Documentație tehnică - Online Testing App

12 ianuarie 2026

Cuprins

1 Prezentare documentație

1.1 Scopul documentației

Acest document explică, într-un limbaj clar, ce face aplicația, cum este construită și cum poate fi rulată. Este gândit să fie util atât dezvoltatorilor care intră în proiect, cât și unui QA sau PM care vrea să înțeleagă funcționalitățile și limitele. Pe scurt, acoperă arhitectura, implementarea, testarea (stare curentă și goluri) și livrarea/deploy.

1.2 Contextul proiectului

Online Testing App este o aplicație de testare grilă care permite administratorilor să creeze teste, iar participanților să le susțină și să primească rezultate imediate. Problema pe care o rezolvă este organizarea rapidă a testelor online, cu un flux de lucru simplu și feedback instant. Beneficiarii principali sunt administratorii (care gestionează testele), participanții (care rezolvă testele) și organizatorii (care urmăresc rezultatele).

1.3 Obiective și cerințe

Obiectivul major al proiectului este să ofere un ciclu complet de testare: creare test, susținere test, calcul scor și vizualizare rezultate, totul protejat de autentificare și actualizat în timp real.

Cerințe funcționale (scurt):

- Autentificare și înregistrare utilizatori, cu sesiune persistentă.
- CRUD pentru teste, rezervat administratorilor.
- Rulare test pentru participanți, cu calcul automat al scorului și salvare istoric.
- Vizualizare și administrare submisii de către admin.
- Notificări live pentru schimbări la teste și submisii.

Cerințe non-funcționale:

- Securitate: sesiune web, CSRF, parole hashate (BCrypt).
- UX: formulare cu validări, mesaje de eroare și stări de încărcare.
- Performanță: potrivit pentru grupuri mici/medii, fără optimizări avansate.
- Scalabilitate: limitată de brokerul WS in-memory și lipsa caching-ului.
- Fiabilitate: persistare în MySQL și sesiuni în baza de date.

1.4 Domeniu și limitări

În domeniul proiectului intră managementul testelor, susținerea testelor, calculul scorului, salvarea submisiilor și notificările live. Nu intră în proiect funcții precum proctoring, import/export test, programări de examene, plăți, email sau analytics avansat. Constrângerile sunt legate de stack-ul ales (Spring Boot + React + MySQL), autentificare pe sesiune (nu JWT) și un broker WebSocket simplu în memorie.

1.5 Glosar și convenții

Glosar: JWT (token stateless, nefolosit în proiect), RBAC (control acces prin roluri), CRUD (Create/Read/Update/Delete), CSRF (protecție pentru cereri mutatoare), STOMP/SockJS (protocol WebSocket).

Convenții: endpoint-uri REST sub `/api`, JSON cu `camelCase`, roluri `ROLE_ADMIN`/`ROLE_USER`, clase Java cu sufixe `Controller`/`Service`/`Repository`, componente React în `PascalCase`.

2 Arhitectura și implementarea sistemului

2.1 Viziune generală (high-level)

Sistemul are o arhitectură client-server clasică. Interfața React comunică prin HTTP/JSON cu backend-ul Spring Boot și folosește WebSockets (STOMP/SockJS) pentru actualizări live. Datele sunt persistate în MySQL, iar sesiunile sunt stocate în DB prin Spring Session JDBC.

Fluxurile principale sunt următoarele:

- **Autentificare:** UI obține token CSRF, trimite `/login`, apoi verifică starea prin `/api/auth/me`.
- **CRUD teste:** adminul creează/actualizează/șterge teste prin `/api/tests`, iar backend-ul trimite notificări live pe `/topic/tests`.
- **Submisii:** participantul trimite răspunsuri la `/api/tests/{id}/submit`, backend-ul calculează scorul și publică evenimente pe `/topic/submissions`.

Diagramă arhitectură:

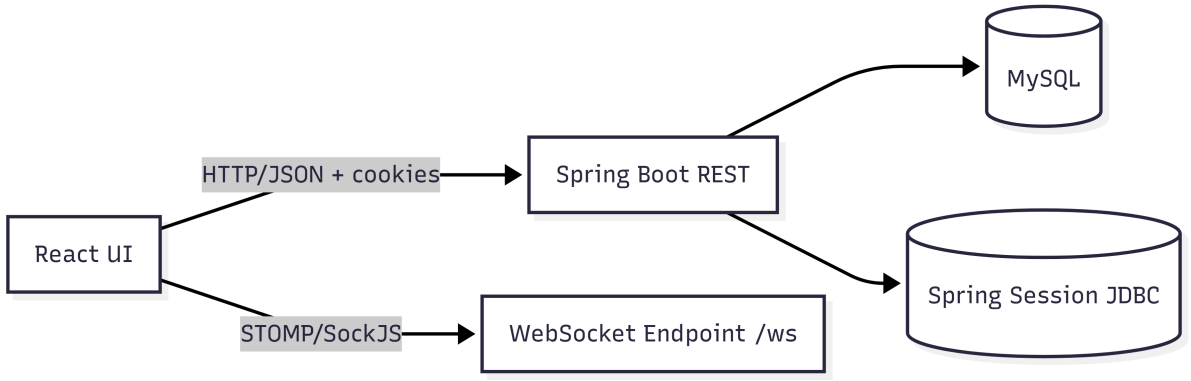

Figura 1: Arhitectura sistemului

Surse: `frontend/src/api.js`, `backend/src/main/java/com/example/testing/config/WebSocketConfig`
`backend/src/main/java/com/example/testing/service/TestNotificationService.java`

2.2 Tehnologii folosite

Stack-ul proiectului este Spring Boot 3.3.5 (Java 21, Maven) pentru backend, React 18 (Vite) pentru UI, și MySQL 8 pentru persistarea datelor. Deploy local este facilitat de Docker Compose. Pe partea de dependințe importante, se folosesc Spring Web, Spring Security, Spring Data JPA, Spring Session JDBC, Spring WebSocket (STOMP), Lombok și Springdoc OpenAPI.

Surse: `backend/pom.xml`, `frontend/package.json`

2.3 Arhitectura backend

Backend-ul este organizat pe straturi. Controlerele gestionează request/response, serviciile conțin logica de business și validările, iar repository-urile asigură accesul la baza de date prin JPA. DTO-urile sunt separate de entitățile JPA pentru a menține un contract API curat și stabil.

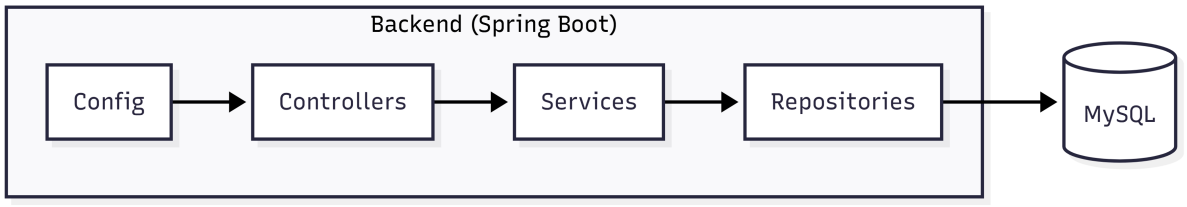

Figura 2: Arhitectura backend pe straturi

Surse: `backend/src/main/java/com/example/testing/controller/TestController.java`, `backend/src/main/java/com/example/testing/service/TestService.java`, `backend/src/main/java/com/`

2.4 Baza de date

S-a ales o bază relațională (MySQL) deoarece modelul de date este clar relațional: teste, întrebări, opțiuni de răspuns și submisii. Relațiile principale sunt între utilizatori și roluri (many-to-many), între teste și întrebări (one-to-many) și între submisii și răspunsuri (collection).

Model de date (rezumat):

```
users <-> users_roles <-> roles
tests -> questions -> answer_options
tests -> test_submissions -> submission_answers
```

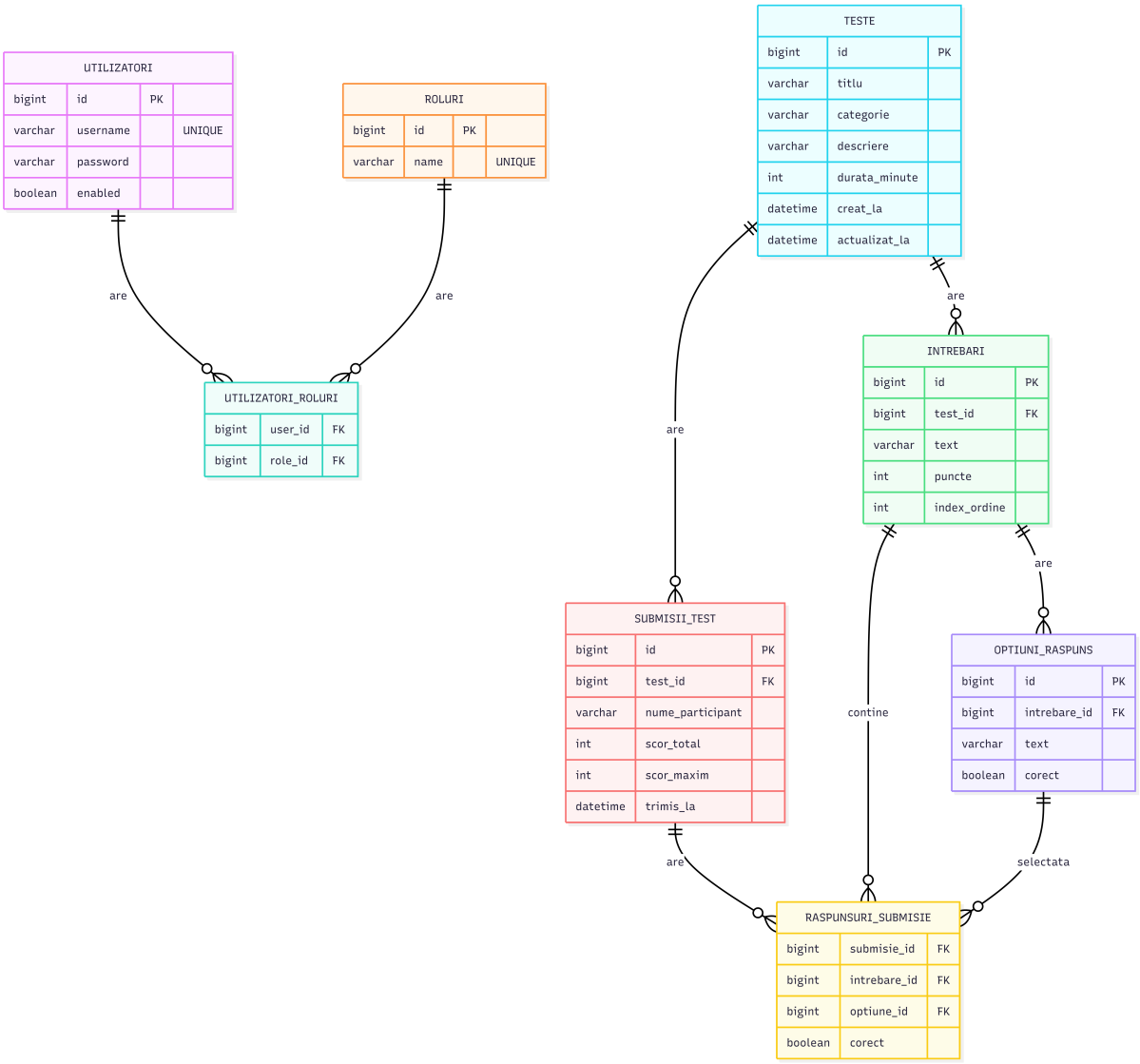

Figura 3: Modelul relațional al bazei de date

Constrângerile includ unicitatea username-ului și a rolului, plus câmpuri obligatorii precum titlul testului și numele participantului. Schema este generată cu `ddl-auto: update`, iar un seed inițial creează utilizatori și teste demo. Backup nu este definit în repository.

Surse: `backend/src/main/java/com/example/testing/model/User.java`, `backend/src/main/java/com/` `backend/src/main/java/com/example/testing/util/DataLoader.java`, `backend/src/main/resources/app`

2.5 REST API

API-ul urmează principiile REST: resurse clare, coduri HTTP standard și payload JSON. În prezent nu există versionare, paginare sau filtrare, ceea ce este acceptabil pentru dimensiunea proiectului.

Endpoint-uri principale:

- `GET /api/auth/me` (public, status autentificare)
- `GET /api/auth/csrf` (public, token CSRF)
- `POST /api/auth/register` (public)

- `POST /login` și `POST /logout` (form login)
- `GET /api/tests` (public)
- `GET /api/tests/{id}` (public, răspunsuri corecte doar pentru admin)
- `POST /api/tests` (admin)
- `PUT /api/tests/{id}` (admin)
- `DELETE /api/tests/{id}` (admin)
- `POST /api/tests/{id}/submit` (autentificat)
- `GET /api/submissions?testId=...` (admin)
- `DELETE /api/submissions/{id}` (admin)

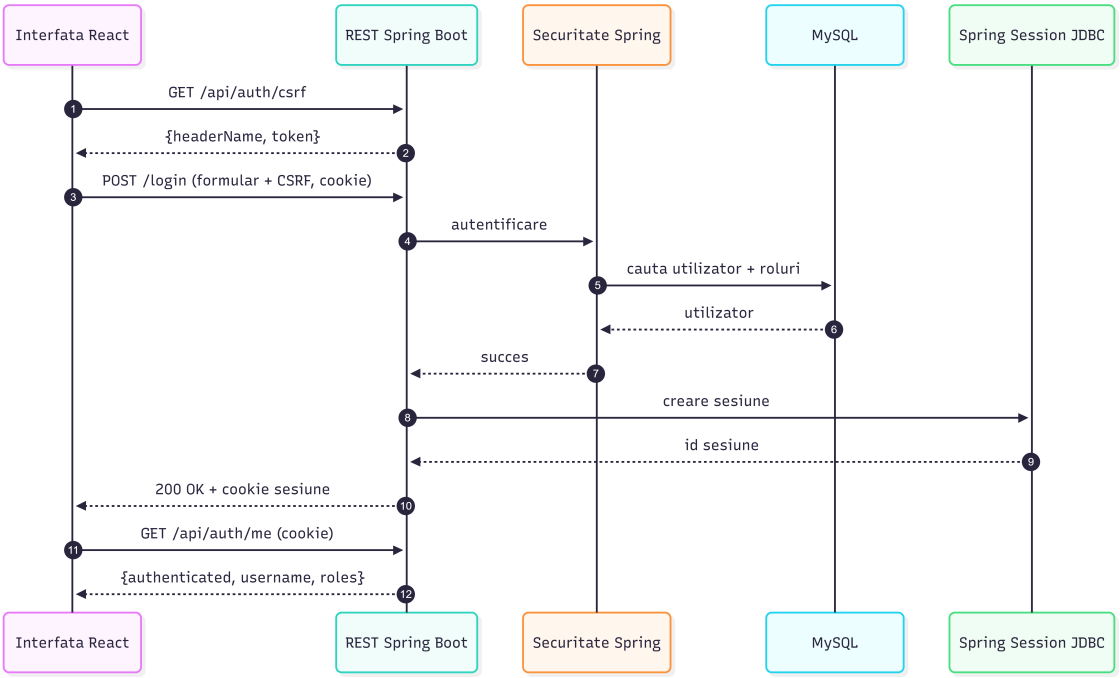

Figura 4: Flux autentificare (login + CSRF)

Exemple request/response:

```
POST /api/tests
{
  "title": "Java Fundamentals",
  "category": "Programare",
  "description": "Notiuni de baza",
  "durationMinutes": 30,
  "questions": [
    {
      "text": "Ce colectie pastreaza ordinea?",
      "points": 2,
      "orderIndex": 1,
```

```
      "options": [
        { "text": "ArrayList", "correct": true },
        { "text": "HashSet", "correct": false }
      ]
    }
  ]
}

POST /api/tests/1/submit
{
  "participantName": "student01",
  "answers": { "101": 1001 }
}

{
  "submissionId": 55,
  "testId": 1,
  "totalScore": 2,
  "maxScore": 2,
  "percentage": 100.0,
  "submittedAt": "2025-12-14T12:34:56Z"
}
```

Validările sunt făcute prin Jakarta Validation, iar erorile se întorc cu HTTP 400/404/409 și un mesaj clar, folosind `ResponseStatusException`.

Surse: `backend/src/main/java/com/example/testing/controller/AuthController.java`, `backend/src/main/java/com/example/testing/controller/TestController.java`, `backend/src/main/jav`

2.6 Securitate

Autentificarea este pe bază de sesiune (form login), cu token CSRF stocat în cookie și verificat la cererile care modifică date. Autorizarea este făcută prin roluri (`ROLE_ADMIN` și `ROLE_USER`) și adnotări `@PreAuthorize`. Parolele sunt stocate cu BCrypt, iar CORS este configurat pentru origini permise prin variabile de mediu. Nu există rate limiting sau audit logging în implementarea actuală.

Surse: `backend/src/main/java/com/example/testing/config/SecurityConfig.java`, `backend/src/mai` `backend/src/main/resources/application.yml`

2.7 Performanță și scalabilitate

Pentru a evita probleme de tip N+1, repository-urile folosesc `@EntityGraph` acolo unde e necesar. În plus, `open-in-view` este dezactivat. Limitările actuale sunt lipsa caching-ului, lipsa paginării și brokerul WebSocket in-memory, ceea ce restrânge scalarea în multi-instanta. Un pas firesc ar fi introducerea paginării, a unui broker extern (ex. RabbitMQ) și a unui cache pentru listări.

Surse: `backend/src/main/java/com/example/testing/repo/TestPaperRepository.java`, `backend/src/main/resources/application.yml`

2.8 Integrare cu servicii externe

Nu sunt integrate servicii externe (email, plăți, storage). Singura dependență externă este MySQL, configurată prin variabile de mediu.

Surse: `docker-compose.yml`, `backend/src/main/resources/application.yml`

3 Implementare Backend (detaliat)

3.1 Setup proiect

Backend-ul necesită Java 21 și Maven. În mod uzual, se rulează cu Docker Compose, care pornește și MySQL. Configurarea se face prin variabilele `SPRING_DATASOURCE_URL`, `SPRING_DATASOURCE_USERNAM` `SPRING_DATASOURCE_PASSWORD` și variabile pentru origini permise. Aplicația rulează pe portul 8080, iar cookie-ul de sesiune se numește `TESTINGSESSION`.

Surse: `backend/pom.xml`, `backend/src/main/resources/application.yml`, `docker-compose.yml`

3.2 Structura cod

Codul este împărțit pe pachete clare: `config`, `controller`, `service`, `repo`, `model`, `dto`, `util`. Această separare reduce cuplarea și face proiectul ușor de întreținut.

Surse: `backend/src/main/java/com/example/testing`

3.3 Implementarea logicii de business

Regulile de business sunt aplicate central în servicii. De exemplu, un test trebuie să aibă cel puțin o întrebare, fiecare întrebare trebuie să aibă cel puțin două opțiuni și exact un răspuns corect. Titlul testului trebuie să fie unic (case-insensitive). La submisii, participantul nu poate trimite de două ori la același test, iar scorul se calculează prin suma punctelor pentru răspunsurile corecte. După fiecare acțiune importantă (create/update/delete/submit) se publică un eveniment WebSocket.

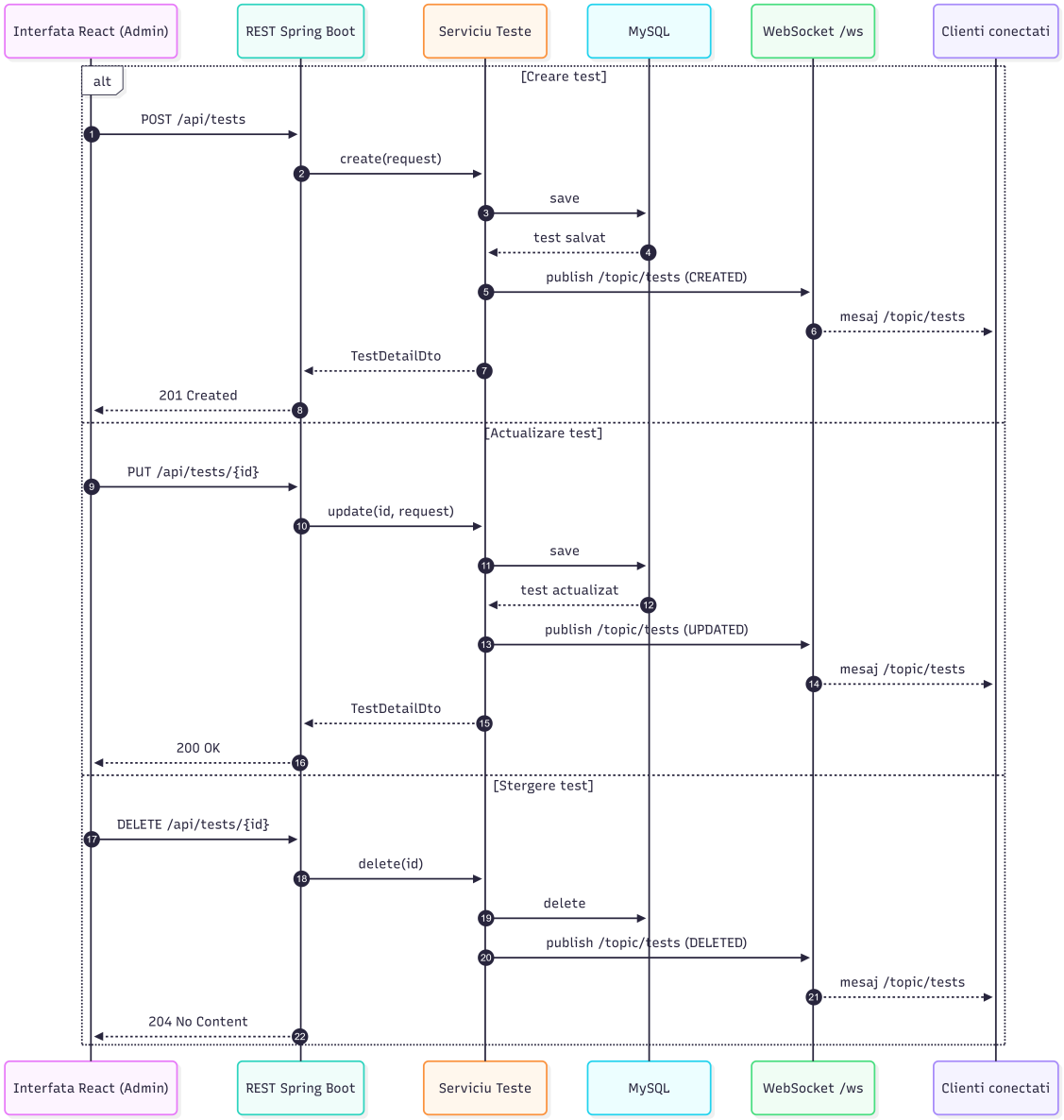

Figura 5: Flux CRUD teste (admin) cu broadcast live

Surse: `backend/src/main/java/com/example/testing/service/TestService.java`, `backend/src/main/java/com/example/testing/service/TestNotificationService.java`

3.4 Implementarea accesului la date

Persistența este realizată cu Spring Data JPA. Repository-urile oferă atât operații standard, cât și metode custom (ex. submisii pentru un test). Tranzacțiile sunt declarate la nivel de serviciu pentru operațiile care scriu în DB.

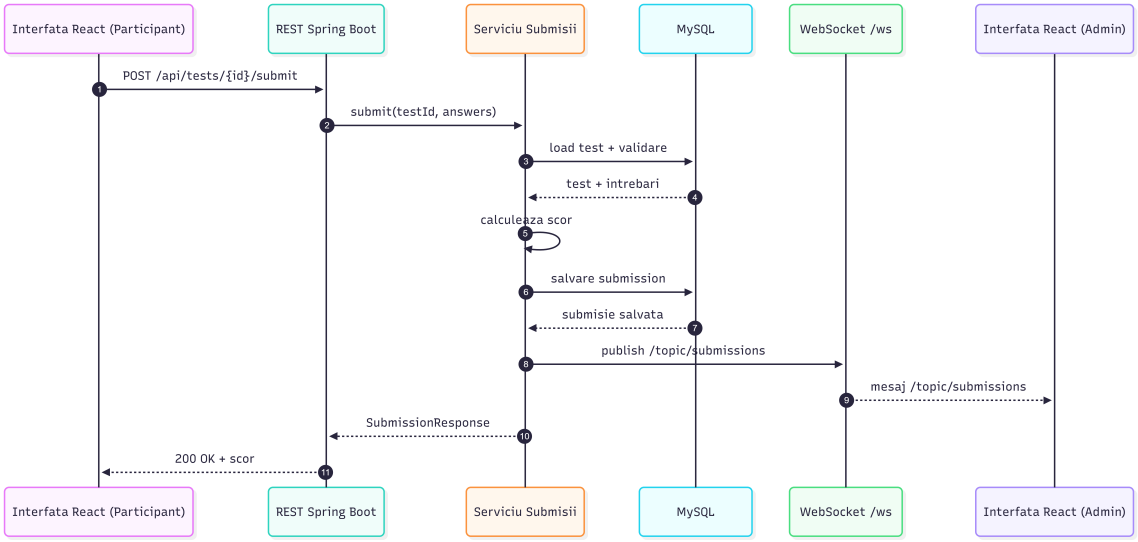

Figura 6: Flux submisie și persistare rezultate

Surse: `backend/src/main/java/com/example/testing/repo/TestSubmissionRepository.java`, `backend/src/main/java/com/example/testing/service/SubmissionService.java`

3.5 Gestionarea erorilor și logging

Aplicația folosește mecanismele implicite Spring Boot pentru tratarea erorilor. Mesajele explicite și statusurile HTTP sunt produse prin `ResponseStatusException`. Logging-ul este standard, la nivel de framework, fără audit logging dedicat.

Surse: `backend/src/main/java/com/example/testing/service/TestService.java`

3.6 Teste backend

Nu există teste unitare sau de integrare în repository. Pentru completare, se recomandă teste pe servicii și integrare REST pentru fluxurile critice.

4 Implementare Frontend (detaliat)

4.1 Setup proiect

Frontend-ul este realizat în React 18 cu Vite. Serverul de dezvoltare rulează pe portul 5173, iar endpoint-ul backend este configurat prin `VITE_API_URL` (implicit `http://localhost:8080`).

Surse: `frontend/package.json`, `frontend/src/api.js`

4.2 Arhitectură UI

Aplicația este single-page, fără routing explicit. Componenta `App` decide între ecranul de login și dashboard pe baza răspunsului `/api/auth/me`. Starea este gestionată local, iar evenimentele live sunt primite prin WebSocket.

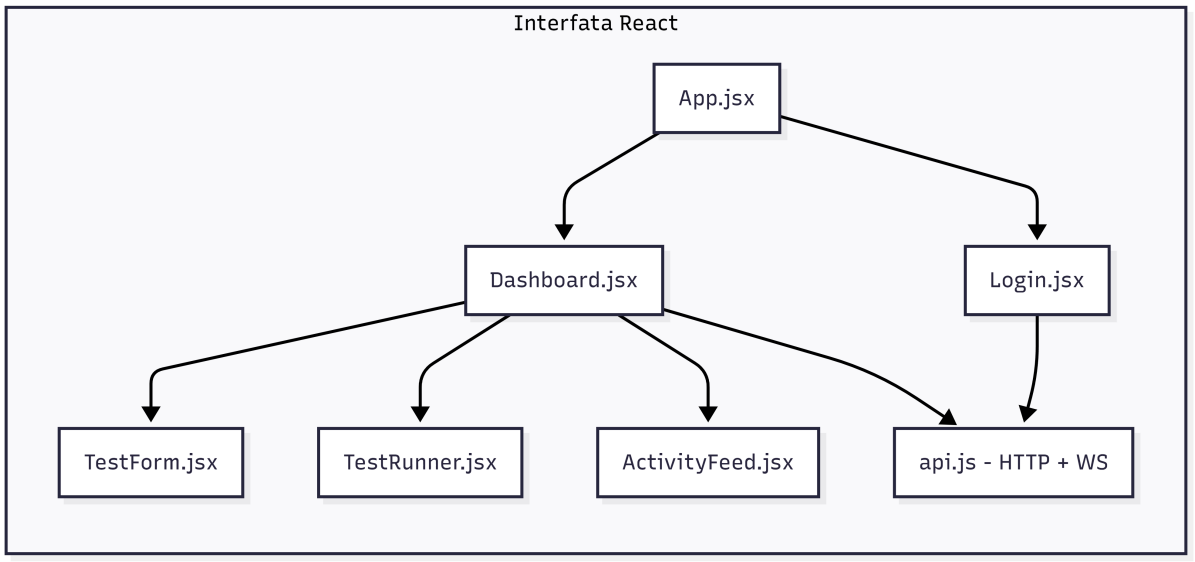

Figura 7: Arhitectura UI

Surse: `frontend/src/App.jsx`, `frontend/src/Dashboard.jsx`

4.3 Pagini și flow-uri principale

Interfața are patru zone principale: login/înregistrare, dashboard-ul cu listă de teste, formularul de creare/editare test și ecranul de rulare al testului. În plus, există un feed live pentru activitate. Adminul vede și gestionează submisiile, în timp ce participantul are un flux simplu de rezolvare.

Surse: `frontend/src/Login.jsx`, `frontend/src/Dashboard.jsx`, `frontend/src/TestForm.jsx`, `frontend/src/TestRunner.jsx`

4.4 Integrarea cu API

Comunicarea HTTP se face cu `fetch`, folosind `credentials: include` pentru sesiune. La fiecare request mutator se obține un token CSRF din `/api/auth/csrf`. Pentru fluxul live, UI se conectează prin STOMP/SockJS la `/ws` și se abonează la `/topic/tests` și `/topic/submissions`.

Surse: `frontend/src/api.js`

4.5 Validări și UX

Formularele folosesc validări HTML simple (required, minLength) și mesaje explicite pentru erori. Se afișează stări de încărcare, confirmări la ștergeri și feedback imediat după trimiterea unui test.

Surse: `frontend/src/Login.jsx`, `frontend/src/TestRunner.jsx`, `frontend/src/Dashboard.jsx`

4.6 Styling și responsive

Stilurile sunt inline, cu un layout bazat pe grid și carduri. Designul este coerent pentru desktop, dar responsivitatea pe mobil este minimală și ar necesita media queries sau un layout adaptiv.

Surse: `frontend/src/Dashboard.jsx`

4.7 Teste frontend

Nu sunt prezente teste unitare, UI sau e2e.

5 Bug fixing și stabilizare

5.1 Metodologie

În repository nu există un jurnal de bug-uri. Pentru o documentație completă se recomandă păstrarea unei liste de issue-uri cu pași de reproducere, cauza rădăcină și verificări de regresie.

5.2 Bug-uri majore rezolvate (template tabel)

ID	Titlu	Severitate	Pași reproducere	Cauză	Soluție	Dov

5.3 Testare de regresie

Un checklist minim ar include: autentificare, CRUD teste, trimitere submisie, recepționare evenimente WebSocket și ștergere submisie.

6 Testare și validare

6.1 Strategia de testare

În acest moment nu există teste automate. Strategia recomandată include teste unitare pe servicii, teste de integrare pentru REST și teste e2e pentru fluxurile principale.

6.2 Plan de test (scurt)

Scenarii cheie: înregistrare, login, creare test, editare test, submit, submit duplicat, listare submisii, și verificarea notificărilor live.

6.3 Rezultate

Nu există raport de testare. Principala limitare este lipsa automatizării.

7 Deploy, configurare și rulare

7.1 Cerințe de sistem

Pentru rulare locală sunt necesare Docker + Docker Compose sau alternativ Java 21, Node 20 și MySQL. Porturi expuse: 8080 (API), 5173 (UI), 3306 (DB).

7.2 Rulare locală

Metoda recomandată este Docker Compose:

```
docker compose up --build
```

7.3 Docker

Sunt definite trei servicii: `mysql`, `backend` și `frontend`, conectate pe aceeași rețea. Backend-ul rulează JAR-ul generat, iar frontend-ul rulează serverul Vite în mod dev.

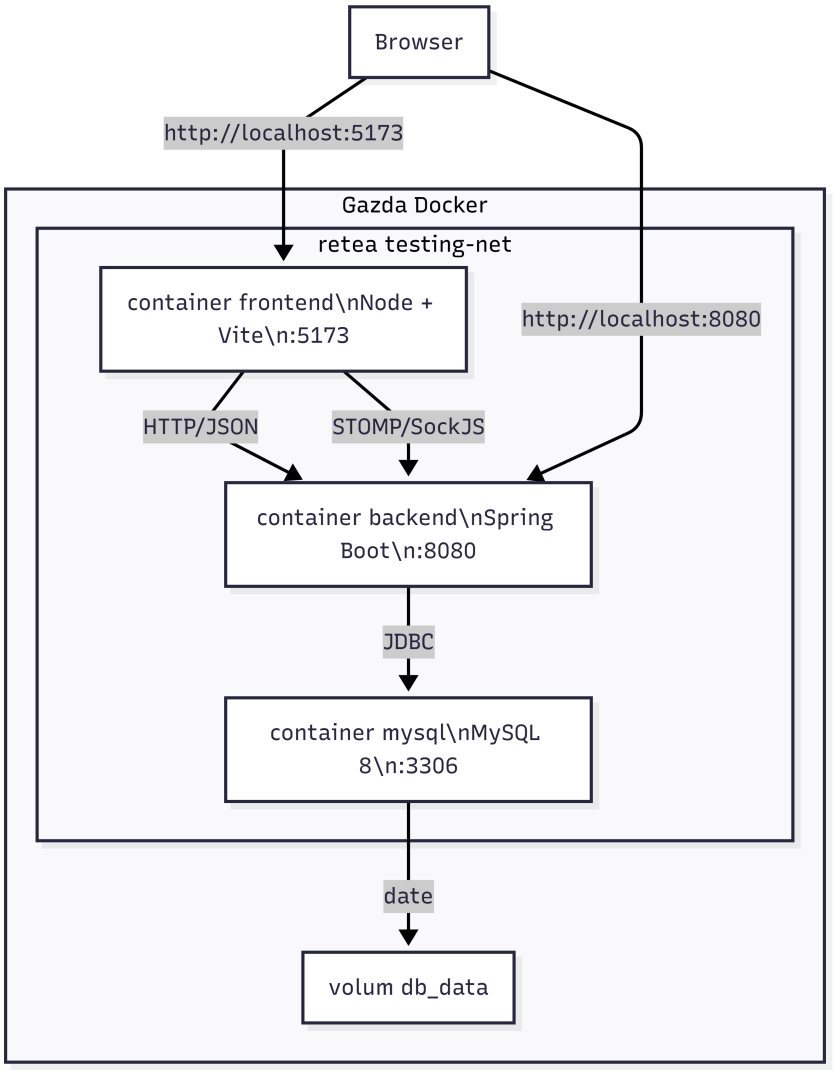

Figura 8: Topologia containerelor

Surse: `docker-compose.yml`, `backend/Dockerfile`, `frontend/Dockerfile`

7.4 Deploy în producție

Nu există CI/CD în proiect. Pentru producție ar fi necesare un build static pentru frontend, un reverse proxy cu HTTPS, precum și o strategie minimă de backup și rollback pentru DB.

8 Utilizare aplicație (ghid scurt)

Aplicația are două roluri: Administrator și Participant. Adminul poate crea/edita/șterge teste și poate vizualiza sau șterge submisiile. Participantul poate susține testele și primește scorul imediat. Pentru testare rapidă, există conturi seed: `admin/admin123` și `user/user123`.

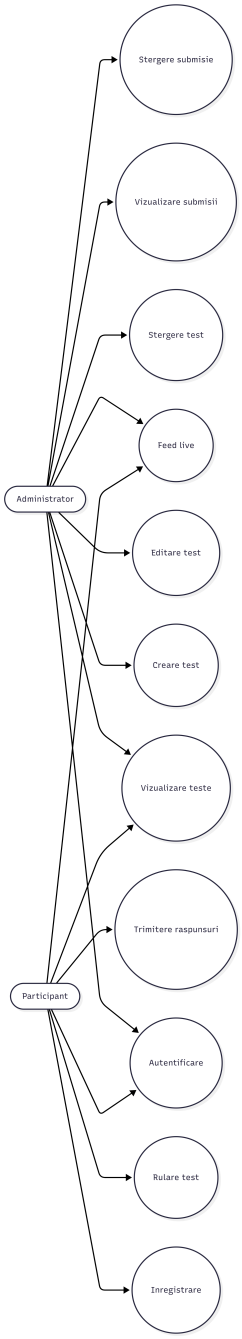

Figura 9: Roluri și acțiuni principale

Flux tipic: login → alegi un test → completezi răspunsurile → trimiți și vezi scorul; adminul vede și notificările live în feed.

Surse: `frontend/src/Dashboard.jsx`, `backend/src/main/java/com/example/testing/util/DataLoader`

9 Concluzii și direcții de dezvoltare

Proiectul livrează un sistem complet de testare online cu management de teste, submisii, scor automat, notificări live și securitate pe sesiune. Pașii firești de evoluție ar fi adăugarea JWT/OAuth, paginarea listelor, testare automată și îmbunătățirea responsivității UI.

10 Anexe

10.1 Diagrame (ERD)

```
users <-> users_roles <-> roles
tests -> questions -> answer_options
tests -> test_submissions -> submission_answers
```

10.2 Documentație API

Swagger UI: http://localhost:8080/swagger-ui.html

10.3 Structura repository-ului

```
online-testing-app/
  backend/
  frontend/
  docker-compose.yml
  README.txt
```

10.4 Liste endpoint-uri / payload-uri

Lista completă este în secțiunea REST API, iar contractele sunt definite în DTO-uri.

10.5 Bibliografie / resurse

- Spring Boot Docs
- Spring Security Docs
- React + Vite Docs