Documentatie Quiz

Alin Sima

October 2025

Contents

1 Introducere

Aplicația „Quiz Online” este un joc web de întrebări și răspunsuri cu componentă socială și runde scurte, orientat spre învățare și competiție prietenoasă între utilizatori. Platforma oferă mod single-player, lobby public sincronizat și dueluri 1v1, cu leaderboard global și istoric rezultate pentru motivare continuă.

1.1 Context general

Micro-învățarea prin quiz-uri rapide crește retenția și implicarea, iar formatul multiplayer live este o practică uzuală la aplicațiile de tip trivia socială. Arhitecturile de quiz live folosesc evenimente în timp real pentru broadcast întrebări, colectare răspunsuri, calcul scor și actualizare clasamente.

1.2 Obiective

Prin dezvoltarea acestei aplicații se urmărește atingerea următoarelor obiective:

- Livrarea unui joc quiz web responsive cu runde cronometrate și feedback instant privind scorul.
- Implementarea modurilor Single-player, Lobby public sincronizat și 1v1 cu sincronizare în timp real.
- CRUD complet pentru categorii, întrebări, chestionare, sesiuni și scoruri prin API REST.
- Integrare API extern de întrebări pentru varietate (Open Trivia DB / The Trivia API), combinat cu conținut creat de Admin.
- Leaderboard global și istoric sesiuni pentru utilizatori autentificați.
- Securitate pe bază de JWT și autorizare pe roluri (USER, ADMIN).

1.3 Lista MoSCoW

În cadrul procesului de dezvoltare a aplicației de testare online a cunoștințelor, este foarte important să stabilim prioritățile și să identificăm cerințele în funcție de importanța lor strategică. Metoda Moscow, o abordare de analiză a cerințelor, oferă o structură clară pentru clasificarea acestora în patru categorii distincte:

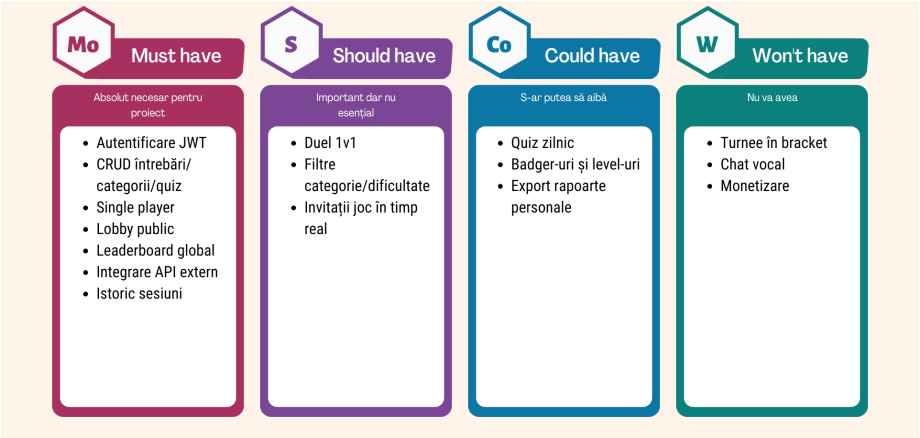

Figure 1: Schema MoSCoW

1.4 Cazuri de utilizare

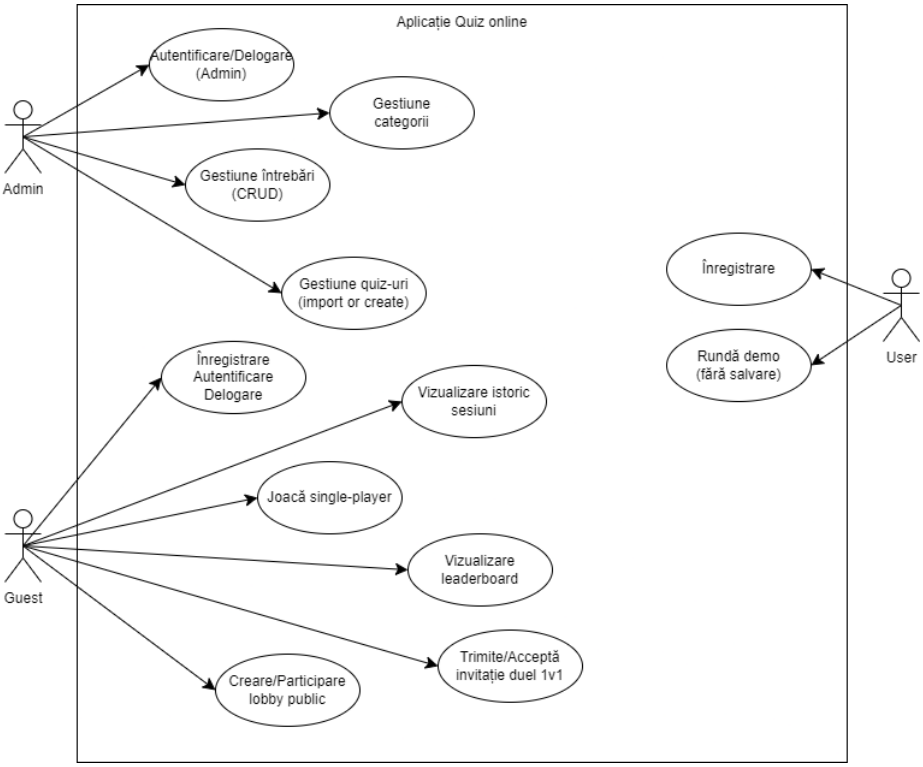

Figure 2: Cazuri de utilizare

2 Arhitectura și tehnologii

Arhitectura aplicației „Quiz Online” este proiectată pentru timp de răspuns redus, scalabilitate orizontală și o experiență coerentă între partea de client și serviciile de pe server. Interfața este implementată ca Single-Page Application în React, iar serviciile sunt livrate de un backend Spring Boot expus prin API REST și canale WebSocket pentru jocurile live, cu persistență relațională gestionată prin JPA/Hibernate.

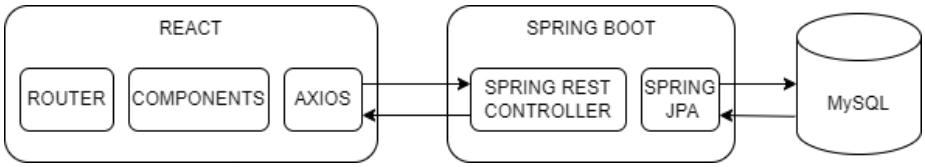

Figure 3: Arhitectura aplicației

2.1 Front-end React

React a fost ales pentru componentizarea clară a ecranelor (lobby, cameră de joc, cronometru, panou răspunsuri, leaderboard), ceea ce simplifică întreținerea și extinderea ulterioară. Interacțiunea cu backend-ul folosește cereri REST pentru operații CRUD și client STOMP peste WebSocket pentru broadcastul întrebărilor și al scorurilor, oferind o experiență în timp real în lobby și 1v1.

2.2 Back-End Springboot

Backend-ul urmează o arhitectură pe straturi cu Controller-e REST, servicii de domeniu și repository-uri JPA, fiecare responsabil pentru validare, logică de joc și acces la date, respectiv, ceea ce facilitează testarea și înlocuirea componentelor prin Dependency Injection. Spring Security 6 gestionează autentificarea și autorizarea, iar JWT oferă un model stateless potrivit pentru aplicații interactive și conexiuni WebSocket asociate utilizatorului.

2.3 Comunicare între front-end și back-end

Comunicarea dintre interfața React și serviciile Spring Boot se face pe două canale complementare: cereri HTTP/REST prin clientul Axios pentru operațiile de administrare și persistență (autentificare, CRUD categorii/întrebări/quiz, rezultate), respectiv STOMP peste WebSocket pentru evenimentele de joc în timp real (lobby public, duel 1v1, countdown, broadcast întrebări și scoruri). Această abordare asincronă reduce latența percepută, optimizează încărcarea inițială și menține o experiență fluidă în runde cronometrate.

2.4 Baza de date MySQL

Persistența este asigurată de o bază de date relațională MySQL cu entități precum Users, Categories, Questions, Quizzes, Rooms, Sessions, Submissions și Scores, mapate prin JPA/Hibernate. Indicii pe câmpuri frecvent filtrate (ex. category, difficulty, created_at) optimizează selecția întrebărilor și interogările de leaderboard, iar tranzacțiile ACID garantează consistența la finalul rundelor.

2.5 Beneficii ale acestei arhitecturii

- Scalabilitate: separarea clară UI/servicii/date și folosirea pub/sub pe WebSocket permit adăugarea de instanțe fără a afecta experiența jucătorilor.
- Eficiență în dezvoltare: combinarea React pentru SPA și Spring Boot pentru REST/WebSocket reduce boilerplate-ul și accelerează livrarea funcționalităților.
- Performanță: REST pentru CRUD și STOMP pentru evenimente minimizează traficul redundant și asigură sincronizare rapidă în runde cronometrate.
- Securitate: JWT și politicile de autorizare pe roluri protejează resursele și asociază identitatea la nivel de conexiune în timp real.
- Extensibilitate: modelul pe componente și straturi permite adăugarea ulterioară a modurilor de joc, badge-uri sau rapoarte fără refactorizări majore.

3 Implementare front-end

Interfața „Quiz Online" este implementată ca aplicație Single-Page în React, construită cu Vite pentru timp minim de build și încărcări incrementalizate. Componentizarea clară a Dashboard-ului, camerelor de joc și spațiilor administrative reduce complexitatea și permite reutilizarea logicii de stare în multiple scenarii (single-player, lobby, duel 1v1).

3.1 Tehnologii front-end

- **Vite + React 18**: bootstrap rapid, HMR și optimizări ESBuild pentru importuri dinamice (*App.jsx*, *main.jsx*).
- **React Router DOM**: gardă de rute private vs. publice, layout-uri separate pentru auth și dashboard (*AppLayout*, *AuthLayout*).
- **Context API + Reduceri**: *AuthContext* memorează token-ul JWT, rolul (USER/ADMIN) și preferințele UI, expunând hook-ul `useAuth()` către restul componentelor.

- **Axios Services**: *authService*, *categoryService*, *questionService* și *api.js* gestionează clienții REST, includ interceptor pentru token și tratează retry la erori 401.
- **UI Layer**: stilizare custom + utilitare CSS, iconografie minimalistă și animații scurte pentru butoanele de acțiune rapidă (*QuickActionsButtons*).

3.2 Implementare front-end

SPA-ul este orchestrată din `App.jsx`, unde rutele publice (`/login`, `/register`) sunt separate de cele private prin `PrivateRoute`. Toate ecranele protejate sunt împachetate de `AppLayout` pentru a păstra bara de navigaţie, acţiunile rapide şi contextul Auth.

3.2.1 Structura componentelor şi rutele aferente

Table 1: Maparea componentelor React pe rutele efective

Componentă	**Rol funcţional**	**Rută asociată**
App	Punctul de intrare; configurează Router-ul, AuthProvider şi `PrivateRoute`.	toate
PrivateRoute	Gardă pentru rute private şi verificare roluri; injectează `AppLayout`.	reutilizată pe toate rutele protejate
AppLayout	Layout global cu header, butoane „Quick Actions” şi wrapper pentru conţinut.	toate rutele private
HomePage	Dashboard personal cu rezumat sesiuni, invitaţii la duel şi CTA „Porneşte un quiz”.	/home
Login	Formular de autentificare (JWT) cu feedback pentru erori şi redirect după logare.	/login
Register	Formular creare cont (rol implicit USER) şi validări de parolă.	/register
SoloSessionStart	Wizard de configurare pentru un quiz single-player (categorie, dificultate, număr întrebări).	/play/solo
PlaySession	UI-ul efectiv al sesiunii single-player; afişează întrebările şi cronometru per item.	/play/solo/:sessionId

Continuă pe pagina următoare...

Continuare de pe pagina anterioară		
Componentă	**Rol funcţional**	**Rută asociată**
LobbyRoom	Cameră publică sincronizată (join, ready, countdown, broadcast întrebări).	/play/lobby
DuelRoom	Interfaţă 1v1: accept invitaţii, transmite răspunsuri live, vede scoruri comparative.	/play/duel
LeaderboardPage	Clasamente globale şi filtrate plus statistici personale.	/leaderboard
AdminDashboard	KPIs, carduri de stare şi scurtături pentru fluxurile administrative.	/admin
QuestionList	Tabel administrativ cu filtre, căutare şi acţiuni bulk pentru întrebări.	/admin/questions
QuestionDetails	Vizualizează întrebarea + metadate şi permite dezactivarea/editarea.	/admin/questions/:id
QuestionManager	Formulare pentru creare/editare întrebare (validări live, preview variantă).	integrat în /admin/questions
LeaderboardWidget	Secţiune reutilizabilă cu top rapid pentru dashboard şi Home.	componente interne
NotificationBanner	Afişează invitaţii la duel şi mesaje în timp real prin contextul Auth.	inclus în HomePage şi AppLayout
SessionHistoryTable	Tabel cu ultimele 5 sesiuni, link către reluarea detaliilor.	parte din HomePage

3.3 Utilizarea aplicației din perspectiva utilizatorului

Autentificare și creare cont Primul contact cu aplicația este ecranul de login, cu feedback imediat pentru credențiale greșite. Butonul „Creează cont” duce la formularul de înregistrare (username, email, parolă confirmată).

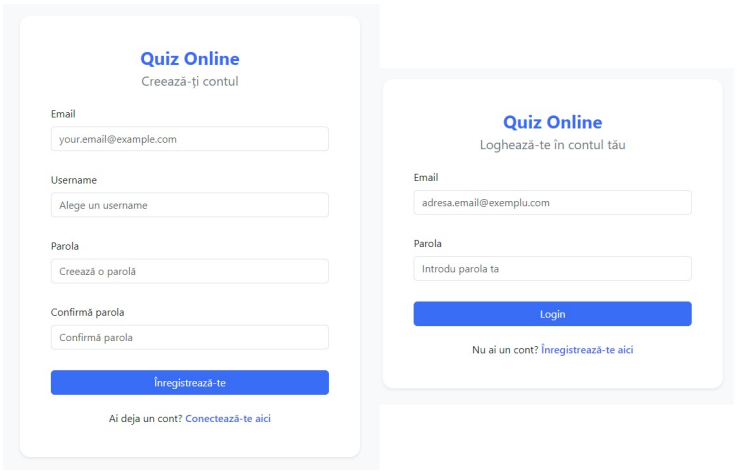

Figure 4: Formular de autentificare/logare

Home personalizat Cardurile din prima secțiune afișează sumarul ultimelor sesiuni, invitațiile la duel și butonul principal „Pornește un quiz acum”.

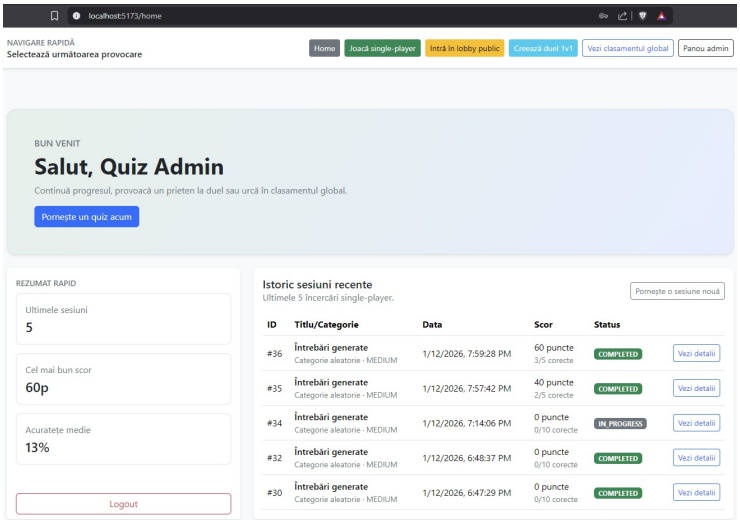

Figure 5: Dashboard după autentificare

Configurare solo Utilizatorul alege categoria, dificultatea, numărul de întrebări și timpul per item înainte de a porni sesiunea single-player.

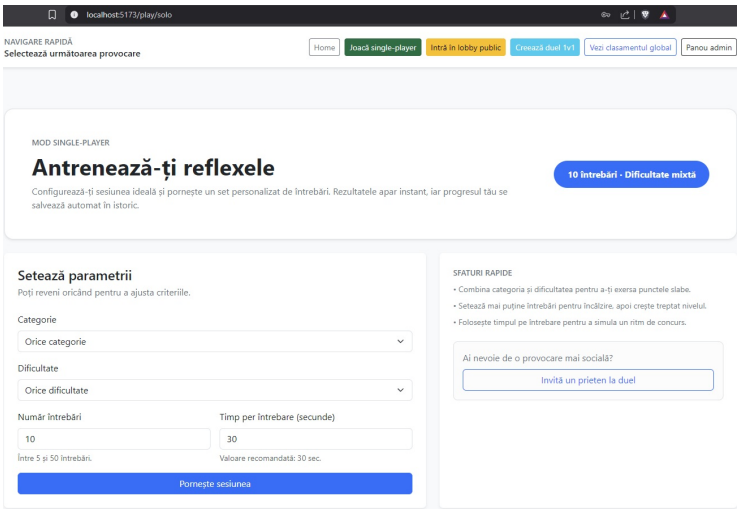

Figure 6: Setarea parametrilor pentru mod single-player

Desfășurarea quiz-ului Ecranul de joc listează întrebarea curentă, progresul, scorul și un navigator rapid între itemi. După trimiterea răspunsului, sistemul evidențiază corectul și actualizează scorul.

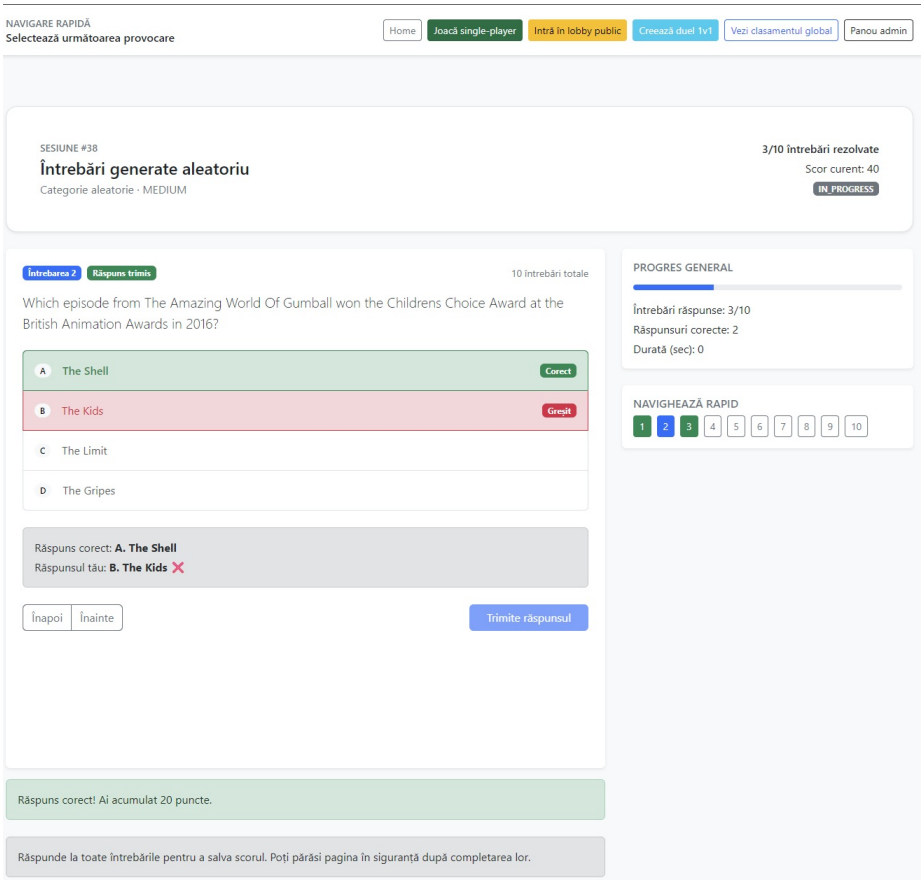

Figure 7: Interfața de lucru pentru o sesiune single-player

Lobby public În lobby-ul sincronizat, toți jucătorii primesc întrebările simultan, văd numărătoarea inversă și clasamentul live.

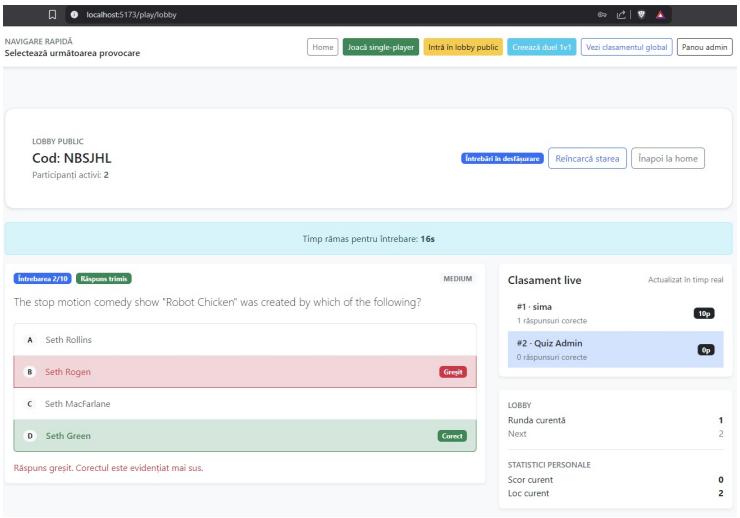

Figure 8: Lobby public cu countdown și leaderboard

Duel 1v1 Secțiunea duel permite crearea unei camere cu cod, trimiterea de invitații și derularea rundei 1v1 cu feedback în timp real.

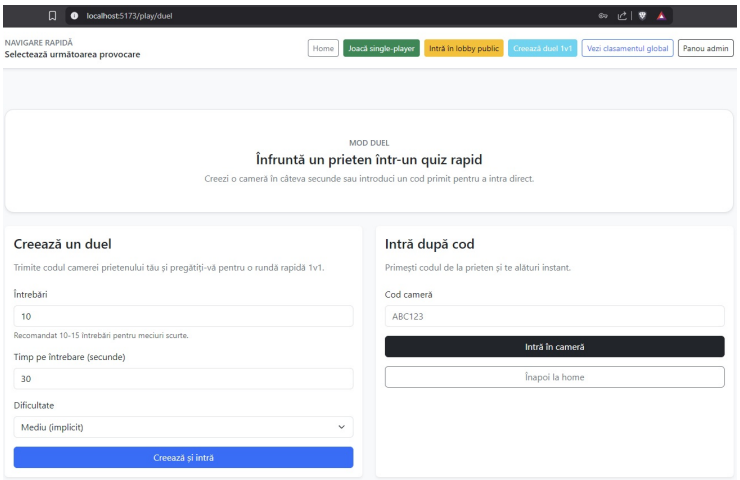

Figure 9: Interfață duel: configurare

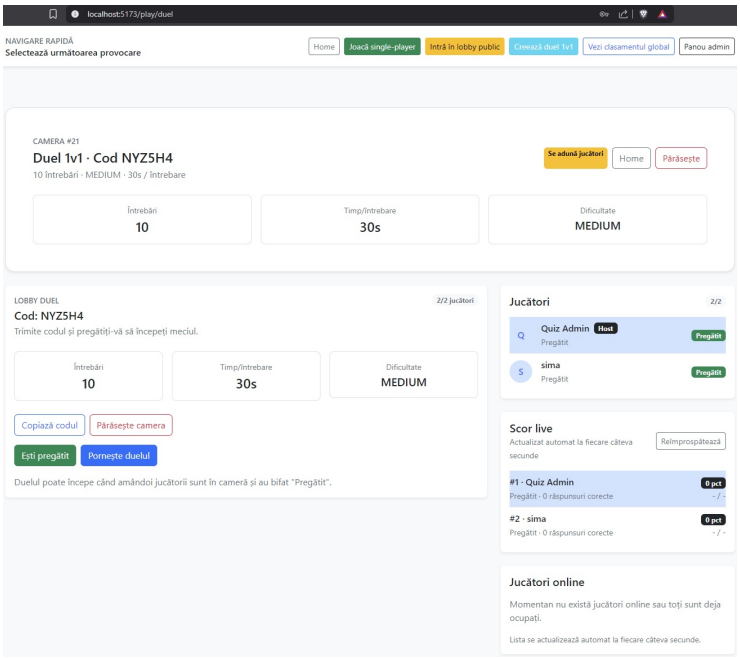

Figure 10: Interfață duel: configurare

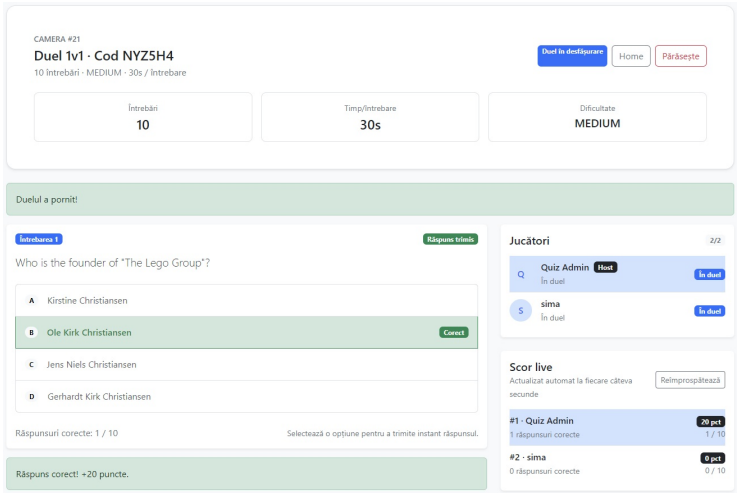

Figure 11: Interfață duel: desfășurare

Leaderboard Leaderboard global și diferite statistici personale.

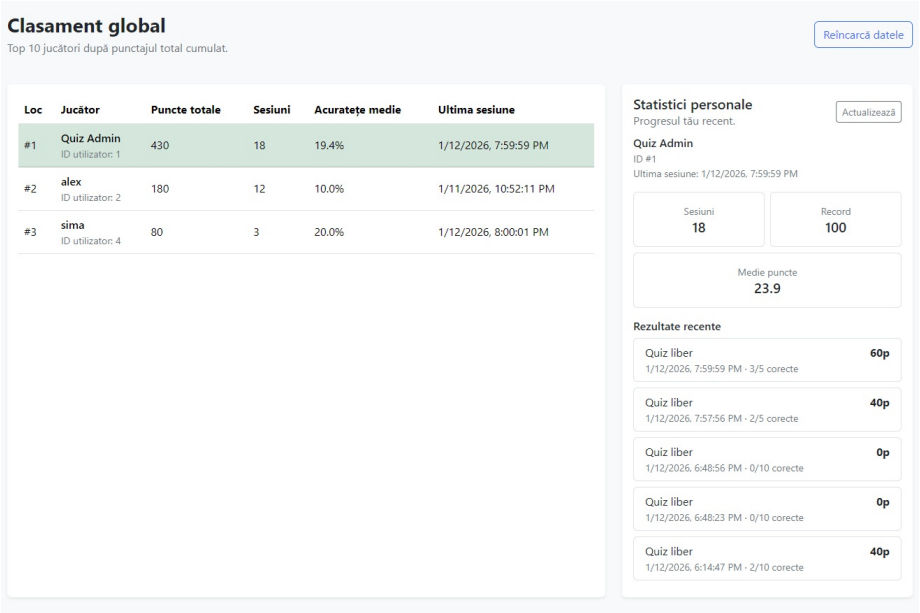

Figure 12: Rezumat scor și clasamente

4 Implementare backend

Backend-ul „Quiz Online” este construit pe Spring Boot şi expune API-uri REST/stateless pentru jocurile single-player, lobby public şi dueluri, completate de canale WebSocket pentru broadcast în timp real. Lag-ul redus este obţinut printr-o arhitectură pe straturi (Controller → Service → Repository) şi prin folosirea tranzacţiilor declarative.

4.1 Tehnologii backend

4.1.1 Spring Boot

Aplicaţia porneşte din `DemoApplication` şi foloseşte auto-configurarea Spring Boot pentru HTTP, securitate şi WebSocket. Controller-ele din `com.example.demo.resource` gestionează doar transportul, delegând logica domeniului către servicii.

4.1.2 Spring Data JPA

Entităţile sunt mapate direct pe tabele MySQL. Exemplu: modelul întrebărilor din `Question` gestionează opţiunile, dificultatea şi starea de activare.

Listing 1: Entitatea Question

```
@Entity
@Getter
```

```
@Setter
@NoArgsConstructor
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@Table(name = "questions")
public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, updatable = false, unique =
        true)
    private Long id;

    @Column(name = "text", nullable = false, length = 1000)
    private String text;

    @Column(name = "option_a", nullable = false)
    private String optionA;

    @Column(name = "option_b", nullable = false)
    private String optionB;

    @Column(name = "option_c", nullable = false)
    private String optionC;

    @Column(name = "option_d", nullable = false)
    private String optionD;

    @Column(name = "correct_option", nullable = false)
    private Integer correctOption;

    @Enumerated(EnumType.STRING)
    @Column(name = "difficulty_level", nullable = false)
    private DifficultyLevel difficultyLevel;

    @Enumerated(EnumType.STRING)
    @Column(name = "source_type", nullable = false)
    private SourceType sourceType;

    @Column(name = "explanation", length = 2000)
    private String explanation;

    @Column(name = "points")
    private Integer points;

    @Column(name = "is_active", nullable = false)
    private Boolean isActive = true;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;
```

```
    @Column(name = "updated_at", nullable = false)
    private LocalDateTime updatedAt;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "category_id", nullable = false)
    private Category category;

    @PrePersist
    private void prePersist() {
        LocalDateTime now = LocalDateTime.now();
        createdAt = now;
        updatedAt = now;
        if (isActive == null) {
            isActive = true;
        }
    }

    @PreUpdate
    private void preUpdate() {
        updatedAt = LocalDateTime.now();
    }
}
```

Repository-ul JPA aferent din `QuestionRepo` oferă metode generoase pentru filtrare şi selecţii randomizate:

Listing 2: Repository cu filtre dinamice

```
@Repository
public interface QuestionRepo extends JpaRepository<Question, Long> {
    List<Question> findByCategoryId(Long categoryId);
    List<Question> findByDifficultyLevel(DifficultyLevel difficulty);
    List<Question> findByCategoryIdAndDifficultyLevel(Long categoryId,
        DifficultyLevel difficultyLevel);
    List<Question>
        findByCategoryIdAndDifficultyLevelAndIsActiveTrue(Long
        categoryId, DifficultyLevel difficultyLevel);
    long countByCategoryId(Long categoryId);
    @Query("""
           SELECT q FROM Question q
           WHERE (:categoryId IS NULL OR q.category.id = :categoryId)
             AND (:difficulty IS NULL OR q.difficultyLevel = :difficulty)
             AND q.isActive = true
           ORDER BY function('RAND')
           """)
    List<Question> findRandomActiveQuestions(
           @Param("categoryId") Long categoryId,
           @Param("difficulty") DifficultyLevel difficulty,
           Pageable pageable);
```

```
    @Query(value = """
            SELECT q FROM Question q
            WHERE (:categoryId IS NULL OR q.category.id = :categoryId)
              AND (:difficulty IS NULL OR q.difficultyLevel = :difficulty)
              AND (:sourceType IS NULL OR q.sourceType = :sourceType)
              AND (:activeOnly = false OR q.isActive = true)
            """,
           countQuery = """
            SELECT COUNT(q) FROM Question q
            WHERE (:categoryId IS NULL OR q.category.id = :categoryId)
              AND (:difficulty IS NULL OR q.difficultyLevel = :difficulty)
              AND (:sourceType IS NULL OR q.sourceType = :sourceType)
              AND (:activeOnly = false OR q.isActive = true)
            """)
    Page<Question> findByFilters(
            @Param("categoryId") Long categoryId,
            @Param("difficulty") DifficultyLevel difficulty,
            @Param("sourceType") SourceType sourceType,
            @Param("activeOnly") boolean activeOnly,
            Pageable pageable);
}
```

4.1.3 Dependency Injection

Serviciile sunt annotate cu `@Service` şi primesc dependenţele în constructor prin `@RequiredArgsConstructor`. `QuestionService` orchestrează validările, maparea DTO-urilor şi tranzacţiile declarative:

Listing 3: Serviciu cu tranzacții declarative

```
@Service
@RequiredArgsConstructor
@Transactional
public class QuestionService {

   private final QuestionRepo questionRepo;
   private final CategoryRepo categoryRepo;

   @Transactional(readOnly = true)
   public QuestionResponse getQuestion(Long id) {...

   @Transactional(readOnly = true)
   public List<QuestionResponse> getQuestions(...

   @Transactional(readOnly = true)
   public PagedResponse<QuestionResponse> getQuestionsPage(...

   public QuestionResponse createQuestion(QuestionRequest request) {...
```

```
    public QuestionResponse updateQuestion(Long id, QuestionRequest
        request) {...

    public void deleteQuestion(Long id) {...

    @Transactional(readOnly = true)
    public List<QuestionResponse> getRandomQuestions(Long categoryId,
        DifficultyLevel difficultyLevel, int count) {...

    private void mapRequestToEntity(QuestionRequest request, Question
        question) {...

    private int resolvePoints(DifficultyLevel difficultyLevel, Integer
        customPoints) {...

    private Category getCategory(Long categoryId) {...

    public Question getQuestionEntity(Long id) {...
}
```

4.1.4 Spring Security + JWT

Securitatea stateless este configurată în `SecurityConfig`, unde:

- se dezactivează CSRF şi se setează `SessionCreationPolicy.STATELESS`;
- se expune CORS pentru originile React (5173/3000);
- se delimitează rutele publice (`/api/auth/**`, `/ws/**`) de cele protejate (întrebări, categorii, lobby, duel etc.);
- se inserează `JwtAuthenticationFilter` înainte de `UsernamePasswordAuthenticationFilter`.

Filtrul JWT extrage token-ul din header, validează semnătura şi setează contextul de securitate astfel încât controller-ele să poată folosi `Principal` sau `Authentication`.

WebSockets `WebSocketConfig` activează STOMP la endpoint-ul `/ws` şi defineşte prefixele `/app` (mesaje de la client) şi `/topic` (broadcast către clienţi). Serviciile de lobby şi duel publică evenimente (COUNTDOWN, QUESTION, SCORE UPDATE) către `/topic/lobby.{code}` sau `/topic/duel.{code}`, menţinând sincronizarea dintre jucători fără pooling agresiv.

Baza de date MySQL Datele persistă într-o bază MySQL: tabelele `users`, `questions`, `categories`, `sessions`, `rooms`, `submissions` etc. Indicii pe `category_id`, `difficulty_level` şi `created_at` optimizează filtrările din ecranele admin şi din selecţiile random pentru joc.

Maven Proiectul folosește `mvnw` pentru a garanta versiunea de Maven necesară. `pom.xml` declară dependenţele (Spring Boot starters, Lombok, MySQL driver) şi plugin-urile de build/test.

4.2 Gestionarea cererilor HTTP

API-ul REST urmează convenţii clare (`/api/auth`, `/api/questions`, `/api/sessions`, `/api/lobby`, `/api/duels` etc.). Controller-ele validează request-urile (`@Valid`) şi se bazează pe DTO-uri pentru a izola entităţile.

Endpoint	**Metodă**	**Descriere**
/api/auth/register	POST	Creează un cont şi returnează token JWT.
/api/auth/login	POST	Autentifică utilizatorul şi emite token-urile.
/api/categories	GET	Listează categoriile active (parametrul `activeOnly`).
/api/categories/opentdb/sync	POST	Sincronizează cu Open Trivia DB.
/api/questions	GET	Filtrare după categorie/dificultate/sursă, selecţie random.
/api/questions	POST	Creează o întrebare nouă (doar ADMIN).
/api/questions/{id}	PUT/ DELETE	Actualizează sau dezactivează întrebarea.
/api/questions/import/opentdb	POST	Importează întrebări externe.
/api/lobby	GET	Starea lobby-ului public pentru user-ul curent.
/api/lobby/join, /leave, /answer	POST	Înscriere, ieşire şi trimitere răspuns în lobby.
/api/rooms (şi /join,/ready,/start)	POST/ GET	Gestionarea camerelor private cu cod.
/api/sessions/solo	POST	Creează sesiune single-player.
/api/sessions/{id}/submit	POST	Înregistrează răspunsul curentului.
/api/duels/{code}	GET/ POST	Status duel + trimitere răspuns.
/api/leaderboards	GET	Clasamente globale sau pe quiz.
/api/presence/	GET/ POST/ DELETE	Heartbeat şi listă utilizatori online.

Table 2: Endpoint-urile principale ale backend-ului

Prin această structură, backend-ul asigură:

- **scalabilitate**: stateless JWT + WebSocket topic-based;
- **siguranţă**: reguli stricte de autorizare în `SecurityConfig`;
- **extensibilitate**: controller-ele sunt subţiri, iar logica rămâne în servicii testabile;

- **timp de răspuns mic**: interogări optimizate, paginare şi selecţii random din baza de date.

5 Concluzii

Platforma „Quiz Online" livrată în acest proiect demonstrează că un ecosistem React + Spring Boot poate susţine atât scenarii individuale, cât şi experienţe multiplayer sincronizate în timp real. Arhitectura pe straturi, împreună cu autentificarea pe bază de JWT şi canalele STOMP, asigură o bază stabilă pe care se pot itera rapid noi funcţionalităţi.

Perspectivele de extindere rămân generoase: introducerea întrebărilor multimedia (imagini, audio, video), suport pentru răspunsuri multiple sau power-up-uri de tip gamification în modul duel, precum şi raportări avansate pentru performanţa jucătorilor şi a claselor. Mai departe, analizele bazate pe AI pentru generarea întrebărilor şi recomandările personalizate ar putea transforma aplicaţia într-un mentor digital adaptiv.

În formă actuală, integrarea dintre front-end-ul React (SPA cu layout unificat şi componente reutilizabile) şi backend-ul Spring Boot (REST, WebSocket, JPA) oferă administratorilor instrumente complete de creare şi moderare, iar utilizatorilor o experienţă fluidă: sesiuni single-player configurabile, lobby public cu leaderboard live şi dueluri 1v1 pe cod. Persistenţa MySQL, tranzacţiile declarative şi validările riguroase garantează consistenţa datelor, în timp ce WebSocket-urile menţin dinamismul jocului.

Prin urmare, aplicaţia îşi atinge obiectivul iniţial de a moderniza procesul de testare, dar, mai important, rămâne deschisă pentru evoluţie: fiecare modul a fost construit cu extensibilitatea în minte, permiţând alinierea viitoare la trenduri educaţionale şi tehnologice fără refactorizări costisitoare.

Bibliografie

1. Spring Boot Documentation, versiunea 3.x, *https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/*
2. Spring Security Reference, *https://docs.spring.io/spring-security/reference/*
3. React Documentation (React 18), *https://react.dev/learn*
4. Vite Guide, *https://vitejs.dev/guide/*
5. STOMP over WebSocket Guide, *https://stomp.github.io/stomp-specification-1.2.html*
6. Open Trivia DB API, *https://opentdb.com/api*$_c onfig.php$
6. *MySQL 8.0 Reference Manual, https://dev.mysql.com/doc/refman/8.0/en/*