## Universitatea Babeș-Bolyai din Cluj-Napoca

## Facultatea de Ştiinţe Economice şi Gestiunea Afacerilor

Proiect la Testarea Produselor Software

Realizat de:

### Damean Andrei-Stefan,

### Cirlig Geanina-Bianca,

### Ciocan Felicia-Ilinca,

### Colceru Dragos Constantin.

### Grupa 2, Echipa 25

### 17.01.2025

User Requirements

Se doreste o aplicatie Backend pentru a facilita functionarea site-ului Web al companiei. Precum am mentionat anterior este vorba despre despre o companie care vinde carti.

Site-ul prezinta un sistem de login si utilizatori. Dorim astfel sa implementam un sistem in care utilizatorii se pot conecta cu contul lor (email + parola). Este obligatoriu ca sesiunile sa fie securizate printr-un mecanism de token-uri JWT. Parolele utilizatorilor vor fi criptate. Utilizatorii trebuie sa poata sa caute carti dupa anumite filtre (Titlu, Autor sau gen literar). Acestia trebuie sa poata sa isi modifice datele profilului. Trebuie sa existe si un sistem de “Forgot Password”, pentru a permite userilor sa isi reseteze parola pe email, astfel usurand treaba echipei suport. Trebuie sa existe utilizatori de tip Administrator. Acestia au privilegii marite precum: adaugarea de carti, editarea lor si stergerea lor. Acestia pot crea, edita si sterge rafturi de carti. Acestea au scopul de a organiza cartile pentru utilizatori (utilizatorii isi pot adauga mai multe carti, iar o carte poate sa fie adaugata de mai multi utilizatori). Administratorii vor putea si sa adauge membrii noi si sa le modifice privilegii

Cartile trebuie sa prezinte cateva atribute obligatorii precum Titlul, Autor, Gen literar, Data publicarii si un Identificator unic. Acest lucru va facilita atat functionarea site-ului cat si experienta utilizatorilor. Prin aceste atribute, utilizatorii vor putea cauta carti si ulterior vor putea filtra rezultatele in functie de aceste atribute.

Functia rafturilor este de a permite utilizatorilor de adauga mai multe carti in functie de diferite criterii. Trebuie astfel sa existe un endpoint ce permite sa se creeze Rafturile iar apoi sa se adauge carti in ele. Va trebui sa exista si o functionalitate care sa afiseze cartile in functie de raft, dar si una pentru a sterge un raft.

Dorim sa existe un sistem de “logging” pentru a inregistra cererile si raspunsurile, aceste loguri vor putea sa fie folosite in Audituri precum si in anumite situatii pentru a da troubleshoot si remedia o problema de functionare. Este nevoie si de sisteme de monitorizare in timp real inclusiv a erorilor. Se doreste si raportare pentru detalii ce tin de datele site-ului precum numarul de utilizatori, carti, carti asociate rafturilor, numarul de rafturi etc. Necesitam si un endpoint ce transmite statistici zilnice/saptamanale/lunare.

Aplicatia trebuie sa aiba o disponibilitate de 5 de 9 (99.999%). Trebuie sa aiba raspunsuri rapide (<500ms pentru majoritatea cautarilor) si sa permita cel putin 1000 de utilizatori simultan. Trebuie sa permite si scalarea in orizontala pentru perioadele cu un numar mai mare de utilizatori decat in mod obisnuit.

Precum am mentionat mai sus securitatea datelor utilizatorilor trebuie sa existe astfel parolele acesotra vor fi criptate iar sesiunile securizate. Pentru a indeplini cerintele anterioare de disponibilitate este nevoie de limitari de rata (rate limiting). Trebuie sa existe protectie impotriva atacurilor de tip SQL injections precum si alte atacuri pe partea de backend.

In cazul erorilor trebuie sa existe mesaje clare care sa indice o eroare, de exemplu: “Cartea nu a fost gasita”. Trebuie sa exista o gestiune pentru exceptiile frecvente precum “404”, “400”, “500” etc.

Vom avea mapari pentru a putea prelua/edita si sterge datele de pe site, astfel cand un utilizator va trimite cereri precum POST sau GET partea de backend va sti ce mapare sa foloseasca pentru a incheia operatiunea si a transmite output-ul catre partea de frontend.

Aplicatia prezentata este o solutie backend construita in Java, utilizand framework-ul Spring Boot, fiind un exemplu reprezentativ al arhitecturii moderne bazate pe straturi. Aceasta include componente esentiale pentru gestionarea utilizatorilor, cartilor si rafturilor, oferind functionalitati de baza pentru o aplicatie de management al resurselor. In cele ce urmeaza, vom analiza în detaliu structura si specificatiile aplicatiei.

### **Aplicatia Backend: Analiza si Specificatii Tehnice**

#### **Arhitectura Aplicatiei**

Aplicatia urmeaza o arhitectura tipica MVC (Model-View-Controller), cusepararea clara a responsabilităților. Aceasta include straturi bine definite pentru gestionarea cererilor HTTP (controller-ele), logica de afaceri (serviciile) și interacțiunea cu baza de date (repozitoriile). Fiecare componenta joaca un rol esential în functionarea eficienta si modulara a aplicatiei.

Aplicatia prezentata este o solutie backend construita in Java, utilizand framework-ul Spring Boot, fiind un exemplu reprezentativ al arhitecturii moderne bazate pe straturi. Aceasta include componente esentiale pentru gestionarea utilizatorilor, cartilor si rafturilor, oferind functionalitati de baza pentru o aplicatie de management al resurselor. In cele ce urmeaza, vom analiza în detaliu structura si specificatiile aplicatiei.

#### **Structura maparii entitatilor:**

1. **Book (Carte):**
   * **Atribute:**
     + **ID:** Identificator unic al cartii.
     + **Titlu:** Titlul cartii.
     + **Autor:** Autorul cartii.
     + **Gen literar:** Genul literar al cartii.
     + **Anul publicarii:** Anul în care a fost publicată cartea.
     + **Descriere:** O scurtă descriere a cărții.
     + **ISBN:** Codul unic de identificare al cărții.
     + **Stare:** Starea curentă a cărții (disponibilă, împrumutată, rezervată etc.).
   * **Relații:**
     + Un raft poate conține mai multe cărți.
     + Poate avea multiple recenzii, fiecare asociată cu un utilizator și o evaluare.
2. **Shelf (Raft):**
   * **Atribute:**
     + **ID:** Identificator unic al raftului.
     + **Nume:** Numele raftului sau locația sa fizică în bibliotecă.
   * **Relații:**
     + Poate conține una sau mai multe cărți.
3. **User (Utilizator):**
   * **Atribute:**
     + **ID:** Identificator unic al utilizatorului.
     + **Nume utilizator:** Numele de utilizator al utilizatorului.
     + **Parolă:** Parola utilizatorului (criptată pentru securitate).
     + **Rol:** Rolul utilizatorului în sistem (ex: utilizator standard, administrator).
   * **Relații:**
     + Poate avea multiple recenzii asociate.

#### **Operațiuni CRUD (Create, Read, Update, Delete):**

1. **Cărți (Books):**
   * **Creare (Create):**
     + Endpoint pentru adăugarea unei noi cărți în sistem. Include validarea datelor și salvarea cărții în baza de date.
   * **Citire (Read):**
     + Endpoint pentru obținerea detaliilor despre o carte specifică în funcție de ID-ul său unic.
     + Endpoint pentru căutarea cărților după titlu, autor, gen literar sau orice alt criteriu relevant.
   * **Actualizare (Update):**
     + Endpoint pentru modificarea detaliilor unei cărți existente în funcție de ID-ul său unic. Asigură actualizarea informațiilor și validarea corectitudinii acestora.
   * **Ștergere (Delete):**
     + Endpoint pentru ștergerea unei cărți din sistem în funcție de ID-ul său unic. Include și gestionarea relațiilor cu rafturile și recenziile asociate.
2. **Rafturi (Shelves):**
   * **Creare (Create):**
     + Endpoint pentru crearea unui nou raft în sistem. Permite specificarea numelui sau locației raftului și gestionarea cărților asociate.
   * **Citire (Read):**
     + Endpoint pentru obținerea listei de rafturi disponibile în sistem. Include detalii despre cărțile plasate pe fiecare raft.
   * **Actualizare (Update):**
     + Endpoint pentru modificarea detaliilor unui raft existent în funcție de ID-ul său unic. Permite adăugarea sau eliminarea cărților de pe raft și actualizarea informațiilor asociate.
   * **Ștergere (Delete):**
     + Endpoint pentru ștergerea unui raft din sistem în funcție de ID-ul său unic. Asigură gestionarea corectă a cărților și a altor entități asociate cu raftul.
3. **Utilizatori (Users):**
   * **Creare (Create):**
     + Endpoint pentru înregistrarea unui nou utilizator în sistem. Include validarea datelor de autentificare și salvarea utilizatorului în baza de date.
   * **Autentificare (Authenticate):**
     + Endpoint pentru autentificarea unui utilizator în sistem. Generează token-uri JWT pentru gestionarea sesiunilor și accesul securizat la resursele protejate.
   * **Citire (Read):**
     + Endpoint pentru obținerea detaliilor despre un utilizator specific în funcție de ID-ul său unic. Include informații despre rolul și istoricul utilizatorului.
   * **Actualizare (Update):**
     + Endpoint pentru modificarea detaliilor unui utilizator existent în funcție de ID-ul său unic. Permite actualizarea datelor de profil și a altor informații relevante.
   * **Ștergere (Delete):**
     + Endpoint pentru ștergerea unui utilizator din sistem în funcție de ID-ul său unic. Asigură gestionarea relațiilor asociate, cum ar fi recenziile și accesul la resurse.

#### **Arhitectura Aplicatiei**

Aplicatia urmeaza o arhitectura tipica MVC (Model-View-Controller), cu separarea clara a responsabilităților. Aceasta include straturi bine definite pentru gestionarea cererilor HTTP (controller-ele), logica de afaceri (serviciile) și interacțiunea cu baza de date (repozitoriile). Fiecare componenta joaca un rol esential în functionarea eficienta si modulara a aplicatiei.

**Componentele Aplicatiei**

1. **Controllerele**Controller-ele reprezinta punctul de intrare al cererilor API. Aplicatia include trei controller-e principale: BookController, ShelfController si UserController. Acestea gestioneaza functionalitati precum adaugarea, stergerea sau actualizarea resurselor, fiind responsabile pentru trimiterea datelor catre serviciile backend. De exemplu, BookController permite operatii precum obtinerea listei de carti, adaugarea unei carti noi sau stergerea unei carti existente.
2. **Serviciile**Serviciile constituie inima aplicatiei, implementand logica de afaceri. Clase precum BookService, ShelfService si UserService gestioneaza procesarea datelor inainte de a le transmite catre repozitories sau de a le returna controller-elor. Aceste servicii asigura validarea si integritatea datelor, contribuind la separarea preocuparilor intre straturi.
3. **Repozitories**Repozitories-urile, precum BookRepository si UserRepository, sunt interfetele de acces la baza de date, utilizand Spring Data JPA. Acestea simplifica operatiunile CRUD (Create, Read, Update, Delete) si permit cautari avansate fara necesitatea scrierii manuale a interogărilor SQL.
4. **Modelele**Modelele de date, cum ar fi Book si User, definesc structura datelor din aplicatie. Ele reprezinta obiectele de transfer utilizate intre controller-e, servicii si repoziții, facilitand manipularea clara si organizata a informatiilor.
5. **Gestionarea exceptiilor**Aplicatia include clase pentru gestionarea erorilor, cum ar fi ResourceNotFoundException si ValidationException. Acestea ofera raspunsuri personalizate in cazul unor situatii precum resurse lipsa sau date invalide, imbunatatind experienta utilizatorului.

**Securitate si Configurare**Aplicatia implementeaza securitatea prin utilizarea JWT (JSON Web Token), care permite autentificarea sigura a utilizatorilor. Configurarile pentru CORS asigura compatibilitatea cererilor de la diferite domenii, in timp ce fisierul application.properties gestioneaza setarile pentru baza de date, portul serverului si alti parametrii esentiali.

**Testarea Functionalitatilor**Aplicatia include o suita bine definita de teste, localizata in directorul src/test. Clase precum BookControllerTest sau UserControllerTest simuleaza cereri HTTP pentru a verifica raspunsurile controller-elor. Aceste teste asigura functionarea corecta a componentelor critice, utilizand framework-uri precum JUnit si Mockito pentru crearea mock-urilor si validarea comportamentului aplicatiei.

**Ce test verifica fiecare metoda?**

1. **BackendApplicationTests** – Testeaza initializarea aplicatiei.
   * **Metoda contextLoads:  
      Scop:** Testeaza daca contextul Spring Boot se incarca fara probleme. Este un test de integrare esential care verifica daca configurarile aplicatiei sunt corecte si ca aplicatia poate fi initializata.  
     **Rezultate asteptate:**

**Daca testul trece:** Aplicatia este configurata corect si poate porni fara erori.

**Daca testul esueaza:** Exista o problema majora in configurare (ex. un bean lipsa, configuratie gresita a bazei de date).

1. **BookControllerTest** - Testeaza functionalitatile controller-ului cartilor.
   * **Metoda testGetBookById\_ThrowsException\_WhenNotFound:  
      Scop:** Verifica raspunsul controller-ului atunci cand se incearca accesarea unei carti care nu exista.  
      **Rezultat asteptat:** Testul confirma ca eroarea este tratata corect.
   * **Metoda testGetBookById\_ReturnsBook\_WhenFound:  
      Scop:** Verifica raspunsul controller-ului pentru o carte existenta.  
      **Rezultat asteptat:** Raspuns corect pentru cartea solicitata.
   * **Metoda testCreateBook\_ThrowsException\_WhenInvalidBook:  
      Scop:** Testeaza comportamentul controller-ului cand se incearca crearea unei carti invalide (ex. fara titlu).  
      **Rezultat asteptat:** Erorile de validare sunt gestionate corect.
   * **Metoda testCreateBook\_Success:  
      Scop:** Verifica succesul crearii unei carti valide.  
      **Rezultat asteptat:** Cartea este creata cu succes.
   * **Metoda testUpdateBook\_ThrowsException\_WhenBookNotFound:  
      Scop:** Testeaza actualizarea unei carti care nu exista.  
      **Rezultat asteptat:** Erorile sunt gestionate corect.
   * **Metoda testUpdateBook\_Success:  
      Scop:** Verifica succesul actualizarii unei carti existente.  
      **Rezultat asteptat:** Cartea este actualizata fara probleme.
   * **Metoda testDeleteBook\_ThrowsException\_WhenBookNotFound:  
      Scop:** Verifica stergerea unei carti inexistente.  
      **Rezultat asteptat:** Mesaj de eroare corect (Raspunsul trebuie sa fie 404 Not Found si un mesaj descriptiv).
   * **Metoda testDeleteBook\_Success:  
      Scop:** Testeaza succesul stergerii unei carti existente.  
      **Rezultat asteptat:** Cartea este stearsa cu succes.

**Rezultatele Testelor:**Fiecare test confirma ca BookController:

* + Trateaza corect cazurile de succes si de eroare.
  + Returneaza status-urile HTTP si mesajele de raspuns corecte.
  + Este capabil sa gestioneze interacțiunile cu serviciul backend in mod izolat.  
    Acest tip de teste ofera incredere ca BookController functioneaza conform asteptarilor in conditii normale si de eroare.

1. **ShelfControllerTest** - Testeaza controller-ul pentru rafturi.  
   * **Metoda testGetAllUserShelves:  
      Scop:** Testeaza daca se pot obtine toate rafturile utilizatorilor.  
      **Rezultat asteptat:** Endpoint-ul functioneaza corect si returneaza datele corespunzatoare.
   * **Metoda testGetUserShelfById\_ThrowsException\_WhenNotFound:  
      Scop:** Testeaza comportamentul cand un raft inexistent este solicitat.  
      **Rezultat asteptat:** Erorile sunt gestionate corect.
   * **Metoda testCreateUserShelf\_ThrowsException\_WhenInvalidShelf:  
      Scop:** Testeaza crearea unui raft invalid (ex. nume gol sau capacitate negativa).  
      **Rezultat asteptat:** Validarea intrarilor functioneaza corect.
   * **Metoda testCreateUserShelf\_Success:  
      Scop:** Testeaza succesul crearii unui raft valid.  
      **Rezultat asteptat:** Crearea resurselor este gestionata corect.
   * **Metoda testUpdateUserShelf\_ThrowsException\_WhenNotFound:  
      Scop:** Verifica actualizarea unui raft inexistent.  
      **Rezultat asteptat:** Erorile sunt gestionate corect.
   * **Metoda testUpdateUserShelf\_Success:  
      Scop:** Testeaza actualizarea unui raft existent.  
      **Rezultat asteptat:** Actualizarea functioneaza fara probleme.
   * **Metoda testDeleteUserShelf\_ThrowsException\_WhenNotFound:  
      Scop:** Testeaza stergerea unui raft inexistent.  
      **Rezultat asteptat:** Mesaj de eroare corect.
   * **Metoda testDeleteUserShelf\_Success:  
      Scop:** Testeaza stergerea unui raft existent.  
      **Rezultat asteptat:** Stergerea resurselor este gestionata corect.

**Rezultatele Testelor:**Fiecare test confirma ca ShelfController:

* + Trateaza corect cererile valide si erorile.
  + Returneaza status-urile HTTP corecte si mesaje descriptive.
  + Este capabil sa comunice eficient cu serviciul backend.  
    Aceste teste sunt esentiale pentru asigurarea integritatii functionalitatilor legate de gestionarea rafturilor in aplicatie.

1. **UserControllerTest** - Testeaza controller-ul pentru utilizatori.  
   * **Metoda testRegisterUser\_Success:  
      Scop:** Testeaza procesul de inregistrare al unui utilizator valid.  
      **Rezultat asteptat:** Raspuns corect pentru o inregistrare reusita.
   * **Metoda testRegisterUser\_Failure\_BadRequest:  
      Scop:** Testeaza procesul de inregistrare al unui utilizator cu date invalide (in acest caz, nume de utilizator gol).  
      **Rezultat asteptat:** Validarea datelor intra corect in vigoare si utilizatorul primeste un mesaj de eroare corespunzator.
   * **Metoda testLoginUser\_Success:  
      Scop:** Testeaza login-ul unui utilizator existent.  
      **Rezultat asteptat:** Login-ul este gestionat corect si utilizatorul primeste un token valid.
   * **Metoda testLoginUser\_Failure\_UserNotFound:  
      Scop:** Testeaza login-ul unui utilizator care nu exista in baza de date.  
      **Rezultat asteptat:** Utilizatorul primeste un mesaj de eroare care indica faptul ca nu a fost gasit.

**Rezultatele Testelor:**Fiecare metoda din clasa de test asteapta sa verifice diferite scenarii posibile pentru utilizatori, precum inregistrarea, login-ul si gestionarea erorilor. Rezultatele asteptate sunt:

**Inregistrare cu succes:** Utilizatorii care furnizeaza date valide sunt inregistrati corect si primesc un raspuns de succes cu un mesaj corespunzator.

**Eroare la inregistrare (BadRequest):** Utilizatorii care furnizeaza date invalide (de exemplu, nume de utilizator gol) primesc un raspuns de eroare cu un mesaj de validare.

**Login cu succes:** Utilizatorii care furnizeaza date corecte primesc un token JWT.

**Eroare la login (UserNotFound):** Utilizatorii care nu exista in sistem primesc un mesaj de eroare corespunzator, indicand ca utilizatorul nu a fost gasit.

**Plan de Testare**

Acest plan de testare este conceput pentru a asigura calitatea proiectului Backend realizat folosind framework-ul Spring Boot. Scopul este de a identifica si remedia defectele din modulele proiectului.

### **1. Obiective de Testare**

* Verificarea functionalitatii API-urilor.
* Testarea corectitudinii interactiunilor cu baza de date.
* Asigurarea conformitatii cu cerintele de securitate (ex. autentificare JWT).
* Validarea comportamentului aplicatiei in conditii normale si de eroare.

### **2. Domeniul de Aplicare**

**Module care vor fi testate:**

* **Controlere:**
  + BookController
  + ShelfController
  + UserController
* **Servicii:**
  + BookService
  + ShelfService
  + UserService
* **Repository-uri:**
  + BookRepository
  + ShelfRepository
  + UserRepository
* **Configuratii:**
  + CorsConfig
  + JWTUtil

### **3. Strategia de Testare**

**Tipuri de Teste:**

* **Teste Unitare**
  + Testarea functionalitatii fiecarei metode din clasele serviciilor si repository-urilor.
  + **Exemple:** Metoda findById din BookRepository trebuie sa returneze o entitate existenta sau sa declanseze o exceptie.
* **Teste Functionale**
  + Verificarea raspunsurilor API la cereri valide.
  + **Exemple:** Endpoint-ul GET/books/{id} trebuie sa returneze detaliile cartii corespunzatoare.
* **Teste de Integrare**
  + Asigurarea ca modulele functioneaza bine impreuna (ex. controlere si servicii).
  + **Exemple:** BookController utilizeaza corect BookService pentru a crea o carte noua.
* **Teste de Securitate**
  + Validarea autentificarii si autorizarii folosind JWT.
  + **Exemple:** Endpoint-ul POST/books ar trebui sa fie accesibil doar utilizatorilor autentificati.
* **Teste de Performanta**
  + Masurarea timpilor de raspuns pentru endpoint-uri cheie.
* **Testare manuala**
  + Evaluarea interactiunilor utilizatorului cu API-ul si asigurarea ca toate functionalitatile functioneaza conform specificatiilor.
  + **Exemple:** Endpoint-ul POST/books ar trebui sa valideze comportamentul aplicatiei cu baza de date si in conditii de eroare.

### **4. Criterii de Acceptare**

* Toate testele unitare trebuie sa treaca cu o acoperire de cod de cel putin 80%.
* API-urile trebuie sa returneze raspunsuri corecte in conformitate cu documentatia.
* Aplicatia trebuie sa functioneze corect in caz de date invalide sau absenta unor resurse.
* Sa fie respectate regulile de acces stabilite prin mecanismele de securitate.
* Pentru testarea manuala a bazei de date cu Postman includ verificarea faptului ca API-urile returneaza raspunsuri corecte si asteptate pentru diverse tipuri de cereri, cum ar fi GET, POST, PUT si DELETE.

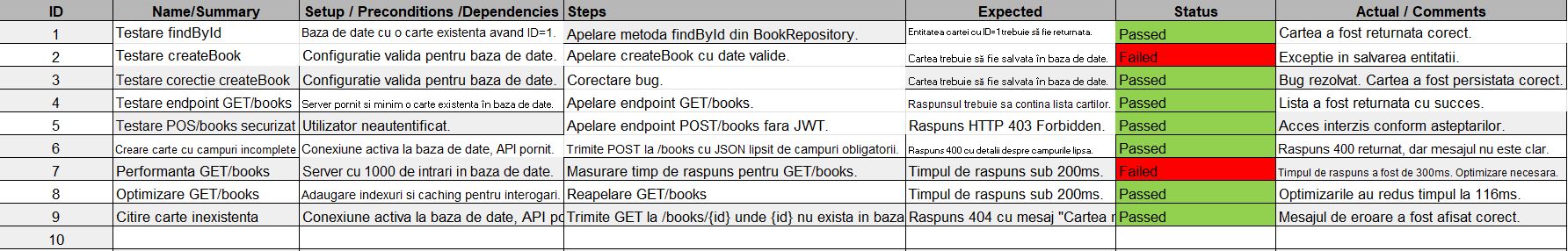
### **5. Planificare si Resurse**

* **Testeri:** Ciocan Felicia-Ilinca, Cirlig Geanina-Bianca
* **Timp Estimat:**
  + Teste Unitare: 3 zile
  + Teste Functionale: 3 zile
  + Teste de Integrare: 3 zile
  + Teste de Securitate: 3 zile

### **6. Livrabile**

* Rapoarte de testare pentru fiecare tip de test efectuat.
* Lista problemelor identificate si solutionate.
* Acest plan de testare detaliaza toate activitatile necesare pentru a asigura calitatea proiectului. Dupa finalizarea testelor, aplicatia ar trebui sa fie gata pentru implementare.

**Raport de testare**



TEST CASE-URI

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case  ID | Descriere | Permise / Necesitati | Pasi | Rezultate asteptate | Rezultate obtinute | Status | Creat/ Executat | Data |
| TC1 | Înregistrarea unui utilizator nou.  Testează endpoint-ul POST /register pentru a verifica dacă un utilizator este înregistrat cu succes. | Aplicația trebuie să fie funcțională. Datele utilizatorului (e.g., email unic, parolă validă) trebuie să fie valide. | 1: Trimite o cerere POST către /register cu payload JSON valid (e.g., {"email": "test@example.com", "password": "Test123!"}). | Serverul răspunde cu status HTTP 201 (Created) și mesajul User registered successfully. | Statusul HTTP nu este 201 sau apare o eroare de validare. | Failed | User:  Felicia  Passw:  123 | 9.12 |
| TC2 | Înregistrare utilizator cu date invalide.  Testează metoda registerUser pentru a verifica dacă sunt gestionate corect datele invalide. | Utilizatorul trimis are date invalide (de exemplu, username gol, email gol). | 1: Creează un obiect User invalid (e.g., User("", "test@example.com", "Password123")).  2: Apelează metoda registerUser(user). | Aruncă o excepție de tip ValidationException cu mesajul Username este obligatoriu. | Metoda aruncă o excepție ValidationException corespunzătoare. | Passed | User:  Bianca  Passw:  abc | 10.12 |
| TC3 | Autentificare cu utilizator inexistent.  Testează scenariul în care un utilizator încearcă să se autentifice cu un email inexistent. | Utilizatorul nu este înregistrat în baza de date. | Trimite o cerere POST către /login cu payload JSON pentru un utilizator inexistent (e.g., {"email": "unknown@example.com", "password": "WrongPass"}). | Serverul răspunde cu status HTTP 404 (Not Found) și mesajul User not found. | Serverul refuză autentificarea pentru utilizatorul inexistent. | Passed | User:  Felicia  Passw:  123 | 10.12 |
| TC4 | Obținerea tuturor cărților disponibile.  Testează endpoint-ul GET /api/books pentru a verifica dacă returnează toate cărțile disponibile. | Aplicația trebuie să fie funcțională. În baza de date trebuie să existe cel puțin două cărți adăugate. | 1: Trimite o cerere HTTP GET la ruta /api/books.  : Verifică răspunsul. | Răspunsul este o listă JSON care conține toate cărțile existente în baza de date (e.g., două sau mai multe obiecte). | odul de status este altul decât 200 OK sau lista este goală/invalidă. | Failed | User:  Bianca  Passw:  abc | 16.12 |
| TC5 | Actualizarea unei cărți neexistente. Testează metoda validateAndUpdateBook pentru a verifica dacă se aruncă o excepție atunci când titlul cărții nu este găsit. | Nu există o carte cu titlul dat în baza de date. | 1: Creează un obiect Book cu detalii valide.  2: Apelează metoda validateAndUpdateBook("nonExistentTitle", updatedBookDetails). | Aruncă o excepție ResourceNotFoundException. | Se aruncă o excepție ResourceNotFoundException. | Passed | User:  Felicia  Passw:  123 | 17.12 |
| TC6 | Crearea unui raft valid.Testează funcționalitatea de creare a unui raft valid (UserShelf) cu detalii corecte. Se verifică dacă raftul este salvat în baza de date. | Aplicația trebuie să fie funcțională. Baza de date conectată și disponibilă. ShelfRepository configurat și injectat corect. | 1: Deschide aplicația (sau API-ul).  Rezultat așteptat: Aplicația este activă și funcțională.  2: Creează un obiect UserShelf cu: shelfName: „Raft de test”.  3: Apelează metoda createUserShelf() cu obiectul creat.  : Verifică în baza de date dacă raftul a fost salvat (folosind o interogare directă). | Raftul este salvat cu succes în baza de date, iar metoda returnează obiectul creat. | Raftul a fost salvat cu succes în baza de date. | Passed | User:  Bianca  Passw:  abc | 17.12 |

Bug report

|  |
| --- |
| Titlu Bug:  Endpoint-ul createBook nu returnează un mesaj de eroare în cazul în care crearea unei cărți eșuează din cauza unei erori interne de server (ex: eroare de bază de date). |
| ID Bug: BOOK-200 |
| Severitate: Mediu |
| Prioritate: Mare |
| Raportat de: Damean Andrei-Stefan |
| Data Raportării: 10.12.2024 |
| Pași pentru a Reproduce Bug-ul: |
| 1.Realizează un apel POST la endpoint-ul /api/books pentru a crea o carte cu un obiect valid de tip Book.  2.Simulează o eroare la nivel de server, cum ar fi o eroare de bază de date sau o problemă internă, care împiedică salvarea cărții.  3.În prezent, endpoint-ul răspunde cu un cod HTTP 200 (OK), dar nu există un mesaj de eroare sau o notificare clară despre eșecul operațiunii. |
| Rezultatul Așteptat:  În cazul unei erori interne (de exemplu, eroare de bază de date), ar trebui să fie returnat un cod HTTP 500 (Internal Server Error) cu un mesaj clar de eroare în corpul răspunsului, indicând problema. |
| Rezultatul Actual:  Atunci când apare o eroare internă în timpul creării unei cărți (ex: eroare de conectare la baza de date), endpoint-ul returnează un răspuns cu HTTP 200 OK, fără a semnala corect că ceva a mers prost. |
| Cauza Principală:  Endpoint-ul createBook nu gestionează erorile interne ale serverului. Nu există o gestionare a excepțiilor care să captureze erorile neașteptate care pot apărea în timpul procesului de creare a cărții. |
| Corectare:  Se recomandă adăugarea unui bloc de captură pentru excepțiile neașteptate (de exemplu, Exception) și returnarea unui răspuns corespunzător cu codul HTTP 500 și un mesaj detaliat. |
| Exemplu de corectare:  @PostMapping  public ResponseEntity<?> createBook(@RequestBody Book book) {  try {  boolean result = service.validateAndCreateBook(book);  return ResponseEntity.ok().build();  } catch (IllegalArgumentException e) {  return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("{\"message\": \"" + e.getMessage() + "\"}");  } catch (Exception e) {  // Tratarea erorilor interne  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("{\"message\": \"A apărut o eroare internă. Vă rugăm să încercați mai târziu.\"}");  }  } |

|  |
| --- |
| Titlu Bug:  Eroare de validare a parolei la autentificare atunci când utilizatorul are o parolă null sau goală |
| ID Bug: USER-201 |
| Severitate: Mediu |
| Prioritate: Mare |
| Raportat de: Ciocan Felicia-Ilinca |
| Data Raportării: 13.12.2024 |
| Pași pentru a Reproduce Bug-ul:  Crează un utilizator cu o parolă null sau goală (de exemplu, un User cu password = null sau password = "").  Încearcă să te autentifici cu utilizatorul respectiv, folosind un nume de utilizator valid și o parolă goală sau null.  În loc să fie validată corect eroarea de parolă, aplicația va returna un răspuns care poate fi confuz, indicând că acreditările sunt invalide. |
| Rezultatul Așteptat:  Atunci când utilizatorul are o parolă null sau goală, ar trebui să fie aruncată o excepție personalizată (de exemplu, ValidationException) care să semnaleze clar că parola nu poate fi goală sau null.  Mesajul de eroare ar trebui să fie clar și să semnaleze direct problema cu parola. |
| Rezultatul Actual:  În prezent, dacă utilizatorul are o parolă null sau goală, codul din metoda validateLoginCredentials nu aruncă o excepție corespunzătoare, iar mesajul de eroare va fi confuz și se va referi doar la "Acreditări invalide" (mesaj standard pentru orice eroare de login). |
| Cauza Principală:  Validarea pentru parolă nu este efectuată înainte de a verifica acreditările în metoda validateLoginCredentials. Validarea ar trebui să verifice dacă parola este validă (nu null sau goală) înainte de a căuta utilizatorul în baza de date. |
| Corectare:  Adăugarea unei verificări pentru parola null sau goală înainte de a căuta utilizatorul în baza de date. |
| Exemplu de corectare: |
| private void validateLoginCredentials(User user) {  if (user.getPassword() == null || user.getPassword().isEmpty()) {  throw new ValidationException("Parola este obligatorie");  }    User foundUser = userRepository.findByUsername(user.getUsername());  if (foundUser == null || !Objects.equals(foundUser.getPassword(), user.getPassword())) {  throw new ResourceNotFoundException("Acreditări invalide");  }  } |

|  |
| --- |
| Titlu Bug:  Eroare în procesul de login când nu se trimite un body valid |
| ID Bug: USER-202 |
| Severitate: Mediu |
| Prioritate: Mare |
| Raportat de: Cirlig Geanina-Bianca |
| Data Raportării: 16.12.2024 |
| Pași pentru a Reproduce Bug-ul:  Trimite o cerere POST către endpoint-ul /login fără un body valid sau cu un body gol.  Aplicația va răspunde cu un răspuns de succes (HTTP 200) și va încerca să proceseze cererea fără a valida corect inputul. |
| Rezultatul Așteptat:  Atunci când cererea POST pentru /login nu conține un body valid (de exemplu, lipsă sau cu date invalide), ar trebui să fie returnat un răspuns cu HTTP 400 (BAD REQUEST) și un mesaj clar care să semnaleze că datele sunt invalide. |
| Rezultatul Actual:  În prezent, dacă se trimite o cerere invalidă sau fără date valide (de exemplu, un body gol sau lipsit de câmpurile necesare), aplicația va încerca să proceseze cererea și va arunca o excepție de tip NullPointerException sau IllegalArgumentException la nivelul serviciului, fără a returna un mesaj de eroare corespunzător la nivelul controller-ului. |
| Cauza Principală:  Controller-ul nu validează corect cererea înainte de a o trimite către serviciu. În cazul în care body-ul cererii este gol sau conține date invalide, ar trebui să fie validat înainte de a apela serviciul de login. |
| Corectare:  Adăugarea unei verificări pentru a valida dacă body-ul cererii conține date valide înainte de a apela serviciul de login. În plus, ar trebui să fie returnat un răspuns cu status 400 și un mesaj de eroare în cazul în care body-ul cererii este invalid. |
| Exemplu de corectare: |
| @PostMapping("/login")  public ResponseEntity<?> loginUser(@RequestBody User user) {  if (user == null || user.getUsername() == null || user.getPassword() == null) {  return ResponseEntity.status(HttpStatus.BAD\_REQUEST)  .body(Collections.singletonMap("error", "Username and password are required"));  }  try {  String token = userService.loginUser(user);  return ResponseEntity.ok()  .body(Collections.singletonMap("token", token));  } catch (ResourceNotFoundException e) {  return ResponseEntity.status(HttpStatus.NOT\_FOUND)  .body(Collections.singletonMap("error", e.getMessage()));  }  } |

|  |
| --- |
| Titlu Bug:  NullPointerException în ShelfService.getNoOfBooks |
| ID Bug: BOOK-203 |
| Severitate: Mare |
| Prioritate: Mare |
| Raportat de: Damean Andrei-Stefan |
| Data Raportării: 17.12.2024 |
| Pași pentru a Reproduce Bug-ul:  Creați un obiect UserShelf în baza de date cu valoarea null pentru câmpul books. Apelați metoda ShelfService.getNoOfBooks utilizând shelfId corespunzător. |
| Rezultatul Asteptat:  Metoda ar trebui să returneze 0 sau să trateze cazul în care books este null fără a arunca o excepție. |
| Rezultatul Actual:  Metoda aruncă o excepție NullPointerException. |
| Cauza Principala:  Metoda getNoOfBooks generează o excepție NullPointerException atunci când câmpul books din obiectul UserShelf este null. Acest lucru se poate întâmpla dacă un raft din baza de date nu are câmpul books inițializat. |
| Corectare:  Modificați metoda getNoOfBooks pentru a trata explicit cazul în care câmpul books este null. |
| Exemplu de corectare: |
| public int getNoOfBooks(String shelfId)  {  UserShelf userShelf = getUserShelfById(shelfId);  raftul List<Book> books = userShelf.getBooks();  cărțile if (books == null)  {  return 0;  lista de cărți este null  }  return books.size(); |

|  |
| --- |
| Titlu Bug:  Metoda getBookById aruncă excepția IllegalArgumentException atunci când se primește un ID valid, dar în format invalid (ex: cu caractere speciale). |
| ID Bug: BOOK-204 |
| Severitate: Mediu |
| Prioritate: Mare |
| Raportat de: Ciocan Felicia-Ilinca |
| Data Raportării: 19.12.2024 |
| Pași pentru a Reproduce Bug-ul:  Apelează metoda getBookById cu un ID valid, dar care conține caractere speciale (de exemplu: "abc@123"). Metoda aruncă o excepție IllegalArgumentException în loc de a returna cartea corespunzătoare sau de a efectua o validare suplimentară a formatului ID-ului. |
| Rezultatul Așteptat:  Metoda ar trebui să accepte ID-uri valide (inclusiv cu caractere speciale) și să proceseze corect cererea, returnând cartea corespunzătoare dacă există. |
| Rezultatul Actual:  În loc să trateze corect ID-urile cu caractere speciale, metoda getBookById aruncă o excepție IllegalArgumentException, chiar și pentru ID-uri care sunt valide din punct de vedere al formatului, dar conțin caractere speciale. |
| Cauza Principală:  Validarea ID-ului din metoda getBookById nu permite caractere speciale, deși ID-urile valide pot conține astfel de caractere. Nu există o verificare de format specific pentru ID-uri. |
| Corectare:  Se poate adăuga o validare suplimentară pentru a permite caractere speciale în ID-uri, fără a arunca o excepție inutilă. De asemenea, se poate îmbunătăți gestionarea excepțiilor. |
| Exemplu de corectare: |
| if (id == null || id.trim().isEmpty()) {  throw new IllegalArgumentException("ID-ul trebuie să fie un șir de caractere valid.");  }  if (!id.matches("^[a-zA-Z0-9@.\_-]+$")) {  throw new IllegalArgumentException("ID-ul conține caractere invalide.");  } |