UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
CENTRUL UNIVERSITAR NORD DIN BAIA MARE

Facultatea de Inginerie
Departamentul de Inginerie Electrică, Electronică și Calculatoare

APLICAȚIE DE TESTARE ONLINE

Proiect la disciplina:
Proiectarea Aplicațiilor Web

Autor:
Titea Dan Ruben

Program de studii:
Calculatoare

Anul: IV

Baia Mare
2026

Cuprins

1 Introducere

1.1 Context general

Aplicația web de testare online este concepută pentru a oferi o platformă interactivă și accesibilă, care permite utilizatorilor să creeze, editeze și participe la teste grilă din diverse domenii. Scopul principal este de a facilita procesul de creare și gestionare a testelor pentru utilizatorii autentificați, precum și participarea la teste pentru utilizatorii guest, într-un mod eficient și securizat. Funcționalitățile de căutare și sortare vor spori accesibilitatea și organizarea conținutului. Aplicația integrează tehnologii moderne pentru a asigura o experiență fluidă și performanțe ridicate.

1.2 Obiective

Scopul aplicației este de a oferi o platformă scalabilă și intuitivă pentru crearea și gestionarea testelor online. Obiectivele specifice includ:

- Permiterea utilizatorilor autentificați să creeze, editeze și șteargă teste proprii.
- Permiterea utilizatorilor autentificați și guest să participe la teste create de alți utilizatori autentificați.
- Implementarea unui sistem de căutare a testelor după titlu și sortare după domeniu și data creării.
- Asigurarea unei comunicări în timp real prin WebSockets pentru afișarea utilizatorilor care participă la un test.
- Implementarea unui sistem de securitate robust pentru autentificare și autorizare, utilizând Web Session sau JWT.

1.3 Lista MoSCoW

Metoda MoSCoW este utilizată pentru a prioritiza cerințele aplicației, clasificându-le în patru categorii: Must-have, Should-have, Could-have și Won't-have.

Tabela 1: Lista MoSCoW

Must-have	**Should-have**	**Could-have**	**Won't-have**
Creare, editare, ștergere teste de către utilizatori autentificați	Căutare teste după titlu	Statistici detaliate despre performanța utilizatorilor	Generare automată de întrebări
Participare la teste pentru utilizatori autentificați și guest	Sortare teste după domeniu și data creării	Sistem de clasament pentru utilizatori	Integrare cu platforme externe de învățare
Autentificare și autorizare securizată (JWT)	Afișare în timp real a utilizatorilor care participă (WebSockets)	Notificări prin email pentru finalizarea testelor	Suport pentru teste multimedia (imagini, video)
	Gestionarea profilului utilizatorului	Personalizare interfață utilizator	

1.4 Cazuri de utilizare

Aplicația deservește două tipuri de utilizatori: utilizatori autentificați și utilizatori guest. Cazurile de utilizare reflectă interacțiunile principale ale acestora cu sistemul:

Utilizator autentificat:

- Creare test grilă (inclusiv întrebări și răspunsuri).
- Editare/ștergere teste proprii.
- Participare la teste create de alți utilizatori autentificați.
- Căutare teste după titlu.
- Sortare teste după domeniu sau data creării.
- Vizualizare utilizatori activi pe un test (via WebSockets).

Utilizator guest:

- Autentificare/Înregistrare în aplicație.
- Participare la teste create de utilizatori autentificați.
- Căutare teste după titlu.
- Sortare teste după domeniu sau data creării.
- Vizualizare utilizatori activi pe un test (via WebSockets).

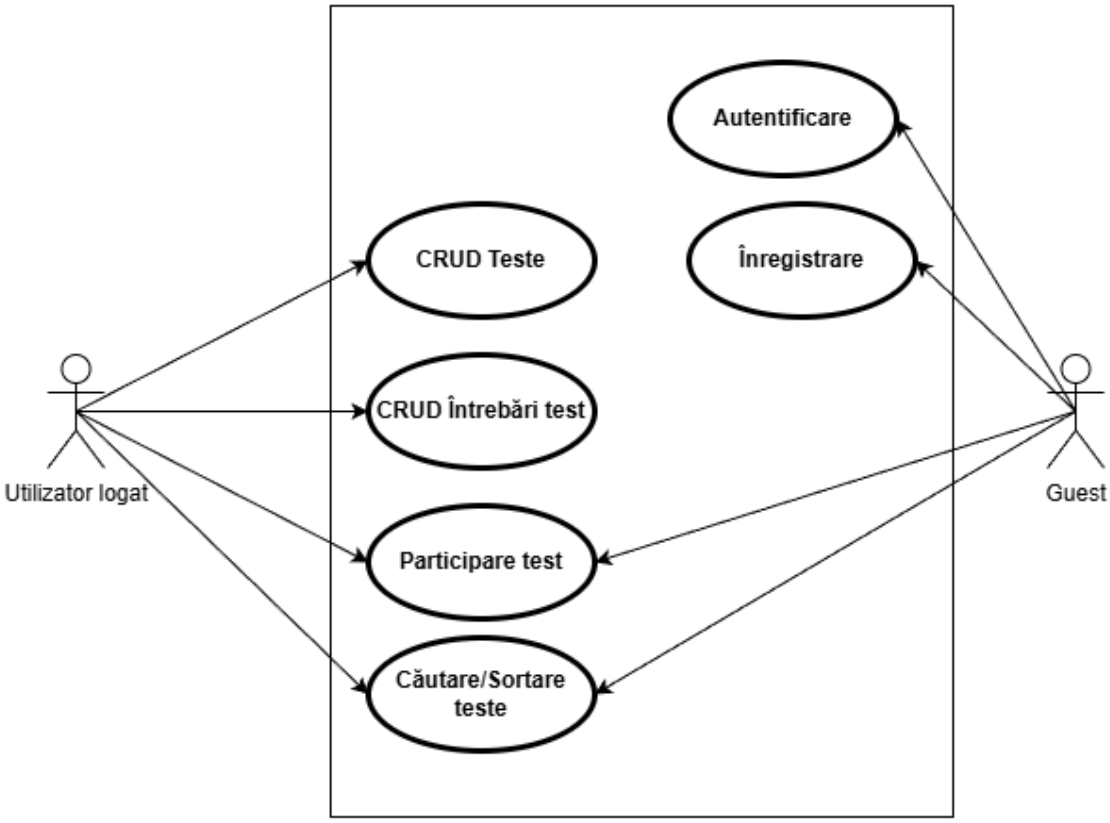

Figura 1: Diagrama cazurilor de utilizare

2 Arhitectura

Arhitectura aplicației este construită pentru a fi scalabilă, modulară și eficientă, integrând tehnologii moderne precum React pentru front-end, Spring Boot pentru back-end și MySQL pentru gestionarea datelor. WebSockets și securitatea bazată pe Web Session sau JWT completează arhitectura, oferind funcționalități în timp real și protecție robustă.

2.1 Front-End React

React este utilizat pentru interfața utilizator datorită modularității și performanței sale. Componentele React permit gestionarea eficientă a stării aplicației și oferă o experiență interactivă. Caracteristici cheie:

- Componente reutilizabile pentru afișarea testelor, formularelor de creare/editare și rezultatelor.
- Gestionarea cererilor HTTP către back-end prin biblioteca Axios.
- Integrare WebSockets pentru afișarea în timp real a utilizatorilor activi.
- Interfață responsivă pentru compatibilitate pe dispozitive mobile și desktop.

2.2 Back-End Spring Boot

Spring Boot servește drept fundament al back-end-ului, oferind un cadru robust pentru gestionarea logicii aplicației. Caracteristici cheie:

- **Spring Data JPA**: Simplifică interacțiunea cu baza de date MySQL, oferind operațiuni CRUD eficiente.
- **Dependency Injection**: Asigură o gestionare modulară a dependențelor, facilitând testarea și mentenanța.
- **REST API**: Expune endpoint-uri pentru crearea, editarea, ștergerea și căutarea testelor, precum și pentru autentificare.
- **WebSockets**: Implementate prin Spring WebSocket pentru a permite comunicarea în timp real (ex. afișarea utilizatorilor activi).
- **Spring Security**: Gestionează autentificarea și autorizarea utilizând Web Session sau JWT.

2.3 Comunicare între Front-End și Back-End

- **Axios**: Utilizat pentru cereri HTTP asincrone către API-urile REST expuse de Spring Boot. Asigură o comunicare rapidă și eficientă între front-end și back-end.
- **WebSockets**: Utilizate pentru actualizări în timp real, cum ar fi afișarea utilizatorilor care participă la un test.
- **Format date**: Datele sunt transmise în format JSON pentru compatibilitate și eficiență.

2.4 Baza de Date MySQL

MySQL este ales ca sistem de gestiune a bazelor de date datorită performanței, scalabilității și suportului pentru relații complexe. Spring Data JPA facilitează maparea obiectelor Java pe tabelele MySQL, reducând codul boilerplate.

2.5 Beneficii ale acestei arhitecturi

- **Scalabilitate**: Arhitectura permite gestionarea unui număr mare de utilizatori și teste prin utilizarea MySQL și Spring Boot.
- **Modularitate**: React și Dependency Injection în Spring Boot facilitează mentenanța și extinderea aplicației.
- **Performanță**: WebSockets și cererile asincrone optimizează comunicarea în timp real și timpul de răspuns.
- **Securitate**: Spring Security, combinat cu Web Session sau JWT, asigură protecția datelor și accesul controlat.
- **Flexibilitate**: Structura modulară permite adăugarea de noi funcționalități (ex. statistici avansate) fără modificări majore.

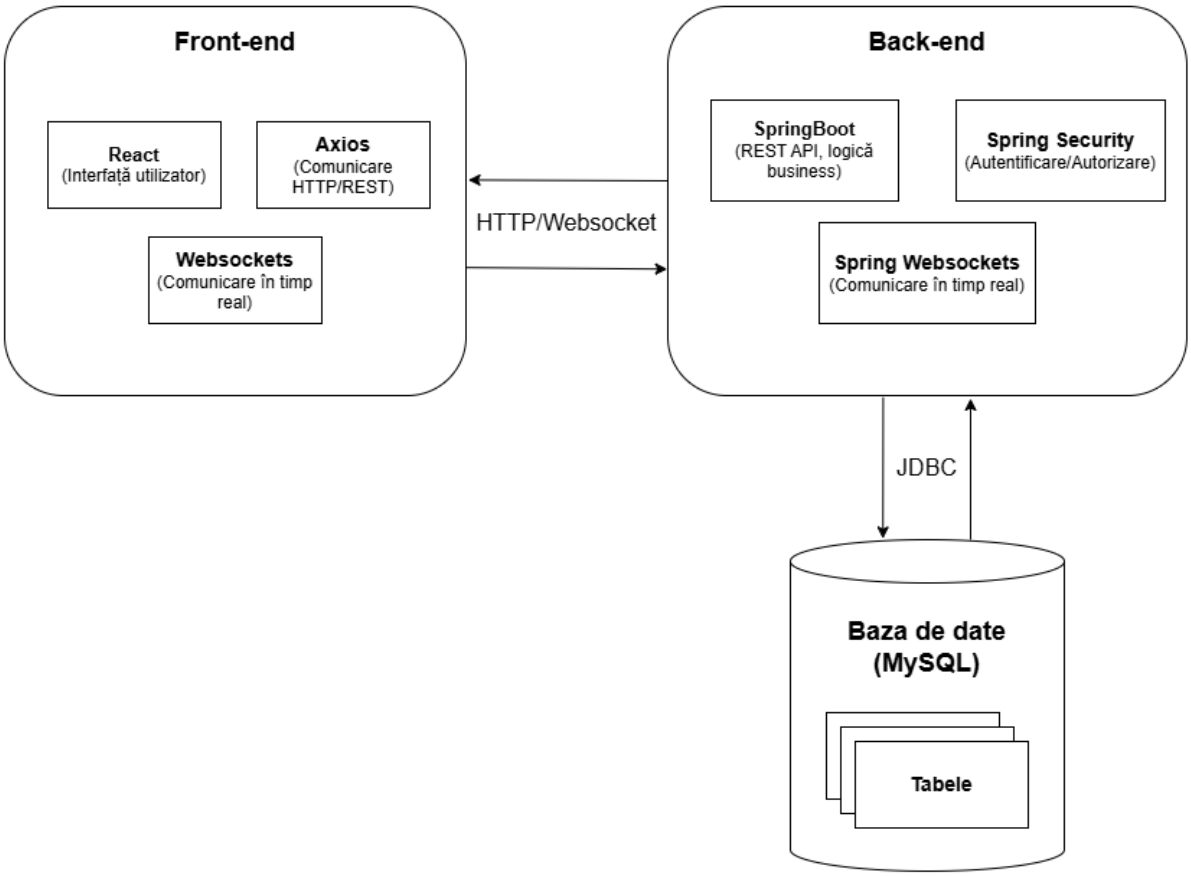

Figura 2: Arhitectura aplicației

3 Implementare Front-End

3.1 Tehnologii Utilizate

Implementarea interfeței utilizator a fost realizată utilizând ecosistemul **React** (versiunea 19.2.0), beneficiind de un proces de dezvoltare rapid și optimizat oferit de **Vite**. Pentru gestionarea dependențelor și a scripturilor de rulare s-a utilizat `npm`.

Principalele biblioteci și instrumente integrate în aplicație sunt:

- **React Router DOM (v7.9.6)**: Asigură navigarea în aplicație (routing) prin componentele `BrowserRouter`, `Routes` și `Route`, permițând crearea unei aplicații de tip Single Page Application (SPA).
- **Axios (v1.13.2)**: Utilizat pentru efectuarea cererilor HTTP asincrone către API-ul REST din back-end. A fost configurată o instanță globală (`api.js`) care utilizează interceptori pentru a atașa automat token-ul JWT în header-ul *Authorization* pentru cererile autentificate.
- **SockJS și StompJS**: Biblioteci utilizate pentru implementarea comunicării în timp real prin protocolul WebSocket, necesare pentru funcționalitatea de afișare a utilizatorilor activi pe un test.
- **Context API**: Utilizat pentru gestionarea stării globale de autentificare prin `AuthContext.jsx` care stochează token-ul JWT și informațiile despre utilizatorul curent.

3.2 Structura și Componentele Aplicației

Aplicația este structurată modular, având punctul de intrare în fișierul `main.jsx`, care încapsulează aplicația în providerii necesari (`AuthProvider`, `BrowserRouter`). Componenta principală `App.jsx` definește rutele și utilizează o componentă de tip `Layout` pentru a menține o structură consistentă (Navbar și conținut principal).

Componentele principale sunt organizate astfel:

3.2.1 Componente de bază (Shared)

- `Navbar.jsx`: Bara de navigare care își adaptează conținutul în funcție de starea de autentificare (afișează Link-uri de Login/Register pentru vizitatori sau Profil/Logout pentru utilizatori).
- `ProtectedRoute.jsx`: O componentă de tip wrapper care restricționează accesul la rutele private (ex. Profil, Logout), redirecționând utilizatorii neautentificați către pagina de Login.

3.2.2 Modulul de Autentificare

- `Login.jsx` și `Register.jsx`: Formulare pentru autentificare și înregistrare care comunică cu `authService.js`.
- `AuthContext.jsx`: Gestionează logica de persistență a sesiunii (folosind `localStorage`) și expune metodele `login` și `logout`.

3.2.3 Modulul de Teste

- `TesteList.jsx`: Afișează lista tuturor testelor disponibile. Include funcționalități de căutare după titlu, filtrare după domeniu și sortare (cele mai noi/vechi). De asemenea, gestionează deschiderea modalului de creare/editare.
- `CreateTest.jsx`: Un formular complex (modal) care permite adăugarea titlului, selectarea domeniului și gestionarea dinamică a întrebărilor și opțiunilor de răspuns (inclusiv marcarea răspunsurilor corecte).
- `TakeTest.jsx`: Interfața de susținere a testului. Încarcă întrebările, gestionează selecția răspunsurilor utilizatorului și trimite datele către server pentru validare. Include componenta `ActiveUsers` pentru funcționalitatea real-time.
- `TestResult.jsx`: Afișează rezultatul final după trimiterea testului (scor, procentaj și statusul promovat/nepromovat).
- `ActiveUsers.jsx`: O componentă laterală care se conectează la WebSocket și afișează lista utilizatorilor care participă simultan la același test.

3.3 Fluxul de utilizare al aplicației

Interacțiunea utilizatorului cu aplicația urmează un flux logic, de la accesare până la obținerea rezultatelor:

1. **Accesarea Paginii Acasă**: Utilizatorul este întâmpinat de componenta `Home.jsx`, care oferă o scurtă descriere și un buton "Răsfoiește teste" pentru navigare rapidă către lista de teste.
2. **Vizualizarea și Căutarea Testelor**: În pagina `/teste`, componenta `TesteList` încarcă asincron testele și domeniile. Utilizatorul poate folosi bara de căutare sau filtrele pentru a găsi un test specific.
3. **Susținerea unui Test**:
   - La apăsarea butonului "Susține Testul", utilizatorul este redirecționat către ruta `/teste/:id/take`.
   - Componenta `TakeTest` inițializează conexiunea WebSocket prin `ActiveUsers`, abonându-se la topicul specific testului pentru a vedea alți participanți.
   - Utilizatorul bifează răspunsurile (checkbox-uri). Aplicația validează local dacă au fost completate toate întrebările obligatorii înainte de trimitere.
4. **Finalizarea și Rezultatul**:
   - La apăsarea butonului "Trimite Testul", răspunsurile sunt trimise către back-end prin `testeService.submitTest`.
   - Utilizatorul este redirecționat automat către pagina de rezultate (`TestResult`), unde i se afișează punctajul obținut și procentajul, fără a mai putea modifica răspunsurile.

3.4 Diferențe între Guest și Utilizator Autentificat

Aplicația adaptează interfața și funcționalitățile în funcție de prezența token-ului JWT în `AuthContext`.

Utilizator Guest (Neautentificat)

- **Navigare**: Bara de navigare afișează opțiunile "Autentificare" și "Înregistrare".
- **Participare la teste**: Poate accesa lista de teste și poate susține orice test. În componenta `TakeTest`, back-end-ul atribuie un nume de utilizator temporar (ex. *"IPqXwUp7"*), care este stocat local pentru sesiunea curentă a testului și utilizat pentru identificare în lista de utilizatori activi (WebSocket).
- **Restricții**: Nu vede butonul "+ Creează Test" în lista de teste și nu are acces la opțiunile de editare sau ștergere a testelor existente. Rutele `/profile` și `/logout` sunt protejate și inaccesibile.

Utilizator Autentificat

- **Navigare**: Bara de navigare afișează "Profil" și "Deconectare".
- **Gestionare Teste**: În pagina `TesteList`, apare butonul pentru crearea unui test nou. De asemenea, pentru testele create de el însuși (verificare pe baza `user.id`), sunt afișate butoanele "Editează" și "Șterge".
- **Identitate**: În timpul susținerii unui test și în comunicarea WebSocket, este utilizat username-ul real al contului.
- **Profil**: Are acces la pagina de profil unde își poate vizualiza detaliile contului (username, rol, ID).

3.5 Interfața Grafică

3.5.1 Interfața Guest

Utilizatorii neautentificați pot vizualiza și susține teste, dar nu pot crea conținut.

Pagina Acasă

Pagina de start oferă acces rapid către lista de teste sau către autentificare.

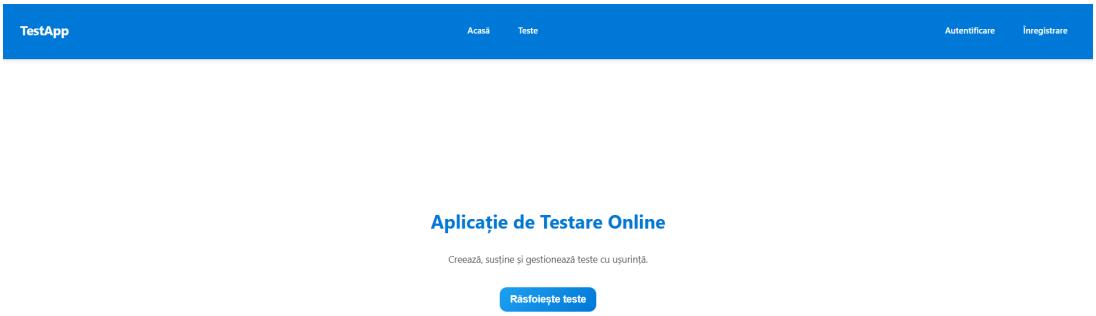

Figura 3: Pagina Home

Autentificare și Înregistrare

Formularele permit accesul în aplicație pe baza numelui de utilizator și a parolei.

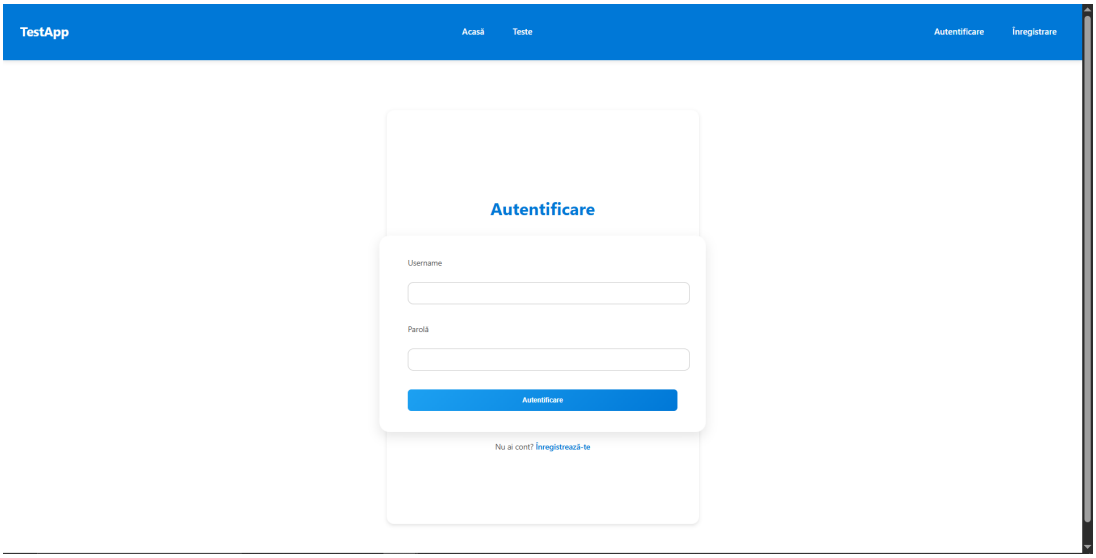

Figura 4: Pagina de Autentificare

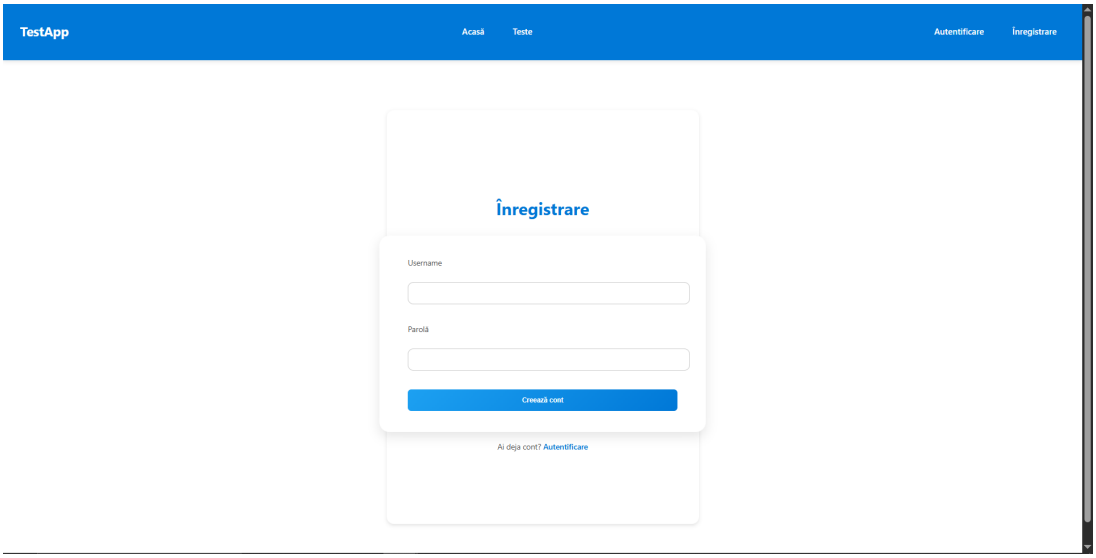

Figura 5: Pagina de Înregistrare

Lista de Teste

Afișează toate testele disponibile, incluzând opțiuni de căutare și filtrare după domeniu.

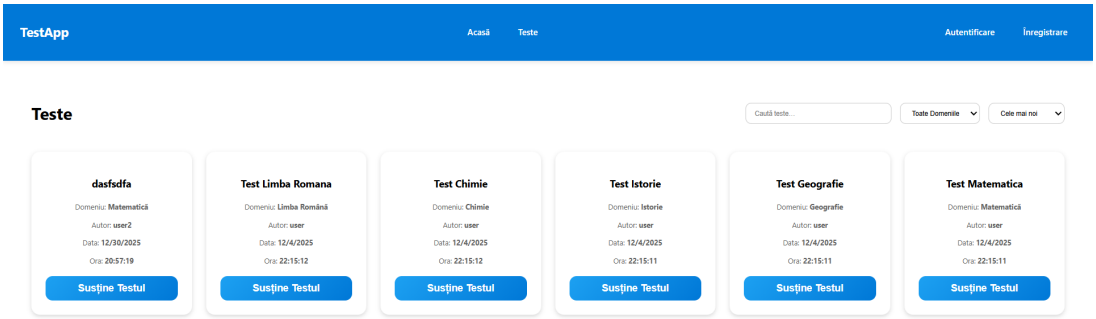

Figura 6: Lista testelor (Guest)

Susținere Test

Interfața de examinare afișează întrebările și variantele de răspuns pentru testul selectat.

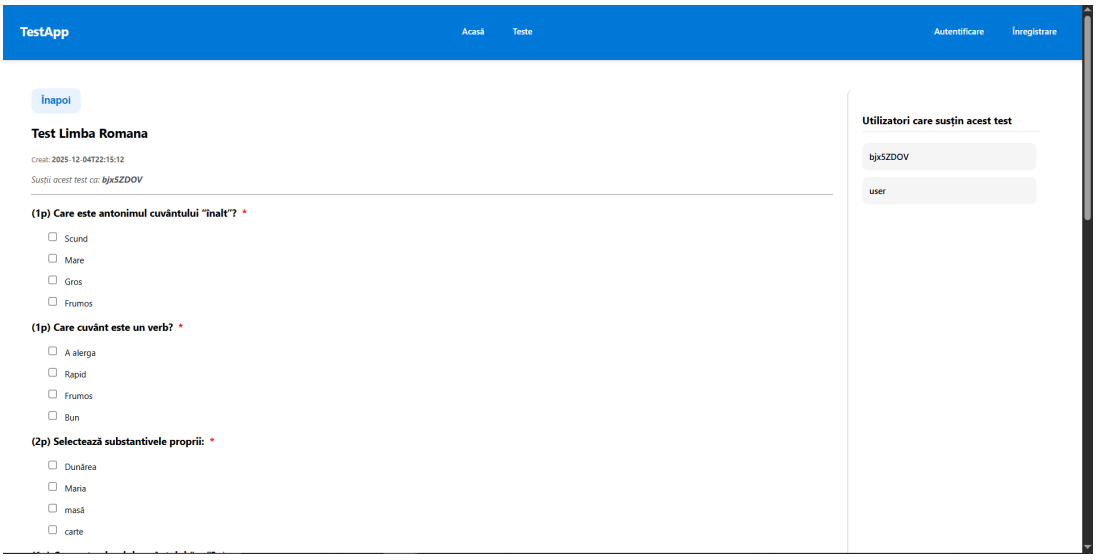

Figura 7: Pagina de susținere a testului

Rezultate

La final, utilizatorul vede scorul obținut, procentajul și un mesaj de validare.

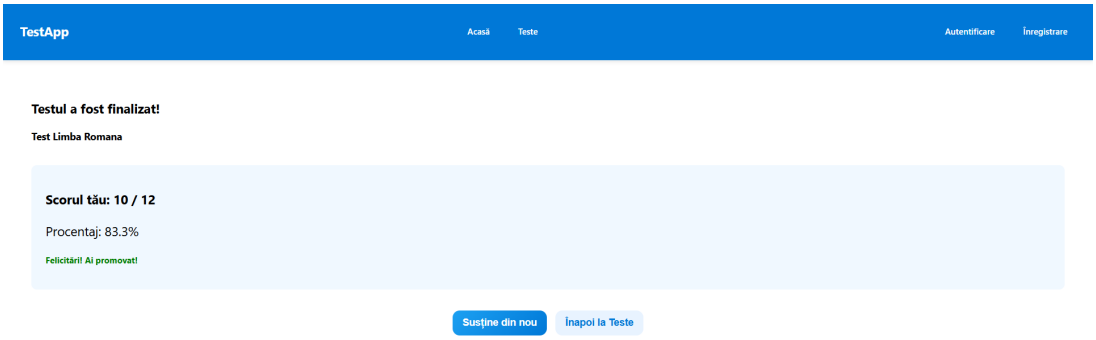

Figura 8: Pagina de rezultate

3.5.2 Interfața Utilizator Autentificat

Utilizatorii logați au acces la funcționalități suplimentare de administrare a testelor și vizualizare a profilului.

Profil

Afișează datele contului curent (username, rol și ID).

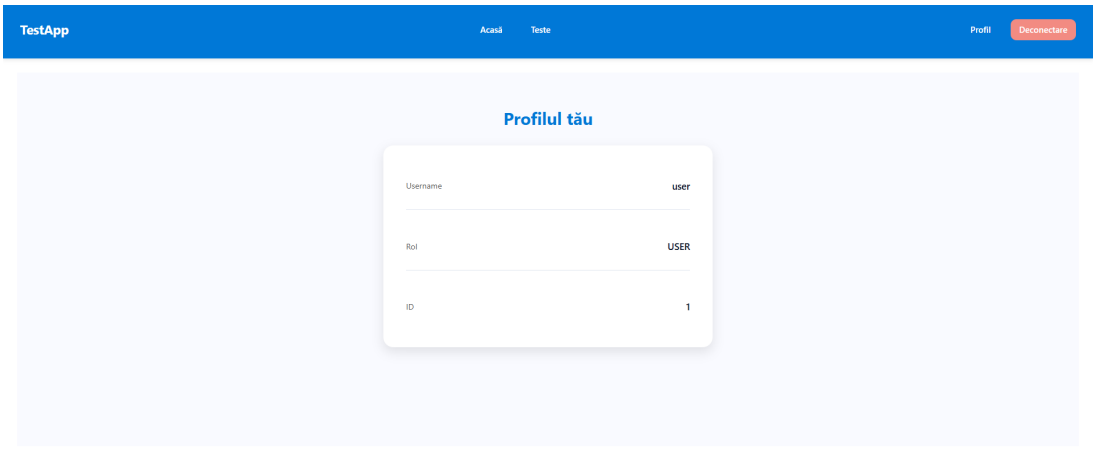

Figura 9: Profilul utilizatorului

Gestiunea Testelor

Față de interfața guest, aici apare butonul „Creează Test”. Testele proprii au butoane suplimentare pentru editare și ștergere.

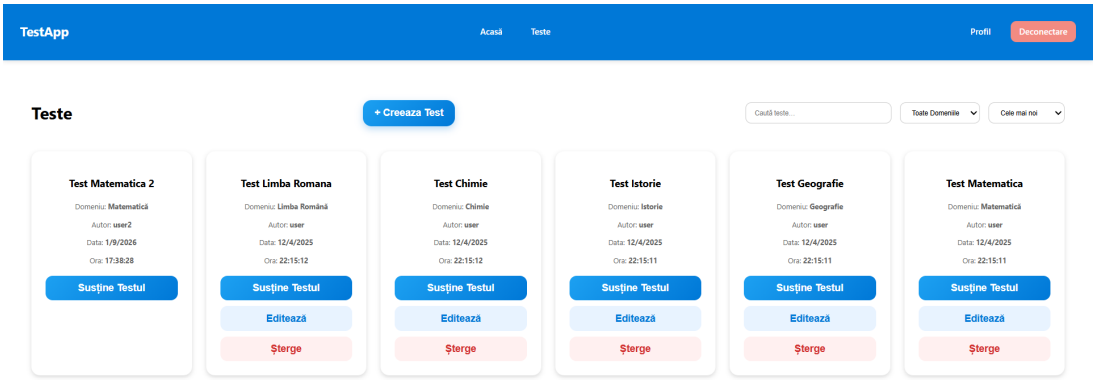

Figura 10: Lista testelor cu opțiuni de administrare

Creare și Editare Test

Permite definirea titlului, domeniului și adăugarea dinamică a întrebărilor și răspunsurilor.

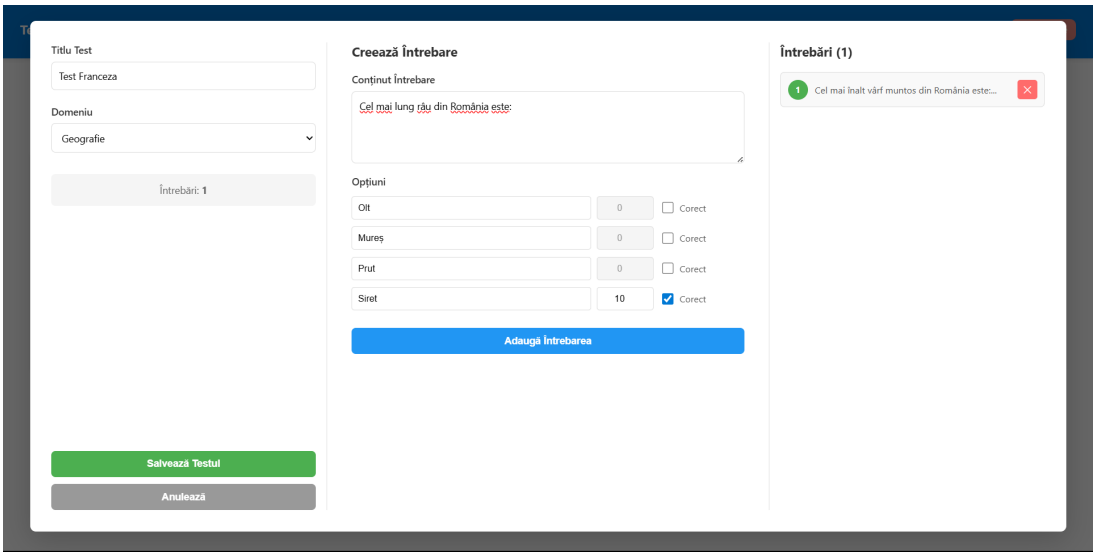

Figura 11: Crearea unui test nou

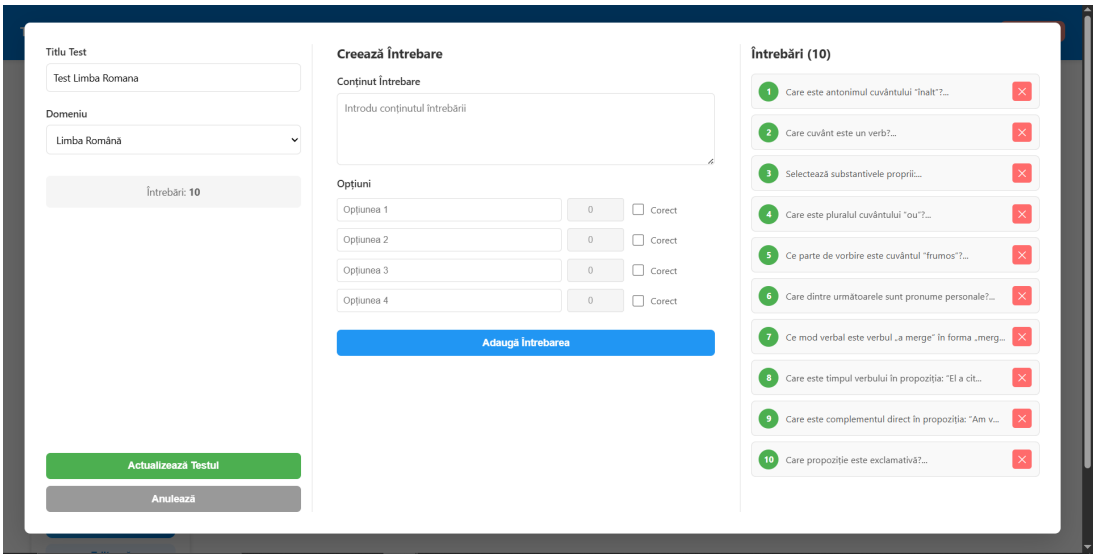

Figura 12: Editarea unui test existent

4 Implementare Back-End

4.1 Tehnologii și Arhitectură

Componenta de server a aplicației este dezvoltată pe platforma **Java**, utilizând framework-ul **Spring Boot** pentru a asigura o configurare rapidă și o arhitectură robustă de tip REST API. Sistemul respectă o arhitectură stratificată (*Layered Architecture*), care separă clar responsabilitățile între gestionarea cererilor HTTP, logica de business și persistența datelor.

Principalele tehnologii și concepte integrate sunt:

- **Spring Boot Web**: Utilizează containerul de inversare a controlului (IoC) și *Dependency Injection* pentru gestionarea componentelor și expunerea endpoint-urilor REST.
- **Spring Data JPA (Hibernate)**: Asigură interacțiunea cu baza de date relațională (MySQL). Entitățile sunt mapate direct pe tabele, iar interfețele de tip `Repository` abstractizează operațiunile SQL, permițând manipularea datelor prin metode orientate pe obiect.
- **Spring Security & JWT**: Implementează mecanismul de autentificare și autorizare. Sesiunea este configurată ca fiind *stateless* (fără stare), securitatea bazându-se pe token-uri JWT (*JSON Web Tokens*) transmise la fiecare cerere.
- **Spring WebSocket (STOMP)**: Facilitează comunicarea bidirecțională în timp real, esențială pentru monitorizarea utilizatorilor activi pe un test.
- **MapStruct**: O bibliotecă utilizată pentru maparea automată și eficientă între entitățile bazei de date și obiectele de transfer de date (DTO), asigurând decuplarea structurii interne de datele expuse clientului.

4.2 Structura Logică a Aplicației

Codul este organizat în straturi distincte, fiecare având un rol specific în procesarea datelor:

Stratul Controller

Acesta este punctul de intrare în aplicație. Interceptează cererile HTTP (`GET`, `POST`, `PUT`, `DELETE`) venite din Front-End. Nu conține logică de business complexă, ci delegă procesarea către serviciile specializate și returnează răspunsuri sub formă de obiecte JSON (prin `ResponseEntity`).

Stratul Service

Conține "inima" aplicației. Aici sunt implementate regulile de business, validările și tranzacțiile. Metodele din acest strat orchestrează apelurile către baza de date și transformă datele brute în informații utile (de exemplu, calculul scorului unui test).

Stratul Repository

Interfațează direct cu baza de date. S-au utilizat optimizări precum *Entity Graphs* pentru a încărca eficient relațiile complexe (de exemplu, un test împreună cu toate întrebările și opțiunile sale) într-un singur query, evitând problemele de performanță de tip *N+1*.

Stratul DTO (Data Transfer Objects)

Definește structura datelor care intră și ies din API. Utilizarea DTO-urilor protejează structura bazei de date de modificări externe și permite filtrarea informațiilor sensibile (ex: parola utilizatorului nu este trimisă niciodată în răspunsuri).

4.3 Securitate și Autentificare

Sistemul de securitate este configurat pentru a intercepta toate cererile HTTP printr-un filtru personalizat.

- **Fluxul de Login**: La autentificare, serverul validează credențialele și generează un token JWT semnat criptografic, care conține ID-ul utilizatorului și rolul acestuia.
- **Validarea Cererilor**: Filtrul de securitate extrage token-ul din header-ul cererii, îi verifică validitatea și expirarea, apoi stabilește contextul de securitate pentru cererea curentă.
- **Permisiuni**: Rutele pentru vizualizarea testelor sunt publice, în timp ce operațiunile de creare, editare, ștergere și accesul la profil sunt restricționate doar utilizatorilor autentificați.

4.4 Fluxuri Principale de Implementare

1. Gestionarea Testelor Complexe (Creare și Editare)

Crearea unui test nu este o operațiune simplă, deoarece implică o structură ierarhică: un `Test` are mai multe `Întrebări`, iar fiecare `Întrebare` are mai multe `Opțiuni`.

- Implementarea utilizează tranzacții atomice. Serviciul primește un obiect complex (DTO), îl mapează în entități și stabilește relațiile părinte-copil.
- Salvarea se face în cascadă: salvarea obiectului părinte (Testul) declanșează automat salvarea tuturor întrebărilor și opțiunilor asociate, garantând integritatea datelor.

2. Susținerea Testului și Identitatea Utilizatorului

Aplicația permite accesul hibrid (utilizatori logați și vizitatori).

- În momentul accesării unui test, sistemul verifică dacă utilizatorul este autentificat.
- Pentru utilizatorii autentificați, se preia identitatea din baza de date.
- Pentru vizitatori (*Guests*), a fost implementat un serviciu dedicat care generează un nume de utilizator temporar, unic, format dintr-un șir de caractere aleatorii. Acest serviciu verifică lista utilizatorilor activi pentru a preveni duplicarea numelor în cadrul aceluiași test.

3. Evaluarea și Calculul Rezultatelor

Pentru a preveni fraudarea, validarea răspunsurilor se face exclusiv pe server (Back-End).

- Clientul trimite doar ID-urile opțiunilor selectate de utilizator.
- Serverul încarcă testul original din baza de date (inclusiv informațiile despre care opțiuni sunt corecte, informații care nu ajung niciodată la client înainte de finalizare).
- Se compară selecția utilizatorului cu baremul corect, se calculează punctajul total și procentajul, iar rezultatul este returnat clientului. Răspunsurile nu sunt salvate persistent în baza de date, evaluarea fiind un proces *stateless*.

4. Monitorizare în Timp Real (WebSockets)

Funcționalitatea care afișează "Utilizatori care susțin acest test" utilizează protocolul WebSocket peste STOMP.

- A fost implementat un serviciu care menține în memorie (folosind structuri de date concurente/thread-safe) o mapare între ID-ul testului și lista utilizatorilor conectați.
- Când un utilizator intră pe pagina de test, clientul se abonează la un canal specific testului. Serverul îl adaugă în listă și notifică toți ceilalți abonați.
- La părăsirea paginii sau închiderea browserului, un eveniment de deconectare este declanșat automat, eliminând utilizatorul din listă și actualizând interfața celorlalți participanți în timp real.

5 Structura Bazei de Date

Persistența datelor este asigurată de o bază de date relațională MySQL, având schema denumită `test_app`. Structura este normalizată pentru a asigura integritatea datelor și eficiența interogărilor, fiind compusă din 5 tabele interconectate.

5.1 Descrierea Tabelelor

1. Tabela `users`

Această tabelă stochează informațiile de autentificare și autorizare pentru utilizatorii înregistrați.

- `id_user` (INT, PK, Auto Increment): Identificatorul unic al utilizatorului.
- `username` (VARCHAR, Unique): Numele de utilizator, utilizat pentru login.
- `parola` (VARCHAR): Parola utilizatorului (stocată criptat).
- `rol` (VARCHAR): Rolul utilizatorului în aplicație (ex: *USER*, *ADMIN*).

2. Tabela `domenii`

Servește ca nomenclator pentru categoriile în care pot fi încadrate testele.

- `id_domeniu` (INT, PK, Auto Increment): Identificatorul unic al domeniului.
- `nume` (VARCHAR): Denumirea domeniului (ex: *Matematică*, *Istorie*, *IT*).

3. Tabela `teste`

Este entitatea centrală a aplicației, reprezentând testele create de utilizatori.

- `id_test` (INT, PK, Auto Increment): Identificatorul unic al testului.
- `titlu` (VARCHAR): Titlul testului.
- `data_crearii` (DATETIME): Data și ora la care a fost creat testul (implicit curentă).
- `id_user` (INT, FK): Referință către utilizatorul care a creat testul.
- `id_domeniu` (INT, FK): Referință către domeniul din care face parte testul.

4. Tabela `intrebari`

Stochează întrebările asociate fiecărui test.

- `id_intrebare` (INT, PK, Auto Increment): Identificatorul unic al întrebării.
- `continut` (VARCHAR): Textul propriu-zis al întrebării.
- `id_test` (INT, FK): Referință către testul părinte.

5. Tabela `optiuni`

Conține variantele de răspuns pentru fiecare întrebare, punctajul aferent și marcajul de corectitudine.

- `id_optiune` (INT, PK, Auto Increment): Identificatorul unic al opțiunii.
- `id_intrebare` (INT, FK): Referință către întrebarea părinte.
- `continut` (VARCHAR): Textul opțiunii de răspuns.
- `punctaj` (INT): Numărul de puncte acordat pentru selectarea acestei opțiuni.
- `is_correct` (TINYINT/BOOLEAN): Indică dacă opțiunea este corectă (1) sau greșită (0).

5.2 Relații și Constrângeri

Baza de date implementează relații de tip **One-to-Many** (Unu-la-Mai-Mulți) pentru a reflecta ierarhia testelor:

1. `users` → `teste`: Un utilizator poate crea mai multe teste.
2. `domenii` → `teste`: Un domeniu poate conține mai multe teste.
3. `teste` → `intrebari`: Un test este compus din mai multe întrebări.
4. `intrebari` → `optiuni`: O întrebare are mai multe opțiuni de răspuns.

Toate cheile străine (*Foreign Keys*) sunt configurate cu clauza `ON DELETE CASCADE`. Aceasta asigură curățarea automată a datelor: ștergerea unui test va duce automat la ștergerea tuturor întrebărilor și opțiunilor asociate acestuia, prevenind existența datelor orfane.

5.3 Schema Bazei de Date

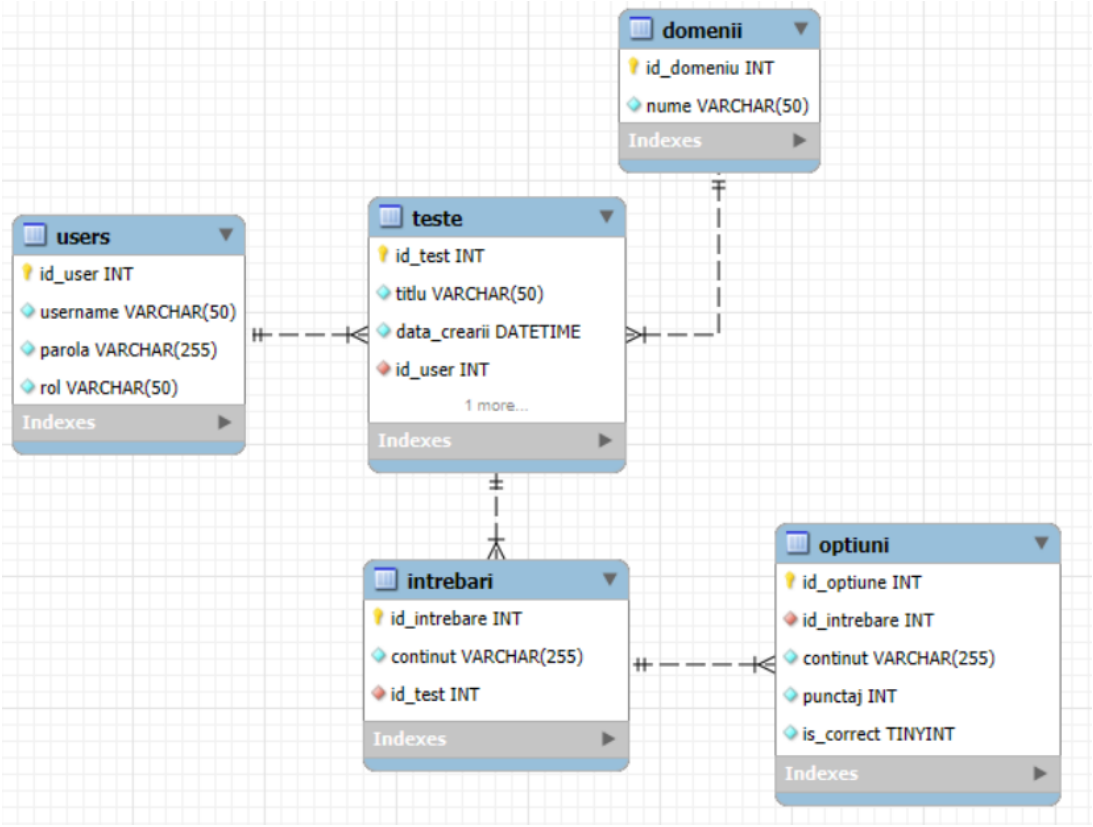

Figura 13: Schema bazei de date

6 Endpoint-uri API

Aplicația expune un API RESTful pentru comunicarea dintre client (React) și server (Spring Boot). Mai jos sunt detaliate principalele endpoint-uri, metoda HTTP asociată și descrierea funcționalității.

6.1 Autentificare și Autorizare (`/auth`)

Aceste endpoint-uri sunt publice și sunt utilizate pentru gestionarea accesului utilizatorilor.

Metodă	Endpoint	Descriere
POST	`/auth/login`	Autentifică un utilizator pe baza numelui de utilizator și a parolei. Returnează un token JWT valid dacă credențialele sunt corecte.
POST	`/auth/register`	Înregistrează un utilizator nou în baza de date cu rolul implicit de *USER*. Returnează un token JWT pentru autentificare imediată.

Metodă	Endpoint	Descriere
GET	`/auth/me`	Returnează detaliile utilizatorului curent (ID, username, rol) pe baza token-ului JWT trimis în header-ul cererii.
GET	`/auth/validate`	Verifică dacă un token JWT este valid și neexpirat.

6.2 Gestiune Teste (`/api/teste`)

Acesta este cel mai complex controller, gestionând atât operațiunile CRUD, cât și logica de susținere a testelor.

Metodă	Endpoint	Descriere
GET	`/api/teste`	Returnează lista tuturor testelor disponibile în aplicație (DTO simplificat).
GET	`/api/teste/{id}`	Returnează informațiile de bază despre un test specific (fără structura completă de întrebări).
GET	`/api/teste/{id}/info`	Returnează structura completă a unui test (inclusiv întrebări și opțiuni) pentru vizualizare.
GET	`/api/teste/{id}/take`	Returnează datele necesare pentru susținerea unui test. Pentru utilizatorii neautentificați (guest), generează și returnează un nume de utilizator temporar.
POST	`/api/teste`	Creează un test simplu (doar titlu, domeniu, user), fără întrebări.
POST	`/api/teste/create/full`	Creează un test complet într-o singură tranzacție, salvând ierarhic testul, întrebările și opțiunile de răspuns.
POST	`/api/teste/{id}/submit`	Primește răspunsurile utilizatorului, calculează scorul pe server și returnează rezultatul (punctaj, procentaj, status promovat).
PUT	`/api/teste/{id}`	Actualizează informațiile de bază ale unui test (titlu, domeniu).
PUT	`/api/teste/{id}/full`	Actualizează complet un test. Șterge întrebările vechi și le recreează pe baza noii structuri primite.
DELETE	`/api/teste/{id}`	Șterge un test și toate datele asociate (întrebări, opțiuni) din baza de date.

6.3 Gestiune Domenii (`/api/domenii`)

Utilizate pentru categorisirea testelor.

Metodă	Endpoint	Descriere
GET	`/api/domenii`	Returnează lista tuturor domeniilor (ex: Matematică, Istorie).
GET	`/api/domenii/{id}`	Returnează un domeniu specific după ID.
POST	`/api/domenii`	Creează un domeniu nou.
PUT	`/api/domenii/{id}`	Actualizează numele unui domeniu existent.
DELETE	`/api/domenii/{id}`	Șterge un domeniu.

6.4 Gestiune Utilizatori (`/api/users`)

Endpoint-uri pentru administrarea conturilor de utilizator.

Metodă	Endpoint	Descriere
GET	`/api/users`	Returnează lista tuturor utilizatorilor înregistrați.
GET	`/api/users/{id}`	Returnează detaliile unui utilizator specific.
POST	`/api/users`	Creează un utilizator nou (alternativă administrativă la Register).
PUT	`/api/users/{id}`	Actualizează datele unui utilizator (username, parolă, rol).
DELETE	`/api/users/{id}`	Șterge un utilizator din sistem.

6.5 Gestiune Granulară

Deși testele sunt create de obicei prin endpoint-ul "full", aplicația expune endpoint-uri pentru manipularea individuală a componentelor unui test.

- **Întrebări** (`/api/intrebari`): Metode GET, POST, PUT, DELETE pentru gestionarea entităților de tip *Intrebare*.
- **Opțiuni** (`/api/optiuni`): Metode GET, POST, PUT, DELETE pentru gestionarea entităților de tip *Optiune*.

6.6 WebSocket (Real-time)

Endpoint-uri utilizate pentru protocolul WebSocket (peste STOMP) pentru funcționalitatea de utilizatori activi.

Tip	Cale / Topic	Descriere
GET	`/ws`	Endpoint-ul de *handshake* pentru stabilirea conexiunii WebSocket (cu suport SockJS).
SUBSCRIBE	`/topic/test/{id}/activeUsers`	Canalul la care clienții se abonează pentru a primi lista actualizată a utilizatorilor care susțin testul cu ID-ul specificat.
MESSAGE	`/app/subscribe`	Mesaj trimis de client la intrarea pe pagina de test pentru a se înregistra ca utilizator activ.

7 Github

Link GitHub: `https://github.com/ruben-23/testare-online`

8 Concluzii

Dezvoltarea aplicației de testare online a demonstrat eficiența utilizării unei arhitecturi moderne, bazată pe separarea clară între client (React) și server (Spring Boot). Obiectivele propuse în etapa de analiză au fost atinse, rezultând o platformă funcțională, securizată și scalabilă.

Principalele aspecte evidențiate în urma implementării sunt:

- **Arhitectură Robustă**: Utilizarea Spring Boot împreună cu MySQL a asigurat o bază solidă pentru gestionarea datelor și a logicii de business, garantând integritatea tranzacțiilor complexe (precum salvarea ierarhică a testelor).
- **Experiență Utilizator Fluidă**: Interfața construită în React (SPA) oferă o navigare rapidă, fără reîncărcări inutile, iar integrarea WebSockets aduce un plus de interactivitate prin funcționalitățile în timp real.
- **Securitate și Accesibilitate**: Sistemul hibrid de acces permite utilizarea facilă a aplicației atât de către persoanele care doresc să testeze platforma rapid (Guest), cât și de către utilizatorii care doresc funcționalități avansate de creare și administrare, totul sub protecția unui mecanism de autentificare securizat (JWT).

În perspectivă, modularitatea codului permite extinderea facilă a proiectului. Direcțiile viitoare de dezvoltare pot include implementarea unor statistici grafice detaliate, suport pentru întrebări multimedia sau integrarea unui modul de generare automată a testelor bazat pe inteligență artificială.