**PROJECT DETAILS:**

**PHASE 3 PROJECT.**

EMPLOYEE ID : 2527463

EMPLOYEE NAME : Vishnu R

PROJECT DETAILS: JAVA FSD Phase 3 assessment module.

To develop a **springboot** **based quiz application**

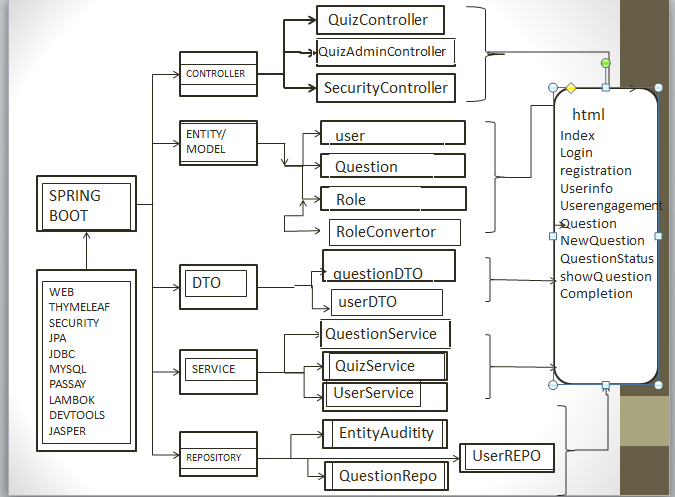
prototype module.

**SPRINTS AND TASKS:**

The task is planned to be completed in a single sprint. The tasks to be completed are

* Creating flow of application.
* Initializing git repository to track changes as the application progress.
* Writing and compiling java program to fulfill the required condition.
* Testing and Dry run the application developed to meet the user requirement.
* Push the Code into github.
* Specifying application details.

**ALGORITHM AND FLOWCHART**



THE ABOVE FLOWCHART DIAGRAM EXPLAINS THE WORKING FLOW OF THE PROGRAM.

**ALGORITHM/ PROGRAM BUILDING:**

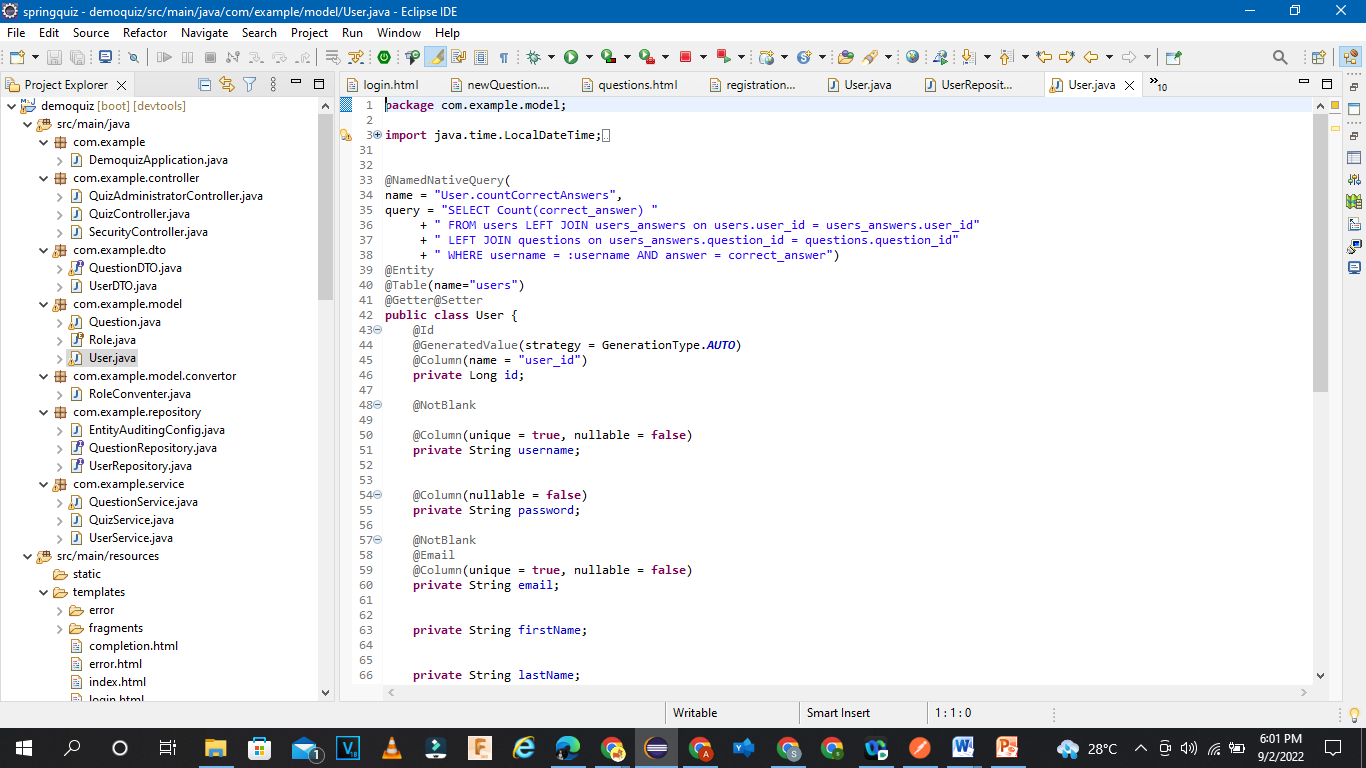
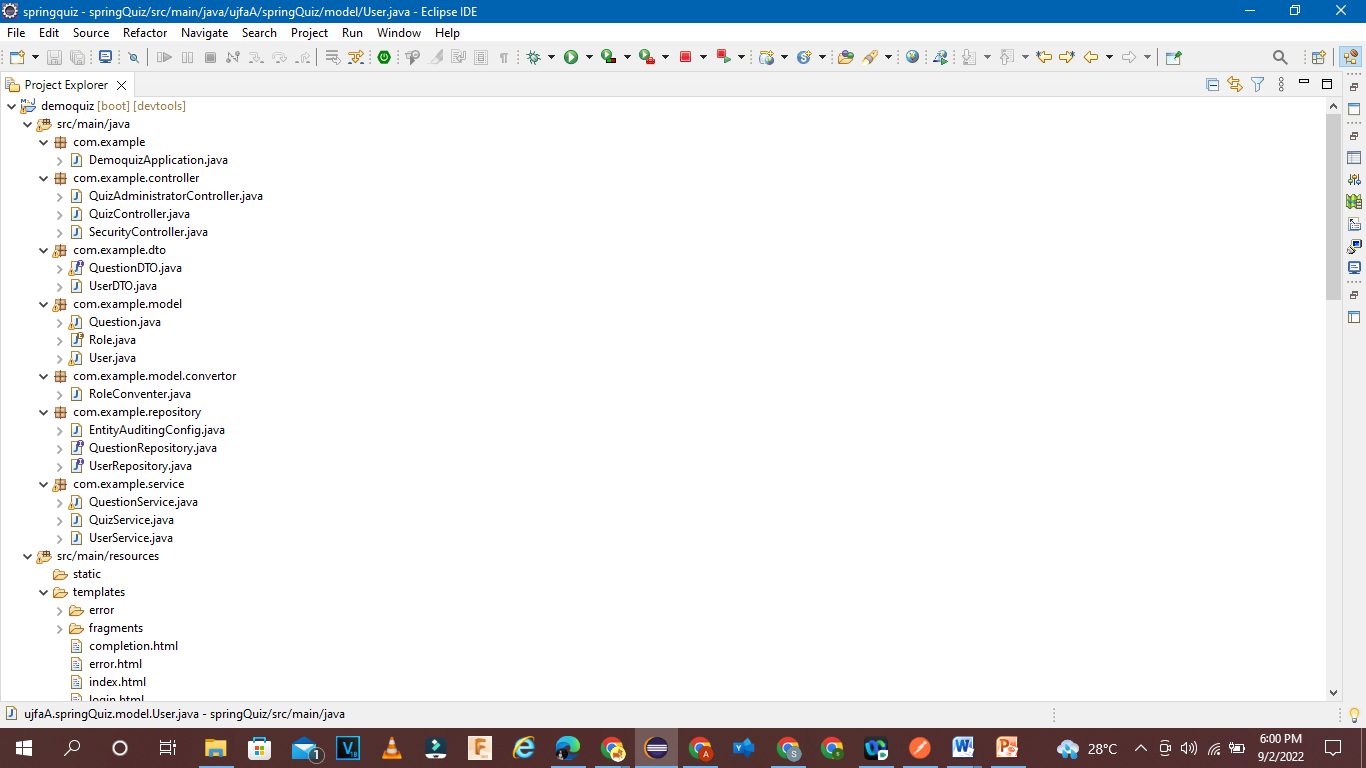
TO DEMONSTRATE THE CONCEPT USED WE ARE STATING THE ALGORITHMIC CONCEPT TO DESIGN THE PROGRAM.

* CREATE THE PROJECT IN ECLIPSE.
* WRITE A PROGRAM TO ALLOW THE ENTRY POINT OR WELCOME PAGE.
* WRITE PROGRAM TO CARRY OUT THE ENTIRE USER DEFINED OPERATIONS.
* RUN THE TEST CASE SCENARIOS.
* UPLOAD THE CONTENT INTO GITHUB LINK.

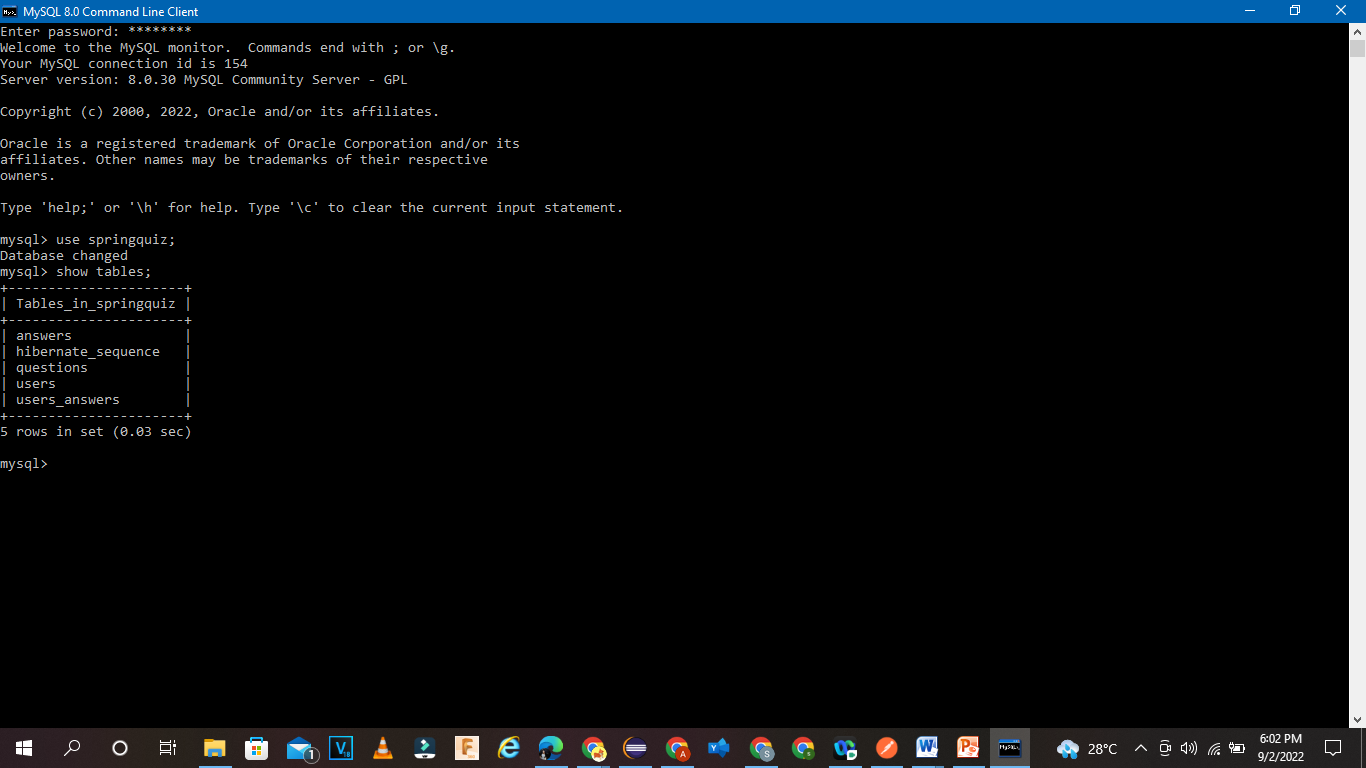
## **Step 1:** Creating a new project in Eclipse and SQL database

* Open Eclipse
* Go to File -> New -> Project -> Maven Project -> Next.
* Type in any project name and click on “Finish.”
* Select your project and go to File -> New -> Class.

Create an entity from which the user defined values will be taken



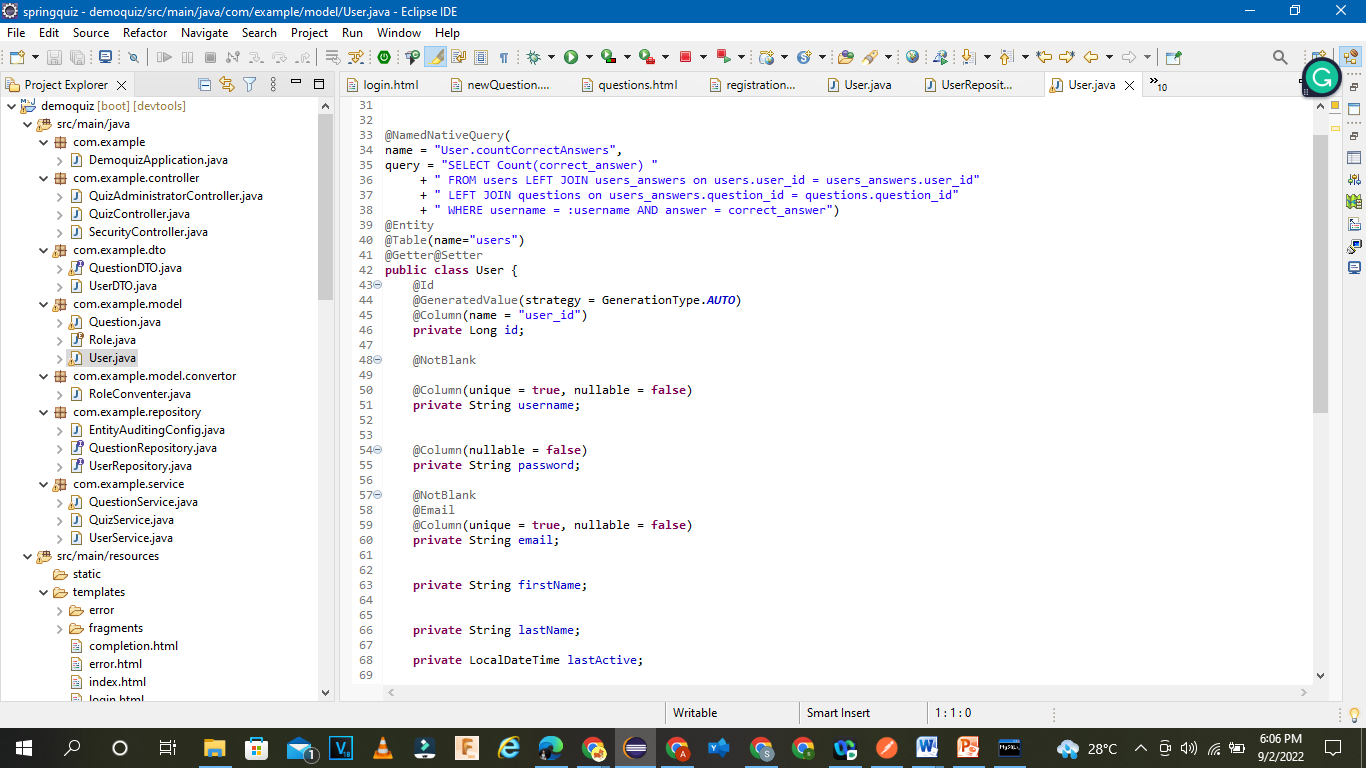
* Create a database in MySQL to access the database and details

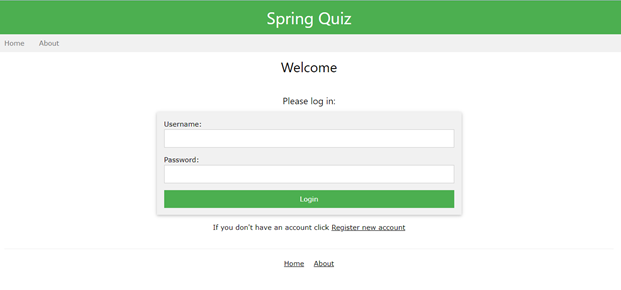


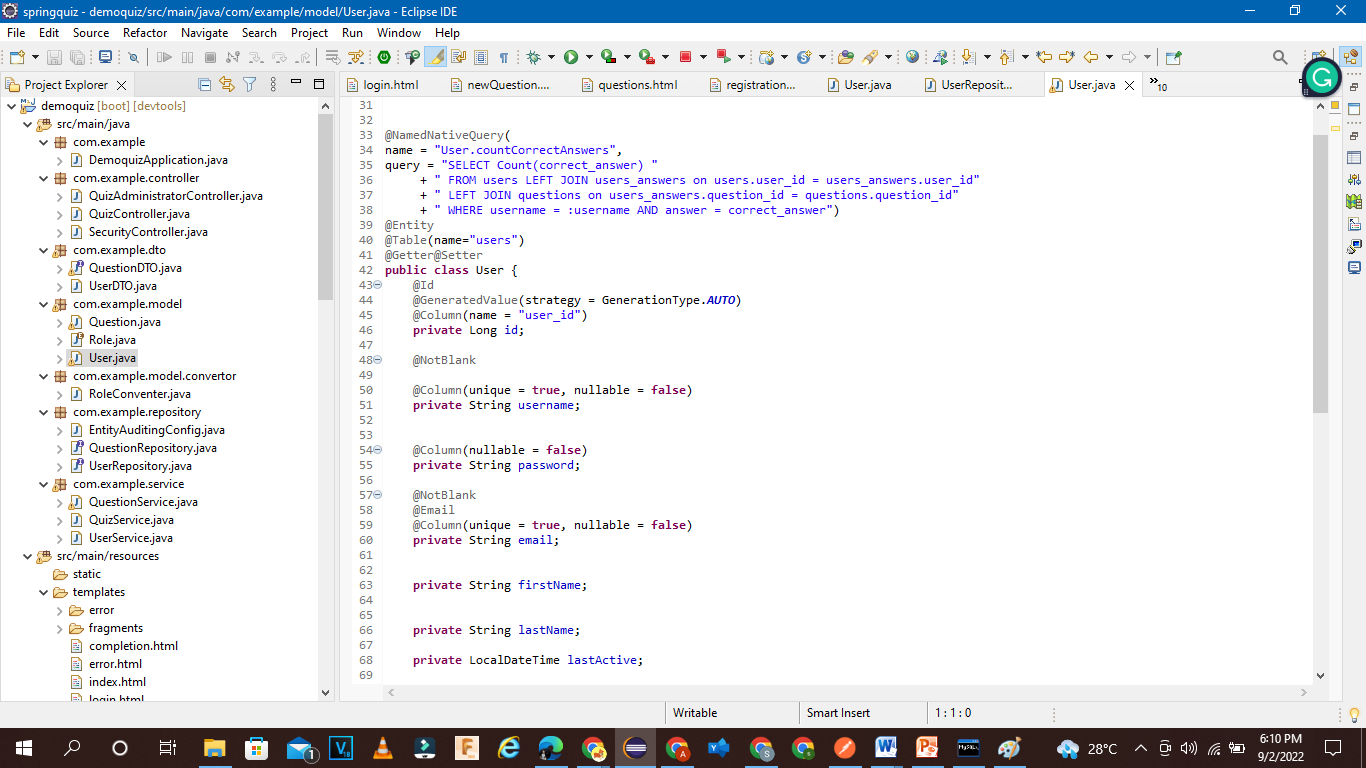
**Step 2: WRITE A PROGRAM TO ALLOW THE ENTRY POINT OR START CREATING USER ENTITY.**

We are creating maven project to include all the dependencies in the program. The maven folder contains all the required jar file details.

* We are creating a class page to get all the required data from the user.
* Check for all the update and installation Lombok from org.lambok from m2 repository to enable possible working or execution of the Lombok setters and getters
* Create entity, controller, service, Dto, repository to access and assess the code.





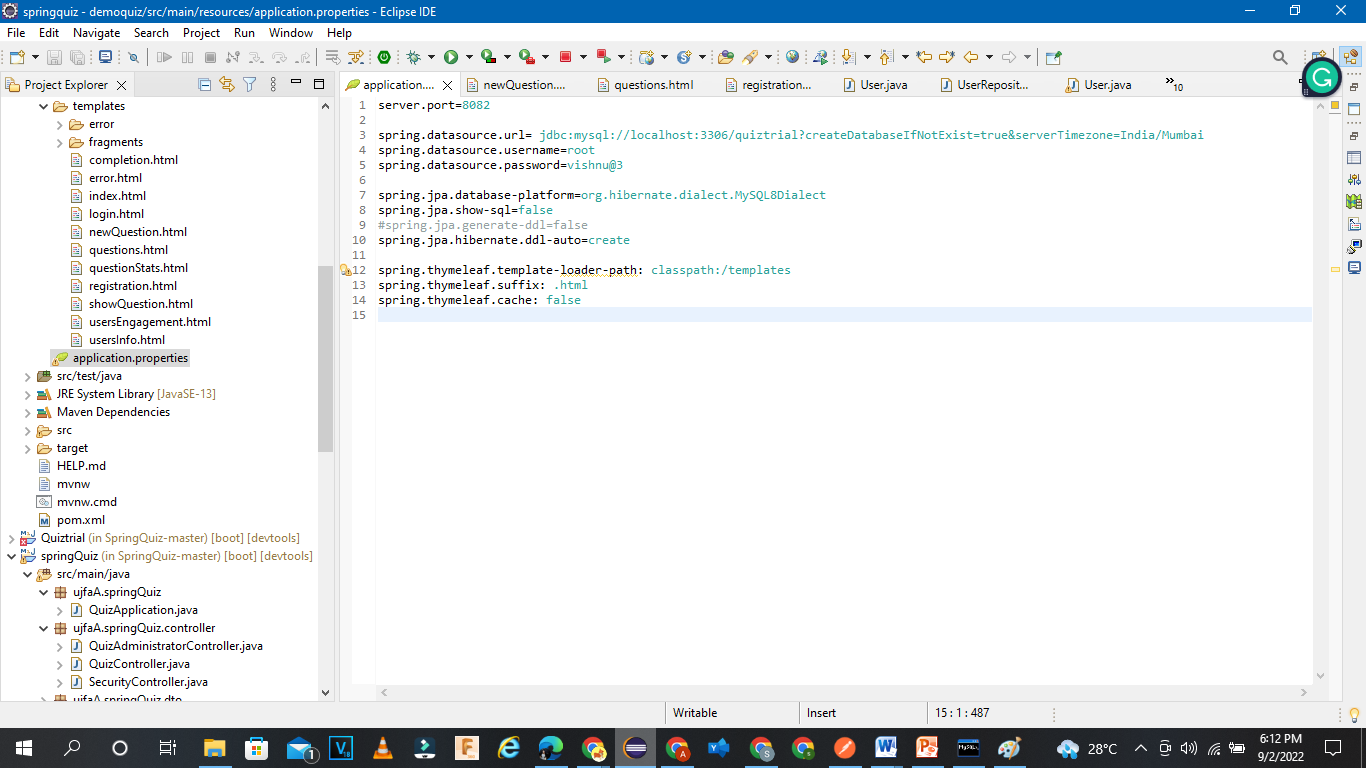


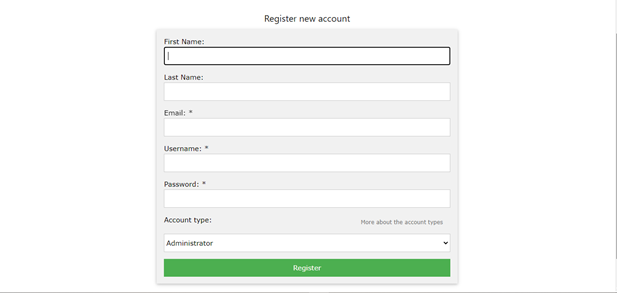
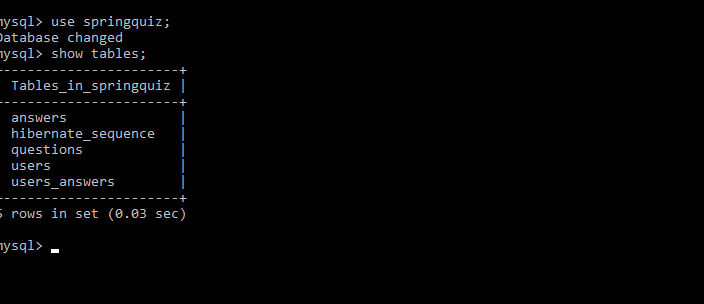
**Step 3:** WRITE PROGRAM TO CARRY OUT THE ENTIRE USER DEFINED OPERATIONS.

We have to enable options to process and access the data from the data base.

In addition we have to enable interfacing jpa repository in order to store the data temporarly.

Enable all the properties in application.properties.





CODE:

**Entity:**

User

package com.example.model;

import java.time.LocalDateTime;

import java.util.HashMap;

import java.util.Map;

import javax.persistence.CollectionTable;

import javax.persistence.Column;

import javax.persistence.Convert;

import javax.persistence.ElementCollection;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.MapKeyJoinColumn;

import javax.persistence.NamedNativeQuery;

import javax.persistence.Table;

import javax.validation.constraints.Email;

import javax.validation.constraints.NotBlank;

import com.example.model.Question;

import com.example.model.User;

import com.example.model.Role;

import com.example.model.convertor.RoleConventer;

import lombok.Getter;

import lombok.Setter;

@NamedNativeQuery(

name = "User.countCorrectAnswers",

query = "SELECT Count(correct\_answer) "

+ " FROM users LEFT JOIN users\_answers on users.user\_id = users\_answers.user\_id"

+ " LEFT JOIN questions on users\_answers.question\_id = questions.question\_id"

+ " WHERE username = :username AND answer = correct\_answer")

@Entity

@Table(name="users")

@Getter@Setter

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name = "user\_id")

private Long id;

@NotBlank

@Column(unique = true, nullable = false)

private String username;

@Column(nullable = false)

private String password;

@NotBlank

@Email

@Column(unique = true, nullable = false)

private String email;

private String firstName;

private String lastName;

private LocalDateTime lastActive;

@Convert(converter = RoleConventer.class)

@Column(length = 4, nullable = false)

private Role role;

@ElementCollection(fetch = FetchType.LAZY)

@CollectionTable(name = "users\_answers", joinColumns = @JoinColumn(name = "user\_id"))

@MapKeyJoinColumn(name = "question\_id")

@Column(name = "answer")

private Map<Question, String> answers = new HashMap<>();

public void storeAnsweredQuestion(Question question, String answer) {

answers.put(question, answer);

}

@Override

public String toString() {

return username;

}

@Override

public boolean equals(Object other) {

if ( ! (other instanceof User))

return false;

User otherU = (User) other;

return this.username.equals(otherU.username);

}

@Override

public int hashCode() {

return this.username.hashCode();

}

}

Question:

package com.example.model;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import javax.persistence.\*;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotEmpty;

import org.hibernate.validator.constraints.UniqueElements;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import lombok.AccessLevel;

import lombok.Getter;

import lombok.Setter;

import com.example.model.Question;

import com.example.model.User;

@Entity @EntityListeners(AuditingEntityListener.class)

@Table(name="questions")

@Getter @Setter

public class Question {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name = "question\_id")

private long id;

@CreatedBy

@ManyToOne(targetEntity = User.class, optional = false)

@JoinColumn(name="created\_by\_user", referencedColumnName = "user\_id", nullable = false)

private User createdBy;

/\* validation \*/

@NotBlank

//

@Column(nullable = false)

private String questionText;

@Column(nullable = false)

private String correctAnswer;

/\* validation \*/

@NotEmpty()

@UniqueElements(message = "Each answer has to be different.")

//

@ElementCollection(fetch = FetchType.LAZY)

@CollectionTable(name = "answers", joinColumns = @JoinColumn(name = "question\_id", referencedColumnName = "question\_id"))

@OrderColumn(name = "ordinal", columnDefinition = "tinyint")

@Column(name = "answer", nullable = false)

private List<@NotBlank String> answers = new ArrayList<String>(); // Includes the correctAnswer.

@Transient

private int selectedAnswerIndex; // Used in a form when user create a question.

@Transient

@Getter(value = AccessLevel.NONE)

@Setter(value = AccessLevel.NONE)

private int hash; // Default to 0

@Transient

@Getter(value = AccessLevel.NONE)

@Setter(value = AccessLevel.NONE)

private boolean hashIsZero; // Default to false;

public Question() {

}

@Override

public String toString() {

return questionText;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if ( ! (obj instanceof Question))

return false;

Question otherQ = (Question) obj;

if ( ! this.questionText.equals(otherQ.questionText))

return false;

/\* Check if answers are the same - irrelevant of order. \*/

return new HashSet<>(this.answers).equals(new HashSet<>(otherQ.answers));

}

@Override

public int hashCode() {

int h = hash;

if (h == 0 && !hashIsZero) {

h = questionText.hashCode();

for (String ans : answers) {

h += ans.hashCode();

}

if (h == 0) {

hashIsZero = true;

} else {

hash = h;

}

}

return h;

}

}

Role:

**package** com.example.model;

**import** com.example.model.Role;

**public** **enum** Role {

***USER***("Basic", "user"),

***CONTRIBUTOR***("Contributor", "cntr"),

***MODERATOR***("Moderator", "mod"),

***ADMINISTRATOR***("Administrator", "admn");

**public** **final** String displayValue;

**public** **final** String shortName; // arbitrary max lenght = 4

Role(String display, String shortName) {

**this**.displayValue = display;

**this**.shortName = shortName;

}

**public** **static** Role fromShortName(String shortName ) {

**switch** (shortName) {

**case** "user":

**return** ***USER***;

**case** "cntr":

**return** ***CONTRIBUTOR***;

**case** "mod":

**return** ***MODERATOR***;

**case** "admn":

**return** ***ADMINISTRATOR***;

**default**:

**throw** **new** IllegalArgumentException("Short[" + shortName + "] not supported.");

}

}

}

Role convertor

package com.example.model.convertor;

import javax.persistence.AttributeConverter;

import com.example.model.Role;

public class RoleConventer implements AttributeConverter<Role, String> {

@Override

public String convertToDatabaseColumn(Role role) {

return role.shortName;

}

@Override

public Role convertToEntityAttribute(String dbData) {

return Role.fromShortName(dbData);

}

}

**DTO:**

Question:

package com.example.dto;

import java.util.List;

import com.example.dto.QuestionDTO.CreatedBy;

public interface QuestionDTO {

Long getId();

CreatedBy getCreatedBy();

String getQuestionText();

String getCorrectAnswer();

List<String> getAnswers();

public interface CreatedBy {

String getUsername();

}

}

User: package com.example.dto;

import java.time.LocalDateTime;

import java.util.Arrays;

import lombok.Value;

import com.example.model.Role;

@Value

public class UserDTO {

String username;

String email;

String firstName;

String lastName;

Role role;

LocalDateTime lastActive;

public UserDTO(String username, String email, String firstName, String lastName, Role role, LocalDateTime lastActive) {

this.username = username;

this.email = redact(email);

this.firstName = firstName;

this.lastName = lastName;

this.role = role;

this.lastActive = lastActive;

}

private String redact(String emailFull) {

StringBuilder sb = new StringBuilder();

String[] parts = emailFull.split("@");

char[] chars = new char[parts[0].length()];

Arrays.fill(chars, '\*');

sb.append(chars);

switch (chars.length) {

case 0:

case 1: break;

case 2: {sb.setCharAt(0, parts[0].charAt(0)); break;}

default: {

int end = parts[0].length()-1;

sb.setCharAt(0, parts[0].charAt(0));

sb.setCharAt(end, parts[0].charAt(end));

break;

}

}

sb.append('@').append(parts[1]);

return sb.toString();

}

}

**SERVICE:**

QuestionService:

package com.example.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.example.dto.QuestionDTO;

import com.example.model.Question;

import com.example.repository.QuestionRepository;

@Service

@Transactional

public class QuestionService {

@Autowired

private QuestionRepository questionRepo;

public Set<QuestionDTO> listAll() {

return questionRepo.findBy();

}

public Set<QuestionDTO> listAllByUser(String username) {

return questionRepo.findByCreatedByUsername(username);

}

public int getNumberOfQuestions() {

return (int) questionRepo.count();

}

public QuestionDTO getQuestion(long questionId) {

return questionRepo.getById(questionId);

}

public Question getQuestionEntity(long id) {

return questionRepo.findById(id).orElseThrow();

}

public boolean exist(Question question) {

Set<Question> fromDB = questionRepo.findByQuestionText(question.getQuestionText());

return fromDB.contains(question);

}

public QuestionDTO getQuestionByIndex(int qIndex) {

Pageable pageRequest = PageRequest.of(qIndex, 1);

Page<QuestionDTO> page = questionRepo.findByOrderById(pageRequest);

if ( ! page.hasContent())

throw new IndexOutOfBoundsException();

return page.getContent().get(0);

}

public Question save(Question q) {

List<String> ans = q.getAnswers();

int markedAsCorrect = q.getSelectedAnswerIndex();

q.setCorrectAnswer(ans.get(markedAsCorrect));

return questionRepo.save(q);

}

public void delete(Long id) {

questionRepo.deleteById(id);

}

public List<String> getQuestionTexts() {

return questionRepo.findAllQuestionTexts();

}

}

QuizService:

package com.example.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.model.Question;

import com.example.model.User;

@Service

public class QuizService {

@Autowired

private QuestionService questionService;

@Autowired

private UserService userService;

public void storeUsersAnswer(String username, long questionId, String answer) {

Question question = questionService.getQuestionEntity(questionId);

User user = userService.getUser(username);

user.storeAnsweredQuestion(question, answer);

userService.update(user);

}

public int getUserScore(String username) {

return userService.getScore(username);

}

public void removeQuestion(long id) {

userService.removeFromUsersAnswers(id);

questionService.delete(id);

}

}

UserService:

package com.example.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.example.dto.QuestionDTO;

import com.example.dto.UserDTO;

import com.example.model.User;

import com.example.repository.UserRepository;

@Service

@Transactional

public class UserService {

@Autowired

private UserRepository userRepo;

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

public boolean usernameIsAvaible(String username) {

return ( ! userRepo.existsUserByUsername(username));

}

public boolean emailIsAvaible(String email) {

return ( ! userRepo.existsUserByEmail(email));

}

public User getUser(String username) {

return userRepo.findByUsername(username).orElseThrow();

}

public User register(User user) {

user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));

return userRepo.save(user);

}

public User update(User user) {

return userRepo.save(user);

}

public Set<UserDTO> getUsersInfo() {

return userRepo.getUsersInfo();

}

public void deleteUser(User user) {

userRepo.delete(user);

}

public Set<String> getUsernamesThatAnswered(QuestionDTO q, boolean correctly ) {

if (correctly)

return userRepo.getUsernamesThatAnsweredWith(q.getId(), q.getCorrectAnswer());

else

return userRepo.getUsernamesThatAnswered(q.getId());

}

public Set<String> getUsernamesThatAnsweredEveryQ(boolean correctly) {

if (correctly)

return userRepo.getUsernamesThatCorrectlyAnsweredEveryQuestion();

else

return userRepo.getUsernamesThatAnsweredEveryQuestion();

}

public int getScore(String username) {

return userRepo.countCorrectAnswers(username);

}

public void removeFromUsersAnswers(long questionId) {

userRepo.removeFromUsersAnswers(questionId);

}

public Float getAnsweredPercentage(long questionId) {

List<Integer> ansCount = userRepo.answersChosenCount(questionId);

int totalAnswers = 0;

for (Integer i : ansCount) {

totalAnswers += i;

}

int timesSkipped = userRepo.skippedCount(questionId);

if (totalAnswers + timesSkipped != 0)

return ((float) totalAnswers) / (totalAnswers + timesSkipped);

else

return -1.0f;

}

public List<Float> getAnswersDistribution(long questionId) {

List<Integer> ansCount = userRepo.answersChosenCount(questionId);

int totalAnswers = 0;

for (Integer i : ansCount) {

totalAnswers += i;

}

List<Float> ansDistribution = new ArrayList<>();

for (Integer i : ansCount) {

if (totalAnswers == 0) {

ansDistribution.add(0.0f);

continue;

}

ansDistribution.add( ((float) i )/ totalAnswers);

}

return ansDistribution;

}

}

**REPOSITORY:**

EntityAudtityrepo:

package com.example.repository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.domain.AuditorAware;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

import org.springframework.security.core.context.SecurityContextHolder;

import com.example.model.User;

@Configuration

@EnableJpaAuditing

public class EntityAuditingConfig {

@Autowired

private UserRepository userRepository;

@Bean

public AuditorAware<User> auditorAware() {

AuditorAware<User> currentAuditor = () -> {String signedInUser = SecurityContextHolder.getContext().getAuthentication().getName();

return userRepository.findByUsername(signedInUser);};

return currentAuditor;

}

}

QuestionRepo:

Package com.example.repository;

import java.util.List;

import java.util.Set;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.PagingAndSortingRepository;

import com.example.dto.QuestionDTO;

import com.example.model.Question;

public interface QuestionRepository extends PagingAndSortingRepository<Question, Long> {

public QuestionDTO getById(Long questionId);

public Set<QuestionDTO> findBy();

public Set<QuestionDTO> findByCreatedByUsername(String username);

public Set<Question> findByQuestionText(String questionText);

public Page<QuestionDTO> findByOrderById(Pageable pageRequest);

@Query("SELECT q.questionText FROM Question q ORDER BY q.id")

public List<String> findAllQuestionTexts();

}

UserRepo:

package com.example.repository;

import java.util.List;

import java.util.Optional;

import java.util.Set;

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.PagingAndSortingRepository;

import com.example.dto.UserDTO;

import com.example.model.User;

public interface UserRepository extends PagingAndSortingRepository<User, Long> {

public Boolean existsUserByUsername(String username);

public boolean existsUserByEmail(String email);

public Optional<User> findByUsername(String username);

@Query("SELECT NEW ujfaA.springQuiz.dto.UserDTO(u.username, u.email, u.firstName, u.lastName, u.role, u.lastActive)"

+ " FROM User u")

public Set<UserDTO> getUsersInfo();

public int countCorrectAnswers(String username);

@Query("SELECT u.username"

+ " FROM User u, IN(u.answers) ans"

+ " WHERE KEY(ans).id = :questionId AND ans <> '' ")

public Set<String> getUsernamesThatAnswered(long questionId);

@Query("SELECT u.username"

+ " FROM User u, IN(u.answers) ans"

+ " WHERE KEY(ans).id = :questionId AND ans = :answer")

public Set<String> getUsernamesThatAnsweredWith(long questionId, String answer);

@Query(nativeQuery = true,

value = "SELECT users.username"

+ " FROM users RIGHT JOIN ("

+ " SELECT user\_id, count(\*) AS questionsAnswered"

+ " FROM users\_answers"

+ " WHERE answer <> '' "

+ " GROUP BY user\_id ) AS t1"

+ " ON users.user\_id = t1.user\_id"

+ " WHERE questionsAnswered = (SELECT count(\*) FROM questions)")

public Set<String> getUsernamesThatAnsweredEveryQuestion();

@Query(nativeQuery = true,

value = "SELECT users.username"

+ " FROM users RIGHT JOIN ("

+ " SELECT user\_id, count(\*) AS questionsAnsweredCorrectly"

+ " FROM users\_answers NATURAL JOIN questions"

+ " WHERE answer = correct\_answer"

+ " GROUP BY user\_id ) AS t1"

+ " ON users.user\_id = t1.user\_id"

+ " WHERE questionsAnsweredCorrectly = (SELECT count(\*) FROM questions)")

public Set<String> getUsernamesThatCorrectlyAnsweredEveryQuestion();

@Modifying

@Query(nativeQuery = true,

value = "DELETE FROM users\_answers WHERE question\_id = ?1")

public void removeFromUsersAnswers(long questionId);

@Query("SELECT count (answer)"

+ " FROM User u, In(u.answers) ans"

+ " WHERE KEY(ans).id = :questionId AND ans = '' ")

public int skippedCount(long questionId);

@Query(nativeQuery = true,

value = "SELECT count(users\_answers.answer)"

+ " FROM answers left join users\_answers on answers.answer = users\_answers.answer AND answers.question\_id = users\_answers.question\_id"

+ " WHERE answers.question\_id = :questionId"

+ " GROUP BY answers.answer"

+ " ORDER BY ordinal")

public List<Integer> answersChosenCount(long questionId);

}

**CONTROLLER:**

QuizAdminController:

package com.example.controller;

import java.security.Principal;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.server.ResponseStatusException;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.dto.QuestionDTO;

import com.example.model.Question;

import com.example.service.QuestionService;

import com.example.service.QuizService;

import com.example.service.UserService;

@Controller

public class QuizAdministratorController {

@Autowired

private QuestionService questionService;

@Autowired

private UserService userService;

@Autowired

private QuizService quizService;

@GetMapping("/questions")

public String getQuestions(ModelMap model) {

model.addAttribute("questions", questionService.listAll());

return "questions";

}

@GetMapping("/questions/byMe")

public String getQuestionsByUser(Principal principal, ModelMap model) {

String currentUsername = principal.getName();

model.addAttribute("questions", questionService.listAllByUser(currentUsername));

return "questions";

}

@GetMapping("/questions/new")

public String newQuestion(@ModelAttribute Question question,

@RequestParam(name = "numberOfAnswers", defaultValue = "3") int numberOfAnswers,

ModelMap model) {

model.addAttribute("numberOfAnswers", numberOfAnswers);

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

@PostMapping("/questions/new")

public String addQuestion(@Valid Question question, BindingResult bindingResult, ModelMap model, RedirectAttributes redirectAttrs) {

if (bindingResult.hasErrors()) {

model.addAttribute("numberOfAnswers", question.getAnswers().size());

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

if (questionService.exist(question)) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "A question like this already exist.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

try {

questionService.save(question);

} catch (Exception e) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "There was an error while adding the question.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

return "redirect:/questions/byMe";

}

@GetMapping("/questions/{qId:[0-9]+}")

public String getQuestionStats(@PathVariable(name = "qId") long qId, Authentication auth, RedirectAttributes redirectAttrs, ModelMap model) {

QuestionDTO question = questionService.getQuestion(qId);

if (question == null)

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "Question not found.");

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

String author = question.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message","You don't have the required permission to see the question's stats.");

return "redirect:/questions/byMe";

}

}

model.addAttribute("question", question);

model.addAttribute("answeredPercentage", userService.getAnsweredPercentage(qId));

model.addAttribute("answersDistribution", userService.getAnswersDistribution(qId));

return "questionStats";

}

@PostMapping("/questions/delete")

public String removeQuestion(@RequestParam Long qId, Authentication auth, RedirectAttributes redirectAttrs) {

String redirectModifier = "";

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

QuestionDTO q = questionService.getQuestion(qId);

String author = q.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message", "You don't have the required permission to delete the question.");

return "redirect:/questions/byMe";

}

redirectModifier = "/byMe";

}

String message;

try {

quizService.removeQuestion(qId);

message = "The question has been removed.";

} catch (Exception e) {

message = "There had been an error while trying to remove the question.";

}

redirectAttrs.addFlashAttribute("message", message);

return "redirect:/questions" + redirectModifier;

}

@GetMapping("/users")

public String getUsersInfo(ModelMap model) {

model.addAttribute("users", userService.getUsersInfo());

return "usersInfo";

}

@GetMapping("/users/usersEng")

public String getUserEngagement(@RequestParam(name = "q", defaultValue = "-1") int qIndex,

@RequestParam(name = "correctly", defaultValue = "false") boolean correctly,

ModelMap model) {

model.addAttribute("texts", questionService.getQuestionTexts());

model.addAttribute("selectedIndex", qIndex);

model.addAttribute("checked", correctly);

if (qIndex == -1) {

model.addAttribute("usernames", userService.getUsernamesThatAnsweredEveryQ(correctly));

}

else {

QuestionDTO q = questionService.getQuestionByIndex(qIndex);

model.addAttribute("usernames", userService.getUsernamesThatAnswered(q, correctly));

}

return"usersEngagement";

}

}

QuizController:

package com.example.controller;

import java.security.Principal;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.server.ResponseStatusException;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.dto.QuestionDTO;

import com.example.model.Question;

import com.example.service.QuestionService;

import com.example.service.QuizService;

import com.example.service.UserService;

@Controller

public class QuizAdministratorController {

@Autowired

private QuestionService questionService;

@Autowired

private UserService userService;

@Autowired

private QuizService quizService;

@GetMapping("/questions")

public String getQuestions(ModelMap model) {

model.addAttribute("questions", questionService.listAll());

return "questions";

}

@GetMapping("/questions/byMe")

public String getQuestionsByUser(Principal principal, ModelMap model) {

String currentUsername = principal.getName();

model.addAttribute("questions", questionService.listAllByUser(currentUsername));

return "questions";

}

@GetMapping("/questions/new")

public String newQuestion(@ModelAttribute Question question,

@RequestParam(name = "numberOfAnswers", defaultValue = "3") int numberOfAnswers,

ModelMap model) {

model.addAttribute("numberOfAnswers", numberOfAnswers);

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

@PostMapping("/questions/new")

public String addQuestion(@Valid Question question, BindingResult bindingResult, ModelMap model, RedirectAttributes redirectAttrs) {

if (bindingResult.hasErrors()) {

model.addAttribute("numberOfAnswers", question.getAnswers().size());

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

if (questionService.exist(question)) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "A question like this already exist.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

try {

questionService.save(question);

} catch (Exception e) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "There was an error while adding the question.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

return "redirect:/questions/byMe";

}

@GetMapping("/questions/{qId:[0-9]+}")

public String getQuestionStats(@PathVariable(name = "qId") long qId, Authentication auth, RedirectAttributes redirectAttrs, ModelMap model) {

QuestionDTO question = questionService.getQuestion(qId);

if (question == null)

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "Question not found.");

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

String author = question.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message","You don't have the required permission to see the question's stats.");

return "redirect:/questions/byMe";

}

}

model.addAttribute("question", question);

model.addAttribute("answeredPercentage", userService.getAnsweredPercentage(qId));

model.addAttribute("answersDistribution", userService.getAnswersDistribution(qId));

return "questionStats";

}

@PostMapping("/questions/delete")

public String removeQuestion(@RequestParam Long qId, Authentication auth, RedirectAttributes redirectAttrs) {

String redirectModifier = "";

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

QuestionDTO q = questionService.getQuestion(qId);

String author = q.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message", "You don't have the required permission to delete the question.");

return "redirect:/questions/byMe";

}

redirectModifier = "/byMe";

}

String message;

try {

quizService.removeQuestion(qId);

message = "The question has been removed.";

} catch (Exception e) {

message = "There had been an error while trying to remove the question.";

}

redirectAttrs.addFlashAttribute("message", message);

return "redirect:/questions" + redirectModifier;

}

@GetMapping("/users")

public String getUsersInfo(ModelMap model) {

model.addAttribute("users", userService.getUsersInfo());

return "usersInfo";

}

@GetMapping("/users/usersEng")

public String getUserEngagement(@RequestParam(name = "q", defaultValue = "-1") int qIndex,

@RequestParam(name = "correctly", defaultValue = "false") boolean correctly,

ModelMap model) {

model.addAttribute("texts", questionService.getQuestionTexts());

model.addAttribute("selectedIndex", qIndex);

model.addAttribute("checked", correctly);

if (qIndex == -1) {

model.addAttribute("usernames", userService.getUsernamesThatAnsweredEveryQ(correctly));

}

else {

QuestionDTO q = questionService.getQuestionByIndex(qIndex);

model.addAttribute("usernames", userService.getUsernamesThatAnswered(q, correctly));

}

return"usersEngagement";

}

}

SecurityController:

package com.example.controller;

import java.security.Principal;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.server.ResponseStatusException;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.dto.QuestionDTO;

import com.example.model.Question;

import com.example.service.QuestionService;

import com.example.service.QuizService;

import com.example.service.UserService;

@Controller

public class QuizAdministratorController {

@Autowired

private QuestionService questionService;

@Autowired

private UserService userService;

@Autowired

private QuizService quizService;

@GetMapping("/questions")

public String getQuestions(ModelMap model) {

model.addAttribute("questions", questionService.listAll());

return "questions";

}

@GetMapping("/questions/byMe")

public String getQuestionsByUser(Principal principal, ModelMap model) {

String currentUsername = principal.getName();

model.addAttribute("questions", questionService.listAllByUser(currentUsername));

return "questions";

}

@GetMapping("/questions/new")

public String newQuestion(@ModelAttribute Question question,

@RequestParam(name = "numberOfAnswers", defaultValue = "3") int numberOfAnswers,

ModelMap model) {

model.addAttribute("numberOfAnswers", numberOfAnswers);

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

@PostMapping("/questions/new")

public String addQuestion(@Valid Question question, BindingResult bindingResult, ModelMap model, RedirectAttributes redirectAttrs) {

if (bindingResult.hasErrors()) {

model.addAttribute("numberOfAnswers", question.getAnswers().size());

model.addAttribute("MAX\_ANSWERS", 5);

return "newQuestion";

}

if (questionService.exist(question)) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "A question like this already exist.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

try {

questionService.save(question);

} catch (Exception e) {

redirectAttrs.addAttribute("numberOfAnswers", question.getAnswers().size());

redirectAttrs.addFlashAttribute("message", "There was an error while adding the question.");

redirectAttrs.addFlashAttribute(question);

return"redirect:/questions/new";

}

return "redirect:/questions/byMe";

}

@GetMapping("/questions/{qId:[0-9]+}")

public String getQuestionStats(@PathVariable(name = "qId") long qId, Authentication auth, RedirectAttributes redirectAttrs, ModelMap model) {

QuestionDTO question = questionService.getQuestion(qId);

if (question == null)

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, "Question not found.");

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

String author = question.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message","You don't have the required permission to see the question's stats.");

return "redirect:/questions/byMe";

}

}

model.addAttribute("question", question);

model.addAttribute("answeredPercentage", userService.getAnsweredPercentage(qId));

model.addAttribute("answersDistribution", userService.getAnswersDistribution(qId));

return "questionStats";

}

@PostMapping("/questions/delete")

public String removeQuestion(@RequestParam Long qId, Authentication auth, RedirectAttributes redirectAttrs) {

String redirectModifier = "";

if (auth.getAuthorities().contains(new SimpleGrantedAuthority("ROLE\_CONTRIBUTOR")) ) {

QuestionDTO q = questionService.getQuestion(qId);

String author = q.getCreatedBy().getUsername();

if ( ! auth.getName().equals(author)) {

redirectAttrs.addFlashAttribute("message", "You don't have the required permission to delete the question.");

return "redirect:/questions/byMe";

}

redirectModifier = "/byMe";

}

String message;

try {

quizService.removeQuestion(qId);

message = "The question has been removed.";

} catch (Exception e) {

message = "There had been an error while trying to remove the question.";

}

redirectAttrs.addFlashAttribute("message", message);

return "redirect:/questions" + redirectModifier;

}

@GetMapping("/users")

public String getUsersInfo(ModelMap model) {

model.addAttribute("users", userService.getUsersInfo());

return "usersInfo";

}

@GetMapping("/users/usersEng")

public String getUserEngagement(@RequestParam(name = "q", defaultValue = "-1") int qIndex,

@RequestParam(name = "correctly", defaultValue = "false") boolean correctly,

ModelMap model) {

model.addAttribute("texts", questionService.getQuestionTexts());

model.addAttribute("selectedIndex", qIndex);

model.addAttribute("checked", correctly);

if (qIndex == -1) {

model.addAttribute("usernames", userService.getUsernamesThatAnsweredEveryQ(correctly));

}

else {

QuestionDTO q = questionService.getQuestionByIndex(qIndex);

model.addAttribute("usernames", userService.getUsernamesThatAnswered(q, correctly));

}

return"usersEngagement";

}

}

**CORE CONCEPTS USED:**

* MAVEN
* DEPENDENCY
* HTML PAGES
* EXCEPTION HANDLING
* CONDITIONAL STATEMENTS
* JASPER
* PASSAY

**GITHUB LINK:**

[**https://github.com/akash-n/assesment**](https://github.com/akash-n/assesment)

**UNIQUE POINTS OF THE PROGRAM:**

* If the username or password already exists, then it throws an exception. The quiz application enables the user to create a profile if there is no record for previous sign or knowledge. A new username and password is created in order for the application to be processes.
* We allow two type of major purpose such a administrator and applicator. additional feature such moderator etc. also included.
* We can add new questions to a quiz in the module through the add question available in the page.
* Finally the result or score card is prompted when the system completes the execution of the program.

**CONCLUSION:**

Thus, all the required data modulation according to the problem statement has been designed. Further, enhancement or modification is also available to

An admin login page where the admin can change the password after login, if he wishes  
● a list to ad and delete new questions  
● A procedure to enable change from admin and user  
● A scorecard available to access the quiz evaluation.