formă de obiecte Java (POJO), abstractizând interogările SQL complexe.

Exemplu de implementare: Definirea entităților JPA pentru obiectele de domeniu, mapate direct în tabele MySQL:

```
@Entity
@Table(name = "crawler_configs")
@Data
public class CrawlerConfig {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(nullable = false, length = 4096)
    private String seedUrls;
    private Integer maxDepth;
    private Integer threadCount;
}
```

Listing 4.1: Entitatea CrawlerConfig

Interfața Repository pentru gestionarea operațiunilor CRUD:

```
public interface CrawlerConfigRepository extends JpaRepository<
    CrawlerConfig, Long> {
    List<CrawlerConfig> findByUser(User user);
}
```

Listing 4.2: Interfața CrawlerConfigRepository

Dependency Injection (DI)

Spring Framework utilizează Inversarea Controlului (IoC) pentru a injecta dependențele în obiecte la rulare. Acest lucru facilitează decuplarea componentelor și testarea ușoară.

Exemplu de implementare: Definirea serviciilor care beneficiază de DI prin adnotarea `@Service` și injectarea lor în controller:

```
@Controller
public class WebController {
    @Autowired
    private CrawlerService crawlerService;

    @Autowired
    private SearchService searchService;

    // Spring injecteaza automat implementarile acestor servicii
}
```

Listing 4.3: Injectarea serviciilor in WebController

Spring Security

Pentru securizarea proceselor de crawling și protejarea configurațiilor fiecărui utilizator, s-a implementat Spring Security.

- **Autentificare:** Utilizatorul se autentifică prin formular (Form Login).
- **Autorizare:** Accesul la rutele de scanare și căutare este permis doar utilizatorilor cu rolul `ROLE_USER`.
- **Criptare:** Parolele sunt stocate în siguranță folosind algoritmul `BCrypt`.

4.1.2 WebSockets

WebSockets reprezintă o tehnologie de comunicație bidirecțională între client și server. În această aplicație, este utilizată pentru a notifica interfața despre progresul scanării fără a reîncărca pagina.

Implementarea comunicației WebSocket:

1. **Conexiunea:** Browserul stabilește o conexiune cu serverul la endpoint-ul `/ws`.
2. **Procesarea:** Pe măsură ce `CrawlerService` procesează un URL, acesta colectează datele de status.
3. **Notificarea:** Serverul trimite obiectul JSON către topicul `/topic/crawlerStatus`.

4.1.3 MySQL

Am optat pentru MySQL ca sistem de gestionare a bazelor de date datorită stabilității și performanței excelente în lucrul cu volume mari de text (LONGTEXT pentru conținutul paginilor). Baza de date centralizează toate paginile vizitate, transformând aplicația într-un index global accesibil tuturor utilizatorilor.

4.2 Gestionarea cererilor HTTP

Aplicația se bazează pe rute bine definite pentru a separa logica de configurare, monitorizare și căutare.

Tabela 4.1: Endpoint-uri HTTP Web Crawler

Endpoint	**Metoda**	**Utilizare**
/login	GET/POST	Autentificare utilizator
/saveConfig	POST	Salvare setări crawler per utilizator
/startCrawler	POST	Inițiere proces asincron de scanare
/status	GET	Vizualizare URL-uri vizitate/în așteptare
/search	GET	Interogare motor de căutare
/export/json	GET	Descărcare rezultate în format JSON

5. TESTARE ȘI VALIDARE

În procesul de dezvoltare al aplicației Web Crawler, s-au efectuat câteva teste automate. Utilizarea ecosistemului Spring Boot permite o testare stratificată, combinând viteza testelor unitare cu acuratețea testelor de integrare.

5.1 Instrumente și Tehnologii de Testare

Pentru realizarea testelor, au fost utilizate următoarele framework-uri și biblioteci din ecosistemul Java:

- **JUnit 5:** Framework-ul principal utilizat pentru scrierea și rularea testelor prin aserțiuni.
- **Mockito:** O bibliotecă de „mocking” utilizată pentru a crea obiecte simulate, permițând izolarea claselor testate prin înlocuirea dependențelor reale cu versiuni controlate.

- **Spring Boot Test:** Oferă suport pentru testarea integrată, permițând încărcarea contextului aplicației pentru a verifica colaborarea componentelor.
- **MockMvc:** Componentă a Spring Test care simulează cererile HTTP către controllere, verificând rutarea și securitatea fără a porni un server real.

5.2 Testarea Unitară (Unit Testing)

Testele unitare verifică logica internă a metodelor în izolare totală, fără a interacționa cu baza de date sau rețeaua.

5.2.1 Validarea Logicii Crawler-ului (CrawlerServiceUnitTest)

În cadrul serviciului `CrawlerService`, testele s-au concentrat pe algoritmii de filtrare:

- **isExcluded:** Verifică dacă motorul de scanare ignoră corect fișierele de tip .jpg sau .pdf pentru a economisi resurse.
- **getDomain:** Testează capacitatea sistemului de a extrage domeniul rădăcină (ex: exemplu.ro) dintr-un URL complex, asigurând funcționarea filtrului de tip „Stay on domain".

5.2.2 Testarea Serviciului de Căutare (SearchServiceTest)

Acest test utilizează un obiect simulat (Mock) pentru `PageContentRepository` pentru a valida logica de business:

- **testSearchByKeyword_Success:** Verifică dacă serviciul returnează lista corectă de pagini atunci când este furnizat un cuvânt cheie valid.
- **testSearchByKeyword_EmptyKeyword:** Asigură că sistemul nu interoghează baza de date inutil dacă utilizatorul introduce un text gol.

5.3 Testarea de Integrare

Testele de integrare verifică dacă modulele aplicației colaborează corect cu regulile de securitate impuse de Spring Security.

5.3.1 Validarea Stratului Web (WebControllerIntegrationTest)

S-a utilizat adnotarea `@WebMvcTest` pentru a valida controllerul principal:

- **Redirecționarea Neautorizată:** Simulează un utilizator anonim și validează că sistemul răspunde cu codul HTTP 302, redirecționând către pagina de login.
- **Acces Autentificat:** Verifică dacă un utilizator logat (simulat prin `@WithMockUser`) primește acces la dashboard-ul aplicației.
- **Ruta de Căutare:** Testează fluxul de la trimiterea unui parametru prin URL până la randarea paginii `searchResults.html`.

6. CONCLUZII

Proiectul prezentat a reușit implementarea unui sistem complet de tip Web Crawler integrat într-o aplicație web, demonstrând capacitatea tehnologiilor din ecosistemul Java de a gestiona procese asincrone complexe. Dezvoltarea acestei platforme a oferit o perspectivă practică asupra provocărilor legate de indexarea datelor nestructurate și monitorizarea în timp real a resurselor de rețea.

6.1 Realizări Tehnice și Funcționale

Prin utilizarea framework-ului Spring Boot împreună cu bibliotecile specializate, s-au atins următoarele obiective majore:

- **Eficiență prin Paralelism:** Implementarea unui pool de fire de execuție (multithreading) în cadrul `CrawlerService` a permis scanarea simultană a mai multor URL-uri, reducând semnificativ timpul de procesare.
- **Interactivitate în Timp Real:** Integrarea protocolului WebSocket a transformat experiența utilizatorului dintr-una statică într-una dinamică, oferind feedback vizual instantaneu asupra progresului scanării.
- **Securitate și Izolare:** Utilizarea Spring Security a garantat protecția datelor și izolarea configurațiilor de crawling pentru fiecare utilizator în parte.
- **Indexare Globală:** Arhitectura bazată pe o bază de date MySQL comună permite funcționarea aplicației ca un motor de căutare, evitând scanarea redundantă a acelorași domenii.

6.2 Limitări Identificate

Soluția actuală prezintă anumite limitări inerente tehnologiilor alese:

- **Conținut Dinamic:** Deoarece biblioteca Jsoup parsează doar HTML-ul static primit de la server, aplicația nu poate indexa conținutul generat pe partea de client de către framework-uri precum React sau Angular.

6.3 Direcții Viitoare de Dezvoltare

Pentru a transforma proiectul într-o soluție industrială, se propun următoarele îmbunătățiri:

1. **Integrarea Selenium/Playwright:** Adăugarea unui „Headless Browser” ar permite randarea paginilor dinamice, eliminând limitarea actuală a Jsoup.
2. **Analiză Semantică (NLP):** Procesarea textului extras folosind algoritmi de procesare a limbajului natural pentru a clasifica automat paginile colectate.
3. **Arhitectură Distribuită:** Implementarea unui sistem de cozi de mesaje (ex: RabbitMQ) pentru a distribui URL-urile către mai multe instanțe de crawlere, permițând scanări la scară mare.

7. BIBLIOGRAFIE

1. **Spring Boot Documentation**, 2025, „Spring Boot Reference Guide", [Online]. Disponibil: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

2. **Jsoup Project**, 2025, „Jsoup Cookbook: Java HTML Parser", [Online]. Disponibil: https://jsoup.org/cookbook/.

3. **Thymeleaf Project**, 2025, „Using Thymeleaf: Java XML/XHTML/HTML5 Template Engine", [Online]. Disponibil: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html.

4. **Spring Security**, 2025, „Architecture and Security Filters", [Online]. Disponibil: https://docs.spring.io/spring-security/reference/index.html.

5. **Brian Goetz**, 2006, „Java Concurrency in Practice", Addison-Wesley Professional.

6. **Christopher D. Manning**, 2008, „Introduction to Information Retrieval", Cambridge University Press.

7. **Note de Curs**, 2025, „Proiectarea Aplicațiilor Web", Centrul Universitar Nord din Baia Mare.