Proiect final

Management chestionare folosind Spring + Hibernate

Diaconu Horia, Filipas Razvan, Moldovan Bogdana

Grupa 244, Sisteme Distribuite in Internet

Scopul actual al aplicației este gestionarea chestionarelor. Există două tipuri de utilizatori: studenți și profesori. Un profesor poate adăuga întrebări și genera teste pe baza întrebărilor adăugate. Un student poate vizualiza chestionarele disponibile, rezolva un test și consulta un istoric al rezolvărilor și punctajelor obținute. Până în prezent, interfața utilizatorului (frontend-ul) este dezvoltată exclusiv pentru studenți.

Entitatile prezente in aceasta aplicatie sunt:

Clasa User – pentru Utilizator

- Id: Long – reprezinta identificatorul unic al clasei utilizator
- fistName: String – reprezinta prenumele utilizatorului
- lastName: String – reprezinta numele utilizatorului
- username: String – reprezinta username-ul utilizatorului
- password: String – reprezinta parola utilizatorului
- role: String – reprezinta rolul utilizatorului

Clasa Quiz – pentru Chestionar

- Id: Long – reprezinta identificatorul unic al clasei chestionar
- title: String – reprezinta titlul chestionarului

Clasa Question – pentru Intrebare

- Id: Long – reprezinta identificatorul unic al clasei intrebari
- text: String – reprezinta textul intrebari
- choiceA: String – reprezinta varianta de raspuns A a intrebarii
- choiceB: String – reprezinta varianta de raspuns B a intrebarii
- choiceC: String – reprezinta varianta de raspuns C a intrebarii
- correctChoice – reprezinta varianta corecta de raspuns a intrebarii

Clasa Attempt – pentru Rezolvarea unui chestionar de catre un utilizator

- Id: Long – reprezinta identificatorul unic al clasei Attempt
- userId: Long – reprezinta ID-ul utilizatorului care a rezolvat chestionarul
- quizId: Long – reprezinta ID-ul chestionarului care a fost rezolvat de catre utilizator
- score: int – reprezinta punctul obtinut de catre utilizator in urma rezolvari chestionarului

Clasa QuestionQuiz – pentru legatura dintre intrebari si chestionar

- Id: Long – reprezinta identificatorul unic al clasei QuestionQuiz
- questionId – reprezinta ID-ul intrebari
- quizId – reprezinta ID-ul chestionarului

Aplicatia dispune de urmatoarele functionalitati:

- Utilizatorul student poate vizualiza chestionarele disponibile
- Utilizatorul student poate deschide si rezolva un chestionar
- Utilizatorul student poate vizualiza pagina cu rezolvarile sale anterioare ale chestionarelor
- Utilizatorul profesor poate vizualiza chestionarele disponibile
- Utilizatorul profesor poate vizualiza intrebarile disponibile
- Utilizatorul profesor poate adauga intrebari
- Utilizatorul profesor poate adauga chestionare pe baza listei de intrebari disponibile

Intre entitatea Question si entitatea Quiz exista o relatie Many-To-Many prin intermediul entitatii QuestionQuiz. Acesta relatie se transmite in felul urmator: O intrebare poate sa fie prezenta in mai multe chestionare diferite, iar intr-un chestionar se folosesc mai multe intrebari.

Pentru implementarea acestei aplicații, s-au utilizat Java, Hibernate și JpaRepository pentru gestionarea datelor în backend. Framework-ul Spring a fost folosit pentru dezvoltarea aplicației web, facilitând gestionarea dependențelor și gestionarea cererilor HTTP. Pe partea de frontend, s-a optat pentru Angular pentru a crea o interfață de utilizator modernă și interactivă. Baza de date a fost implementată folosind SQLite, oferind o soluție ușor de utilizat și eficientă pentru stocarea datelor. Combinația acestor tehnologii a permis dezvoltarea unei aplicații robuste și scalabile, care satisface cerințele atât ale profesorilor, cât și ale studenților.

Pentru fiecare entitate prezentă în aplicație, s-au creat și entități DTO (Data Transfer Object). Aceste entități DTO sunt utilizate pentru a transmite datele între frontend și backend, facilitând comunicarea eficientă între aceste două părți ale aplicației. Prin utilizarea entităților DTO, se asigură o separare clară între modelul de date folosit intern în backend și datele care sunt transmise către interfața de utilizator. Astfel, se optimizează performanța și se reduce traficul de date între server și client, contribuind la îmbunătățirea performanței și scalabilității aplicației.

Interfețelele pentru repository definesc contracte care abstractizează operațiile de bază efectuate asupra entităților persistente din baza de date. Aceste interfețe extind JpaRepository sau alte interfețe specifice Spring Data JPA și includ metode pentru a găsi, salva, actualiza și șterge entitățile. Această abordare permite o izolare eficientă a logicii de acces la date și promovează o arhitectură ușor de întreținut, în care implementările concrete pot fi schimbate fără a afecta codul consumator.

Interfețele pentru service definesc operațiile logice care pot fi efectuate asupra entităților și gestionează interacțiunile între diferitele componente ale aplicației. Aceste interfețe abstractizează operațiile de bază și oferă o abordare flexibilă pentru implementarea logicii în diverse moduri. Utilizarea interfețelor pentru service încurajează o separare clară între nivelul de acces la date și logica, acestora,

facilitând testarea izolarea componentelor. De asemenea, promovează principiul de inversare a dependențelor, în care componentele aplicației depind de abstracții (interfețe), nu de implementări concrete, ceea ce face codul mai modular și mai ușor de întreținut.

Configurarea Hibernate:

Fisierul hibernate.cfg.xml reprezintă fișierul de configurare pentru Hibernate, un framework de persistență în Java. În cadrul acestui fișier, sunt specificate setările necesare pentru conectarea la o bază de date SQLite și pentru configurarea sesiunilor Hibernate.

Fișierul începe cu declarații pentru versiunea XML și DTD-ul (Document Type Definition) pentru configurarea Hibernate.

În cadrul elementului `<hibernate-configuration>`, se află `<session-factory>`, care este locul unde sunt specificate proprietățile de configurare a sesiunii Hibernate.

Proprietățile incluse în cadrul `<session-factory>` sunt:

- `<property name="dialect">`: Specifică dialectul SQL utilizat de Hibernate, în acest caz `SQLiteDialect`.

- `<property name="connection.driver_class">`: Specifică clasa driver-ului JDBC pentru conexiunea la baza de date SQLite.

- `<property name="connection.url">`: Specifică URL-ul de conexiune către baza de date SQLite.

- `<property name="connection.username">` și `<property name="connection.password">`: Specifică numele utilizatorului și parola pentru autentificarea la baza de date.

- `<property name="show_sql">` și `<property name="format_sql">`: Aceste proprietăți controlează afișarea SQL-ului generat de Hibernate în consolă și formatarea acestuia.

- `<property name="hbm2ddl.auto">`: Această proprietate controlează modul în care Hibernate va gestiona schema bazei de date. În acest caz, este setată la "update",

ceea ce înseamnă că Hibernate va încerca să actualizeze schema bazei de date pentru a corespunde cu entitățile Hibernate.

De asemenea, în cadrul `<session-factory>` este inclus și un element `<mapping>`, care specifică locația fișierului de mapare a clasei `User`, `User.hbm.xml`.

In acest fișier de configurare sunt definiti parametrii de conectare și comportamentul general al sesiunilor Hibernate, permițându-ne sa integram eficient Hibernate cu o bază de date SQLite în cadrul aplicației noastre.

De asemenea, s-au creat fișiere de mapare pentru entitatile specifice în cadrul framework-ului Hibernate. În cadrul acestor fișier, sunt definite detaliile de mapare între clasele Java și tabelele corespunzătoare din baza de date.

În cadrul elementului `<hibernate-mapping>`, se specifică pachetul în care se află entitatea, folosind atributul `package`.

În cadrul elementului `<class>`, se definesc detaliile de mapare pentru clasa respective. Atributele specifică numele clasei Java și numele tabelei din baza de date.

În cadrul elementului `<composite-id>`, este definită cheia compusă a entității, care este formată dintr-un singur atribut, `id`.

În cadrul elementelor `<property>`, sunt definite proprietățile entității și se specifică tipurile acestora și coloanele corespunzătoare din tabelul bazei de date. De exemplu, pentru entitatea User: proprietățile `firstName`, `lastName`, `username`, `password` și `role` sunt definite.

În general, pentru fiecare entitate din aplicație, s-a creat un astfel de fișier de mapare care specifică detaliile de mapare între clasele Java și tabelele din baza de date, astfel încât Hibernate să poată realiza operațiile de persistență în mod corespunzător. Această abordare asigură o separare clară între modelul de date al aplicației și schema bazei de date, facilitând dezvoltarea și întreținerea codului.

Endpoint-urile prezenta in aceasta aplicatie:

Pentru User Controller:

- /users/{username} [GET] (function GetUserByUsername()) – returneaza UserDTO-ul care are username-ul cautat

Pentru Quiz Controller:

- /quizes/ [GET] (function getAllQuizes()) – returneaza o lista cu toate QuizDTO-urile disponibile
- /quizzes/ [POST] (function addQuiz()) – permite adaugarea unui nou Quiz

Pentru Question Controller:

- /questions/ [GET] (function findAllQuestions()) – returneaza o lista cu toate QuestionDTO-urile disponibile
- /questions/ [POST] (function addQuestion()) – permite adaugarea unei noi intrebari in baza de date

Pentru QuestionQuiz Controller:

- /questionQuizzes/{quizId} [GET] (function findAllQuestionsByQuizId()) – returneaza o lista cu toate QuestionDTO-urile dintr-un anumit Quiz
- /questionQuizzes/ [POST] (function addQuestionQuiz()) – permite adaugarea unei Question la un anumit Quiz

Pentru Attempt Controller:

- /attempts/{userId} [GET] (function findAllAttemptsByUserId()) – returneaza o lista cu toate Attempts-urile unui User la Quiz-uri
- /attempts/ [POST] (function addAttempt) – permite adaugarea unui noi Attempt in momentul in care un utilizator rezolva un chestionar

Contributia membrilor echipei

Moldovan Bogdana – implementare front-end, configurari Swagger, configurari pentru persistenta si set-up proiect back-end
Filipas Razvan – implementare end-pointuri back-end: getAllQuizes, createQuiz, getAllQuestions, createQuestions
Diaconu Horia – implementare end-pointuri back-end: getAllQuestionsByQuiz, addQuestionToAQuiz, getAllAttemptsByUserId, createAttempt