UNIVERSITATEA TEHNICA
DIN CLUJ-NAPOCA
CENTRUL UNIVERSITAR NORD DIN BAIA MARE
FACULTATEA DE INGINERIE

Aplicație testare online

Bancoș Gabriel-Răzvan

Cuprins

1 Introducere

1.1 Context general

Platforma web de testare online a cunoștințelor programatorilor a fost concepută pentru a oferi o alternativă modernă și eficientă de evaluare a competențelor tehnice, având atât un beneficiu pentru intervievat, cât și pentru intervievator.

1.2 Obiective

Scopul principal al acestei platforme este de a îmbunătăți metodologia de testare a candidaților în cadrul unui domeniu complex, programarea software. În cadrul dezvoltării acestei aplicații se urmăresc îndeplinirea următoarelor obiective:

1. Implementarea unui algoritm de evaluare automată cu feedback pentru rezultatele obținute;
2. Dezvoltarea unei interfețe responsive pentru următoarele tipuri de dispozitive: telefon, desktop și tabletă;
3. Atenționarea administratorilor în legătură cu numărul de candidați din ziua respectivă, dar și din viitor;

1.3 Listă MoSCoW

În cadrul oricărui proiect de dezvoltare a unei aplicații este recomandat să începem cu o listă MoSCoW pentru a stabili prioritățile și a identifica limitele aplicației. Structura pentru metoda MoSCoW este următoarea:

- **Must-have** — funcționalități de bază care sunt esențiale aplicației;
- **Should-have** — funcționalități importante, dar nu obligatorii;
- **Could-have** — funcționalități care ar putea fi implementate;
- **Won't-have** — funcționalități care nu apar în această versiune.

Must-have	**Should-have**	**Could-have**	**Won't-have**
Autentificare (Login/Register)	Interfață responsive	Generare automată de teste (randomizare)	Supraveghere (proctoring)
CRUD pentru quiz-uri și întrebări	Dashboard rezultate	Timer de expirare / auto-submit	ML pentru predicții / recomandări
Calcul scor + afișare rezultat	Vizualizare istoric rezultate	Export rezultate (CSV/PDF)	Integrare ATS / SSO enterprise

Tabela 1: Listă MoSCoW pentru funcționalitățile aplicației

1.4 Cazuri de utilizare

Această aplicație este destinată evaluării rapide a cunoștințelor prin chestionare (quiz-uri) create de administratori și completate de către utilizatori. În acest sistem există două tipuri de utilizatori:

- **Administrator** — persoana care gestionează quiz-urile, întrebările și rezultatele;
- **Utilizator** — persoana care își creează cont, se autentifică și susține quiz-urile.

Cazuri de utilizare pentru Administratori

- Autentificare în aplicație (Login)
- Creare quiz și întrebări
- Adăugare întrebări în quiz
- Editare quiz și / sau întrebări
- Ștergere quiz și / sau întrebări
- Validare rezultate

Cazuri de utilizare pentru Utilizatori

- Înregistrare cont (Register)
- Autentificare în aplicație (Login)
- Vizualizare listă quiz-uri
- Accesare quiz
- Vizualizare rezultat

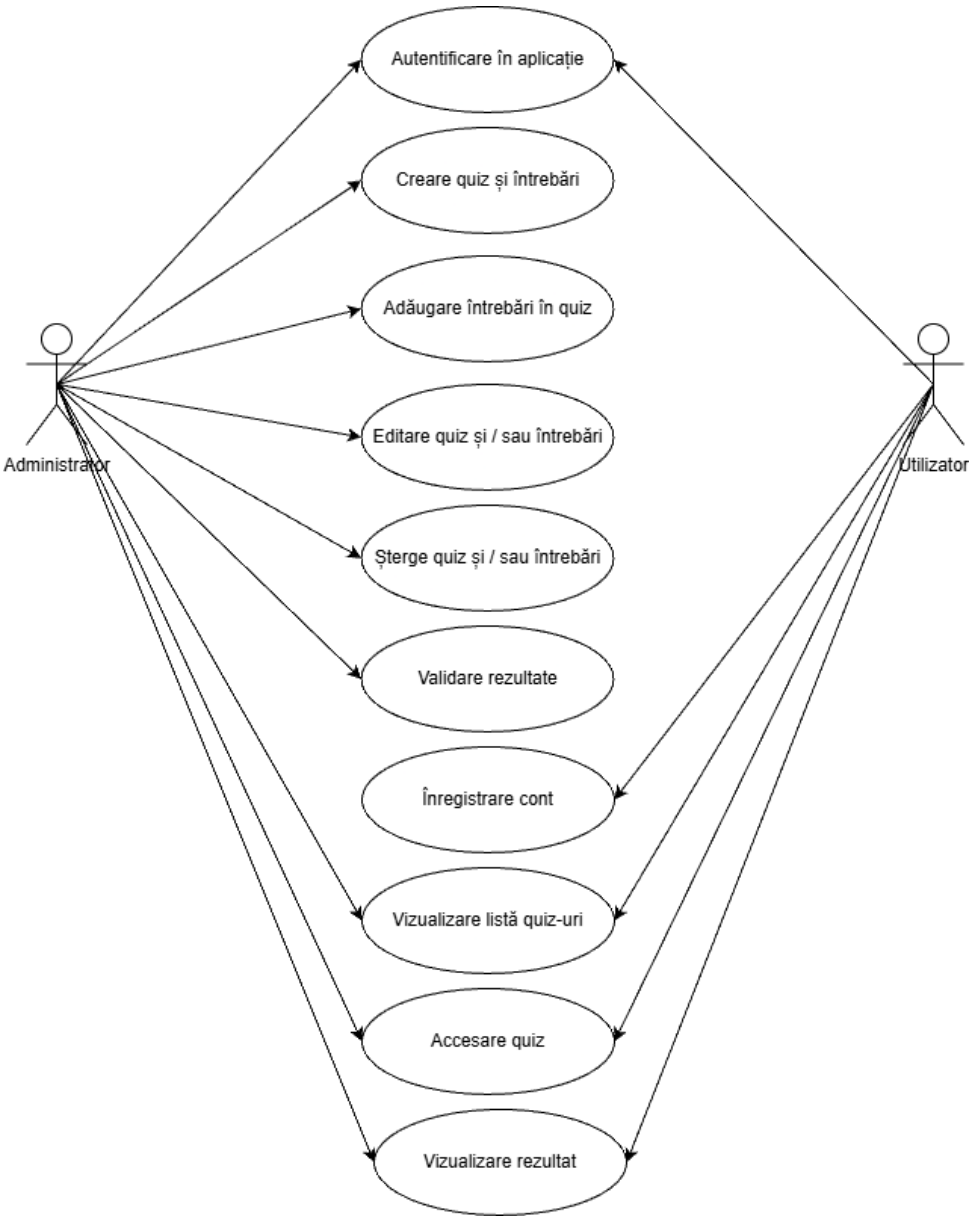

Figura 1: Diagrama UML Use Case pentru aplicația de testare online

2 Arhitectura

Arhitectura aplicației este concepută pentru a asigura performanță, scalabilitate și ușurință în utilizare. Soluția este bazată pe **Spring Boot** pentru partea de backend și **Thymeleaf** pentru generarea interfeței web, permițând o separare clară între logica aplicației și prezentare. Comunicarea în timp real este realizată prin **WebSocket**, iar stocarea datelor este gestionată cu ajutorul **Spring Data JPA** și **PostgreSQL**.

2.1 Frontend

Partea de frontend a aplicației este realizată utilizând **Thymeleaf**, un motor de template-uri integrat în ecosistemul Spring Boot, împreună cu **Tailwind CSS** și extensia sa **DaisyUI** pentru stilizarea interfeței. Această combinație permite dezvoltarea unei interfețe moderne, responsive și ușor de întreținut, păstrând în același timp simplitatea randării server-side.

Thymeleaf este utilizat pentru generarea dinamică a paginilor HTML, permițând integrarea directă a datelor provenite din backend în interfața utilizator. Tailwind CSS oferă un sistem bazat pe clase utilitare, care facilitează controlul precis asupra aspectului vizual al aplicației, iar DaisyUI adaugă un set de componente predefinite ce accelerează procesul de dezvoltare.

Principalele avantaje ale utilizării acestei soluții de frontend includ:

- **Randare server-side** – paginile sunt generate pe server, reducând complexitatea logicii din client;
- **Design responsive** – Tailwind CSS permite adaptarea interfeței pentru diferite dimensiuni de ecran (desktop, tabletă, mobil);
- **Dezvoltare rapidă** – DaisyUI oferă componente UI gata de utilizare (butoane, formulare, carduri);
- **Cod curat și modular** – stilizarea prin clase utilitare elimină necesitatea fișierelor CSS complexe;
- **Integrare facilă cu Spring Boot** – Thymeleaf funcționează nativ cu controllerele MVC și mecanismele de validare.

Arhitectura părții de frontend include următoarele componente:

- **Template-uri Thymeleaf** – pagini HTML dinamice pentru autentificare, gestionarea quiz-urilor și afișarea rezultatelor;
- **Tailwind CSS** – sistem de stilizare bazat pe clase utilitare pentru controlul layout-ului și al designului;
- **DaisyUI** – bibliotecă de componente UI reutilizabile, construită peste Tailwind CSS;
- **Formulare HTML** – utilizate pentru autentificare, înregistrare și completarea quiz-urilor;
- **JavaScript** – folosit pentru interacțiuni minime și integrarea clientului WebSocket;
- **WebSocket Client** – permite actualizarea în timp real a informațiilor și afișarea notificărilor fără reîncărcarea paginii.

2.2 Backend

Backend-ul aplicației este implementat în **Spring Boot** și are rolul de a gestiona întreaga logică de business: autentificare și înregistrare, administrarea quiz-urilor și a întrebărilor, rularea testelor, calculul scorurilor și persistarea rezultatelor. Arhitectura backend-ului urmează un stil pe straturi, ceea ce permite separarea clară între accesul la date, regulile de business și expunerea funcționalităților către interfața web.

2.2.1 Structură pe straturi

Aplicația este organizată în următoarele straturi:

- **Controller** — primește cererile HTTP, validează datele de intrare și întoarce răspunsuri (HTML prin Thymeleaf sau redirecționări).
- **Service** — conține logica principală: validări suplimentare, reguli de punctaj, verificarea drepturilor și gestionarea fluxului de rulare a unui quiz.
- **Repository** — layer-ul de persistență bazat pe Spring Data JPA, responsabil de interacțiunea cu baza de date.
- **Model / Entity** — definește entitățile persistente (User, Quiz, Question, Result etc.) și relațiile dintre ele.

2.2.2 Autentificare și securitate

Autentificarea este realizată prin mecanisme specifice aplicațiilor web (session-based), iar parolele sunt stocate în mod sigur prin hash-uire folosind **Spring Security Crypto**. La înregistrare, parola introdusă este procesată cu un algoritm de hash (ex.: BCrypt), iar la autentificare se compară hash-ul stocat cu valoarea rezultată din parola introdusă.

Pentru separarea rolurilor, utilizatorii sunt împărțiți în:

- **Administrator** — poate crea/edita/șterge quiz-uri și întrebări, poate vizualiza și valida rezultate;
- **Utilizator** — poate parcurge quiz-urile disponibile și poate susține testele.

2.2.3 Validare și integritatea datelor

Datele primite din formulare (creare quiz, creare întrebări, înregistrare, completare test) sunt validate folosind **Spring Boot Validation**. Astfel se asigură constrângeri precum câmpuri obligatorii, lungime minimă, format corect (ex.: email), precum și consistența datelor înainte de salvare.

2.2.4 Rularea testelor și calculul scorului

Fluxul de susținere a unui quiz presupune:

1. Selectarea unui quiz de către utilizator;
2. Afișarea întrebărilor asociate quiz-ului;
3. Trimiterea răspunsurilor către backend;
4. Calcularea scorului prin compararea răspunsurilor utilizatorului cu răspunsurile corecte;

5. Salvarea rezultatului în baza de date și afișarea feedback-ului.

Scorul este calculat ca număr de răspunsuri corecte (sau procent), iar rezultatul poate include informații suplimentare precum data susținerii, durata și detalii despre răspunsurile greșite (în funcție de nivelul de feedback dorit).

2.2.5 Comunicare în timp real prin WebSocket

Pentru notificări în timp real (de exemplu, atenționarea administratorilor privind numărul de candidați din ziua curentă sau programări viitoare), aplicația utilizează **WebSocket**. Backend-ul publică evenimente (ex.: la începerea unui quiz, la finalizarea unui quiz, la crearea unei programări), iar interfața administratorului primește actualizările fără a reîncărca pagina.

2.2.6 Integrare cu baza de date

Persistența datelor este realizată cu **Spring Data JPA** peste **PostgreSQL**. Repository-urile oferă operații CRUD standard, iar relațiile dintre entități (de exemplu Quiz–Question, User–Result) sunt modelate folosind adnotările JPA. Pentru eficiență, interogările pot fi extinse prin metode derivate (findBy...) sau query-uri custom atunci când este necesar (de exemplu, pentru istoricul rezultatelor unui utilizator sau pentru rapoarte administrative).

2.3 Baza de date PostgreSQL

PostgreSQL a fost ales ca sistem de gestionare a bazelor de date relaționale datorită stabilității, suportului avansat pentru tranzacții și integrității datelor, precum și compatibilității excelente cu aplicațiile de tip enterprise. În cadrul aplicației, PostgreSQL este utilizat pentru persistarea utilizatorilor, rolurilor, quiz-urilor, întrebărilor, opțiunilor de răspuns și a rezultatelor obținute de utilizatori în urma testelor.

Principalele avantaje ale PostgreSQL:

- **Fiabilitate** — conformitate ACID și tranzacții sigure, care garantează consistența datelor;
- **Performanță** — suport solid pentru indexare și optimizare atât pentru citire, cât și pentru scriere;
- **Scalabilitate** — poate gestiona volume mari de date și conexiuni simultane;
- **Flexibilitate** — suport pentru tipuri de date avansate și extensibilitate (unde este necesar);
- **Ecosistem matur** — integrare stabilă cu Spring Boot / Hibernate și instrumente de administrare.

2.3.1 Integrare JPA (Java Persistence API)

Persistența este realizată prin **Spring Data JPA**, care oferă o abstractizare de nivel înalt peste SQL, eliminând necesitatea scrierii manuale a interogărilor pentru operațiile uzuale. JPA permite maparea automată între clasele Java (entități) și tabelele din PostgreSQL, precum și definirea relațiilor dintre entități folosind adnotări.

Caracteristici principale utilizate:

- **Object-Relational Mapping (ORM)** — mapare automată între obiecte Java și tabelele PostgreSQL;
- **Repository Pattern** — interfețe simple pentru operații CRUD fără implementare explicită;
- **Query Methods** — generare automată de query-uri bazate pe numele metodelor;
- **Tranzacții** — gestionare declarativă prin adnotări precum `@Transactional`;
- **Relații între entități** — modelarea relațiilor OneToMany / ManyToOne prin adnotări JPA.

2.3.2 Structura bazei de date

Structura bazei de date include tabele care corespund entităților definite în aplicație:

- **users** — utilizatorii aplicației (administratori și participanți), cu date de autentificare și email unic (*entitatea* `User`);
- **roles** — rolurile disponibile (ex.: `ADMIN`, `USER`) (*entitatea* `Role`);
- **quizzes** — quiz-urile create de administratori, cu titlu, descriere, status de publicare și autor (*entitatea* `Quiz`);
- **questions** — întrebările asociate fiecărui quiz, incluzând tipul (Single / Multiple choice) (*entitatea* `Question`);
- **answer_options** — opțiunile de răspuns pentru întrebări, cu marcaj pentru răspuns corect (*entitatea* `AnswerOption`);
- **quiz_attempts** — încercările de susținere ale unui quiz (utilizator, start, finalizare, scor) (*entitatea* `QuizAttempt`);
- **user_answers** — răspunsurile utilizatorului la întrebări în cadrul unei încercări (*entitatea* `UserAnswer`).

2.3.3 Relațiile dintre entități

Relațiile dintre entități sunt modelate cu ajutorul adnotărilor JPA și reflectă structura logică a aplicației:

- **Role ↔ User: OneToMany**
  Un rol poate fi asociat mai multor utilizatori, iar fiecare utilizator are exact un rol.
  (`Role.users` ↔ `User.role`)
- **User ↔ QuizAttempt: OneToMany**
  Un utilizator poate avea multiple încercări (attempts) pentru quiz-uri.
  (`QuizAttempt.user`)
- **Quiz ↔ Question: OneToMany**
  Un quiz conține un set de întrebări.
  (`Quiz.questions` ↔ `Question.quiz`)
- **Question ↔ AnswerOption: OneToMany**
  O întrebare are mai multe opțiuni de răspuns, iar una sau mai multe pot fi corecte în funcție de tip.
  (`Question.options` ↔ `AnswerOption.question`)
- **Quiz ↔ QuizAttempt: OneToMany**
  Un quiz poate fi susținut de mai mulți utilizatori, generând încercări multiple.

(`QuizAttempt.quiz`)

- **QuizAttempt ↔ UserAnswer: OneToMany**
  O încercare conține răspunsurile utilizatorului pentru întrebările quiz-ului.
  (`QuizAttempt.answers` ↔ `UserAnswer.attempt`)
- **Question ↔ UserAnswer: ManyToOne**
  Fiecare răspuns este asociat unei întrebări.
  (`UserAnswer.question`)
- **AnswerOption ↔ UserAnswer: ManyToOne (opțional)**
  Răspunsul selectat de utilizator este reprezentat de o opțiune (`selectedOption`).
  (`UserAnswer.selectedOption`)

2.3.4 Integritate și ștergeri în lanț (Cascade)

Pentru menținerea integrității datelor și pentru simplificarea operațiilor administrative, anumite relații sunt configurate cu `CascadeType.ALL` și `orphanRemoval=true`. Astfel:

- La ștergerea unui **quiz**, sunt șterse automat întrebările asociate (`Quiz` → `Question`);
- La ștergerea unei **întrebări**, sunt șterse automat opțiunile asociate (`Question` → `AnswerOption`);
- La ștergerea unei **încercări** (*attempt*), sunt șterse automat răspunsurile asociate (`QuizAttempt` → `UserAnswer`).

2.3.5 Observație privind întrebările `MULTIPLE_CHOICE`

Modelul actual salvează pentru fiecare întrebare o singură opțiune selectată (`UserAnswer.selectedOption`), ceea ce corespunde perfect pentru întrebările de tip `SINGLE_CHOICE`. Pentru suport complet `MULTIPLE_CHOICE`, este necesară o extindere a modelului, de exemplu:

- utilizarea unei relații `ManyToMany` între `UserAnswer` și `AnswerOption`, sau
- stocarea selecțiilor ca mai multe înregistrări (câte una pentru fiecare opțiune selectată) pentru aceeași întrebare și aceeași încercare.

2.4 Beneficii ale acestei arhitecturi

Arhitectura aleasă oferă multiple avantaje care o fac potrivită pentru diverse contexte de utilizare, de la evaluări educaționale până la procese de recrutare sau testări interne în companii. Combinarea **Spring Boot** (backend), **Thymeleaf** (randare server-side), **Spring Data JPA** cu **PostgreSQL** (persistență) și **WebSocket** (comunicare în timp real) asigură un echilibru între performanță, mentenabilitate și extensibilitate.

- **Scalabilitate** — Separarea clară a responsabilităților în straturi (Controller–Service–Repository) permite extinderea aplicației fără modificări majore în cod. Persistența prin PostgreSQL și utilizarea WebSocket-urilor permit gestionarea unui număr mare de utilizatori și actualizări în timp real, iar componentele pot fi optimizate independent (de exemplu, caching la nivel de servicii sau optimizări ale interogărilor).
- **Eficiență în dezvoltare** — Spring Boot accelerează dezvoltarea prin configurare minimă și integrare directă cu biblioteci esențiale (JPA, Validation,

WebSocket). Thymeleaf permite dezvoltarea rapidă a interfețelor web prin template-uri, fără a necesita un framework frontend separat. Spring Data JPA reduce codul repetitiv prin repository-uri și metode de tip *query derivation*.

- **Performanță optimizată** — Randarea server-side prin Thymeleaf scade complexitatea din client și reduce timpul până la afișarea conținutului pentru utilizator. PostgreSQL oferă suport excelent pentru indexare și tranzacții, iar JPA permite optimizarea accesului la date prin relații și query-uri specifice. Comunicarea prin WebSocket elimină nevoia de refresh-uri repetate sau polling.
- **Flexibilitate și extensibilitate** — Structura modulară a aplicației permite adăugarea facilă de funcționalități noi (ex.: categorii de quiz-uri, export rezultate, timer, randomizare). Dependency Injection facilitează introducerea de servicii noi fără schimbări masive în componentele existente, iar modelarea entităților JPA poate fi extinsă gradual (de exemplu, suport complet pentru `MULTIPLE_CHOICE`).
- **Mentenabilitate** — Separarea între prezentare (Thymeleaf + UI), logica de business (Service) și accesul la date (Repository) crește lizibilitatea și ușurează depanarea. Codul este organizat pe responsabilități clare, iar schimbările într-o zonă (de exemplu în UI) au impact minim asupra celorlalte componente.
- **Securitate** — Parolele sunt stocate în mod sigur prin mecanisme de hash (folosind **Spring Security Crypto**), iar aplicația poate implementa control pe roluri (`ADMIN`/`USER`) la nivel de rute și funcționalități. PostgreSQL contribuie la integritatea datelor prin tranzacții ACID, iar validarea datelor de intrare prin **Spring Validation** reduce riscul introducerii de date invalide.
- **Comunicare în timp real** — WebSocket-urile permit notificări instant și actualizări live (ex.: informarea administratorilor când utilizatorii finalizează quiz-uri sau când crește numărul de participanți într-o anumită zi). Acest lucru îmbunătățește semnificativ experiența de utilizare și reduce latența informațiilor afișate în interfața de administrare.
- **Compatibilitate și accesibilitate** — Aplicația rulează în browser și nu necesită instalări suplimentare. Interfața poate fi adaptată ușor pentru dispozitive diferite (desktop, tabletă, telefon) datorită abordării responsive (Tailwind CSS + DaisyUI), iar randarea server-side menține funcționalitățile accesibile și pe dispozitive mai puțin performante.
- **Testabilitate crescută** — Dependency Injection permite înlocuirea ușoară a componentelor în teste (mock-uri pentru servicii/repository-uri). Separarea responsabilităților facilitează testarea unitară a logicii (Service) și testarea de integrare pentru fluxuri complete (Controller + DB).

În concluzie, arhitectura propusă susține o dezvoltare durabilă și eficientă, oferind atât performanță și securitate, cât și flexibilitatea necesară pentru evoluții viitoare. Prin utilizarea tehnologiilor moderne și stabile (*Spring Boot, Thymeleaf, Spring Data JPA, PostgreSQL, WebSocket*), aplicația poate deservi scenarii variate precum evaluări educaționale, certificări, training intern sau procese de recrutare, menținând o experiență de utilizare fluentă și ușor de administrat.

3 Implementare Frontend

Dezvoltarea interfeței utilizator a fost realizată având ca obiectiv principal crearea unei experiențe fluide, intuitive și responsive. Implementarea frontend-ului se bazează pe randare server-side prin **Thymeleaf**, cu accent pe modularitate și reutilizabilitate, respectând principiile moderne de design și arhitectură software. Pentru stilizare s-au folosit **Tailwind CSS** și extensia **DaisyUI**, care oferă componente reutilizabile și un sistem coerent de design.

3.1 Tehnologii Frontend

Pentru realizarea interfeței grafice au fost utilizate tehnologii moderne, integrate în ecosistemul Spring Boot:

- **Thymeleaf** — motor de template-uri utilizat pentru generarea dinamică a paginilor HTML pe server. Thymeleaf permite integrarea directă a datelor provenite din backend în UI (de exemplu, listarea quiz-urilor, afișarea întrebărilor și a rezultatelor), fără a necesita o aplicație SPA separată. În plus, suportă ușor validările și mesajele de eroare din Spring MVC.
- **Spring MVC (Model–View–Controller)** — folosit pentru gestionarea rutelor și a fluxurilor aplicației (ex.: `/login`, `/register`, `/quizzes`, `/admin`). Controller-ele furnizează modelul de date către template-urile Thymeleaf, iar navigarea se realizează prin redirecționări și încărcare de pagini.
- **Tailwind CSS** — framework de stilizare bazat pe clase utilitare, utilizat pentru controlul rapid și precis al layout-ului (grid, flex, spațiere, tipografie) și pentru implementarea unei interfețe responsive adaptate la desktop, tabletă și mobil.
- **DaisyUI** — extensie peste Tailwind CSS, care oferă componente UI predefinite (butoane, formulare, carduri, alerte, modale). Aceasta reduce timpul de dezvoltare și asigură consistența vizuală în întreaga aplicație.
- **JavaScript** — folosit pentru interacțiuni punctuale în pagini (de exemplu, confirmări, afișare/ascundere elemente, validări minore) și pentru integrarea cu partea de notificări în timp real.
- **WebSocket Client** — utilizat în paginile de administrare pentru recepționarea notificărilor în timp real trimise de backend (de exemplu, finalizarea unui quiz de către un utilizator sau actualizări privind numărul de participanți). Astfel, interfața se poate actualiza fără reîncărcarea paginii.

3.1.1 Principii de implementare a interfeței

Interfața a fost proiectată cu focus pe claritate și ergonomie, urmărind:

- **Reutilizabilitate** — componente vizuale repetabile (carduri, formulare, tabele) construite cu clase Tailwind/DaisyUI;
- **Responsive design** — adaptare automată la rezoluții diferite prin breakpoints Tailwind;
- **Accesibilitate** — formulare clare, mesaje de eroare vizibile și feedback imediat la acțiuni;
- **Experiență fluidă** — pagini structurate logic, navigare simplă și notificări în timp real pentru administratori.

4 Implementare Backend

Backend-ul aplicației reprezintă nucleul funcțional, responsabil de logica de business, securitate și persistența datelor. Arhitectura server-side este construită pe o stivă tehnologică robustă, bazată pe ecosistemul Java și Spring, asigurând scalabilitate, mentenabilitate și integrare facilă cu interfața web realizată prin Thymeleaf.

4.1 Tehnologii Backend

Selecția tehnologiilor a fost realizată pentru a asigura o integrare coerentă între componente și o performanță optimă, menținând în același timp simplitatea dezvoltării și extinderii aplicației.

4.1.1 Spring Boot

Spring Boot a fost ales ca framework principal datorită abordării de tip *Convention over Configuration*, care reduce semnificativ configurațiile boilerplate și accelerează dezvoltarea. Acesta orchestrează modulele aplicației (web, persistență, validare, WebSocket), oferind o bază solidă pentru un backend modular și ușor de întreținut.

Spring MVC (Web Layer) Aplicația utilizează **Spring MVC** pentru gestionarea rutelor și a fluxurilor de navigare. Controller-ele procesează cererile HTTP, validează datele de intrare și returnează pagini HTML generate dinamic cu Thymeleaf sau redirecționări către alte rute.

Spring Data JPA Componenta de persistență folosește **Spring Data JPA** pentru a abstractiza interacțiunea cu baza de date **PostgreSQL**. Prin utilizarea interfețelor de tip Repository (ex.: `UserRepository`, `QuizRepository`, `QuestionRepository`), se elimină necesitatea scrierii manuale de SQL pentru operațiile uzuale. Framework-ul generează automat interogări pe baza numelor metodelor (de exemplu, `findByUsername`, `findByEmail`), iar pentru cerințe mai specifice se pot utiliza interogări custom (JPQL), menținând independența față de dialectul bazei de date.

Dependency Injection (DI) Mecanismul de **Dependency Injection** al Spring, prin containerul **Inversion of Control (IoC)**, gestionează ciclul de viață al componentelor (bean-uri). Serviciile (ex.: `QuizService`, `UserService`, `AttemptService`) sunt injectate automat în Controllere, decuplând logica de business de stratul web. Această abordare crește modularitatea codului și facilitează testarea unitară prin înlocuirea dependențelor cu mock-uri.

Validare (Spring Validation) Validarea datelor introduse de utilizatori este realizată prin **Spring Boot Validation** (adnotări precum `@NotNull`, `@Size`, `@Email`). Astfel, aplicația asigură integritatea datelor înainte de persistare și oferă feedback clar în formulare (de exemplu, la înregistrare sau la crearea unui quiz).

Securitate (Spring Security Crypto) Securitatea credențialelor este asigurată prin **Spring Security Crypto**. Parolele nu sunt stocate în clar, ci sunt hash-uite (de exemplu folosind BCrypt) înainte de salvarea în baza de date. La autentificare,

parola introdusă este comparată cu hash-ul stocat, asigurând protecția utilizatorilor în cazul unor accesări neautorizate ale bazei de date.

4.1.2 WebSockets

Pentru a îmbunătăți experiența de utilizare și a permite actualizări instant, aplicația integrează suport **WebSocket**. Această tehnologie este folosită în special pentru notificări către administratori (de exemplu, finalizarea unui quiz de către un utilizator, modificări în numărul de participanți, actualizări de tip dashboard), fără a fi necesară reîncărcarea paginii.

Implementarea comunicării WebSocket Configurarea se realizează printr-o clasă dedicată (de tip `WebSocketConfig`), care expune un endpoint de conectare (de exemplu `/ws`) pentru client și definește canale de comunicare pentru transmiterea mesajelor către interfața web. Mesajele pot fi publicate către un canal de tip *publish-subscribe* (de exemplu `/topic`), astfel încât mai mulți clienți conectați să primească simultan actualizările.

4.1.3 PostgreSQL

Ca sistem de gestiune a bazei de date s-a optat pentru **PostgreSQL**, datorită stabilității, suportului avansat pentru tranzacții și integrității datelor. Modelul relațional a fost proiectat pentru a asigura integritatea referențială (chei străine între tabelele `users`, `roles`, `quizzes`, `questions`, `answer_options`, `quiz_attempts`, `user_answers`) și eficiența interogărilor. Utilizarea strategiilor de generare a cheilor primare (de tip `GenerationType.IDENTITY`) asigură unicitatea identificatorilor la nivelul întregii aplicații.

4.1.4 Maven

Managementul dependențelor și automatizarea procesului de build sunt asigurate de **Apache Maven**. Fișierul `pom.xml` centralizează librăriile necesare aplicației (Spring Boot Starter Web, Thymeleaf, Data JPA, WebSocket, Validation, PostgreSQL Driver etc.), garantând consistența versiunilor și reproductibilitatea build-ului pe orice mediu. Plugin-ul `spring-boot-maven-plugin` permite împachetarea aplicației într-un fișier `JAR` executabil, care poate fi rulat independent.

4.2 Gestionarea cererilor HTTP

Interacțiunea dintre interfață și backend se realizează prin cereri HTTP gestionate de **Spring MVC**. În cadrul proiectului, rutele sunt implementate în clase de tip `@Controller`, iar răspunsurile sunt în principal pagini HTML generate server-side cu **Thymeleaf**. Pentru anumite funcționalități (de exemplu acțiuni declanșate din JavaScript sau integrarea cu WebSocket), aplicația poate expune și endpoint-uri de tip `@ResponseBody` (JSON), însă fluxul principal rămâne unul de tip MVC.

Fiecare controller gestionează o zonă funcțională specifică:

- **AuthController** — gestionează rutele publice pentru autentificare și înregistrare (de exemplu `/login` și `/register`). Controller-ul validează datele primite din formulare, realizează autentificarea utilizatorului și redirecționează către paginile corespunzătoare după login.

- **QuizController** — gestionează funcționalitățile legate de quiz-uri pentru utilizatori: listarea quiz-urilor publicate, accesarea unui quiz, trimiterea răspunsurilor și afișarea rezultatului. De asemenea, poate include rute pentru inițierea unei încercări (*attempt*) și salvarea acesteia în baza de date.
- **AdminController** — expune funcționalitățile de administrare: creare/editare/ștergere quiz-uri și întrebări, publicare/depulicare, precum și vizualizarea rezultatelor și a statisticilor. Accesul la aceste rute este restricționat pe bază de rol (de exemplu, doar utilizatorii cu rol `ADMIN`), astfel încât operațiile sensibile să fie disponibile exclusiv administratorilor.

Prin această separare, aplicația menține o structură clară și ușor de extins: fiecare controller tratează o singură responsabilitate, iar logica de business este delegată către servicii (ex.: `UserService`, `QuizService`, `AttemptService`), păstrând controller-ele cât mai simple și orientate pe fluxul cererilor HTTP.

5 Concluzii

Proiectul dezvoltat reprezintă o platformă completă și modernă de testare online, care demonstrează integrarea coerentă a unor tehnologii actuale în dezvoltarea software. Prin combinarea unui backend robust realizat cu **Spring Boot** și a unei interfețe web construite cu **Thymeleaf**, **Tailwind CSS** și **DaisyUI**, s-a obținut o soluție software performantă, sigură și ușor de extins. Persistența datelor prin **Spring Data JPA** și **PostgreSQL**, împreună cu suportul **WebSocket** pentru notificări în timp real, contribuie la o experiență de utilizare fluentă atât pentru utilizatori, cât și pentru administratori.

Obiectivele inițiale ale proiectului au fost atinse în mare parte, prin implementarea următoarelor direcții:

- **Arhitectură modernă** — aplicația este organizată pe straturi (Controller–Service–Repository), cu separare clară între logica de business, accesul la date și interfața web, ceea ce crește mentenabilitatea și permite extinderea facilă.
- **Experiență unitară** — interfața responsive, realizată cu Tailwind CSS și DaisyUI, oferă o utilizare plăcută pe desktop, tabletă și telefon, păstrând consistența vizuală și ergonomia fluxurilor.
- **Securitate** — parolele sunt stocate în mod sigur prin hash-uire (Spring Security Crypto), iar accesul la funcționalitățile de administrare este controlat pe bază de roluri (`ADMIN`/`USER`), protejând datele și operațiile sensibile.
- **Interactivitate** — notificările în timp real prin WebSocket îmbunătățesc considerabil zona de administrare, oferind actualizări instant (de exemplu la finalizarea quiz-urilor sau la schimbări relevante în activitate).

Aplicația oferă o bază solidă pentru dezvoltări ulterioare și poate fi extinsă cu funcționalități suplimentare, precum: randomizarea automată a testelor, introducerea unui timer cu auto-submit, exportul rezultatelor (CSV/PDF), rapoarte statistice avansate sau integrarea cu platforme externe. De asemenea, pot fi adăugate mecanisme suplimentare de analiză a performanței (de exemplu, evidențierea ariilor slabe ale candidaților) și îmbunătățiri de securitate și audit.

În concluzie, acest proiect nu reprezintă doar o demonstrație tehnică a competențelor de dezvoltare web, ci și un produs software viabil, capabil să răspundă unor nevoi reale de digitalizare a proceselor de evaluare și testare, oferind un flux clar, o administrare facilă și posibilitatea unei evoluții continue.

Github