KIRAN KUMAR D NATIKAR

EMP ID:2576955

ALGORITHM:

Algorithm:

1. AddSQLQuestionServlet:

- Extract the question, options, and correct option from the request parameters.
- Create a new SQLQuestion object and set its properties.
- Open a Hibernate session.
- Begin a transaction.
- Save the SQLQuestion object to the database using the session.
- Commit the transaction if successful; otherwise, rollback the transaction.
- Close the session.
- Redirect the response to the "addquestion.jsp" page.

2. DeleteSQLQuestionServlet:

- Retrieve the question ID from the request parameter.
- Delete the SQL question from the database using the SQLQuestionDAO class.
- Redirect the response back to the view page ("/viewsqllist").

3. EditSQLQuestionServlet:

- Retrieve the question ID from the request parameter.
- Retrieve the SQL question from the database using the SQLQuestionDAO class.
- Forward the question object to the "editsqlquestion.jsp" page for editing.
- Retrieve the updated question details from the request parameters.
- Update the SQL question object with the new values.
- Update the SQL question in the database using the SQLQuestionDAO class.
- Redirect the response to the "viewsqllist.jsp" page.

4. SQLQuestion.java:

- Define the SQLQuestion class with properties such as ID, question, options, and correct option.
- Implement getter and setter methods for the properties.
- Implement a method to format the options as a string.

5. SQLQuestionDAO.java:

- Implement a method to retrieve all SQL questions from the database.

KIRAN KUMAR D NATIKAR

EMP ID:2576955

ALGORITHM:

- Implement a method to format the options of a SQLQuestion object as a string.
- Implement a method to retrieve a SQL question by its ID from the database.
- Implement a method to delete a SQL question from the database.
- Implement a method to update a SQL question in the database.

6. SQLTestServlet:

- Retrieve all SQL questions from the database using the SQLQuestionDAO class.
- Set the SQL questions as a request attribute.
- Forward the request to the "sqlTest.jsp" page.

7. SubmitAnswerServlet:

- Create a map to store user answers.
- Retrieve the submitted answers from the request parameters.
- Iterate over the SQL questions.
- Extract the question ID.
- Get the selected option from the request parameters.
- Store the question ID and selected option in the user answers map.
- Calculate the score by comparing user answers with the correct options.
- Set the score as a request attribute.
- Forward the request to the "sqlTestResult.jsp" page.

8. ViewSQLQuestionServlet:

- Retrieve the list of SQL questions from the database using the SQLQuestionDAO class.
- Set the question list as a request attribute.
- Forward the request to the "viewsqllist.jsp" page.

9. Admin.java:

- Define a class `Admin` with private instance variables for `id`, `email`, and `password`.
- Generate getters and setters for the variables.

10. AdminDAO.java:

- Create a class `AdminDAO` responsible for handling data access operations for the `Admin` entity.

KIRAN KUMAR D NATIKAR

EMP ID:2576955

ALGORITHM:

- Create a session factory using HibernateUtil.
- Implement a method `validateAdmin` that takes an email and password as parameters.
- Open a session, create a query to retrieve an `Admin` based on the provided email and password.
- Set query parameters and retrieve the unique result (an instance of `Admin`) or null.
- Handle any exceptions and return null if an exception occurs.

11. AdminLoginServlet.java:

- Create a servlet `AdminLoginServlet` to handle admin login requests.
- Retrieve the email and password from the request parameters.
- Create an instance of `AdminDAO` and call `validateAdmin` to check if the admin credentials are valid.
- If the admin is authenticated, set a session attribute to indicate the authentication and redirect to the admin dashboard page.
- If the admin credentials are invalid, redirect to the login page with an error message.

12. DeleteUserServlet.java:

- Create a servlet `DeleteUserServlet` to handle deleting a user.
- Retrieve the `userId` parameter from the request.
- Create an instance of `UserDAO` and call `deleteUser` to delete the user with the given ID.
- Redirect to the "viewuser.jsp" page.

13. EditUserServlet.java:

- Create a servlet `EditUserServlet` to handle editing a user.
- Retrieve the `userId` parameter from the request.
- Create an instance of `UserDAO` and call `getUserById` to get the user with the given ID.
- Set the retrieved user as a request attribute and forward the request to the "edituser.jsp" page.

14. HibernateUtil.java:

- Create a class `HibernateUtil` to handle Hibernate session factory creation.
- Implement a static method `buildSessionFactory` to build the session factory using the configuration from "hibernate.cfg.xml".
- Implement a static method `getSessionFactory` to return the session factory.
- Implement a static method `shutdown` to close the session factory.

KIRAN KUMAR D NATIKAR

EMP ID:2576955

ALGORITHM:

15. LogoutServlet.java:

- Create a servlet `LogoutServlet` to handle user logout.
- Invalidate the current session.
- Redirect to the "loggedout.jsp" page.

16. RegisterServlet.java:

- Create a servlet `RegisterServlet` to handle user registration.
- Retrieve the name, email, password, and dob (date of birth) from the request parameters.
- Parse the dob string into a `Date` object.
- Create a new instance of `User`, set the retrieved data, and save it using `UserDAO`'s `registerUser` method.
- Redirect to the "userdashboard.jsp" page.

17. UpdateUserServlet.java:

- Create a servlet `UpdateUserServlet` to handle updating a user.
- Retrieve the `userId` parameter from the request.
- Retrieve the updated name and email from the request parameters.
- Create an instance of `UserDAO` and call `getUserById` to get the user with the given ID.
- If the user exists, update the name and email and call `updateUser` in `UserDAO`.
- Redirect to the "viewuser.jsp" page.

18. User.java:

- Define a class `User` with private instance variables for `id`, `name`, `email`, `password`, and `dob`.
- Generate getters and setters for the variables.

19. UserDAO.java:

- Create a class `UserDAO` responsible for handling data access operations for the `User` entity.
- Implement methods for user registration, user validation, retrieving all users, getting a user by ID, updating a user, and deleting a user.

20. UserLoginServlet.java:

KIRAN KUMAR D NATIKAR

EMP ID:2576955

ALGORITHM:

- Create a servlet `UserLoginServlet` to handle user login requests.
- Retrieve the email and password from the request parameters.
- Create an instance of `UserDAO` and call `validateUser` to check if the user credentials are valid.
- If the user is authenticated, set a session attribute with the user's name and redirect to the user dashboard.

21. hibernate.cfg.xml:

- This file is the configuration file for Hibernate, a popular Object-Relational Mapping (ORM) framework. It defines the database connection settings, such as the JDBC driver, URL, username, and password. It also includes mappings for entity classes.

22. **JSP files:** JSP (JavaServer Pages) is a technology used for creating dynamic web pages. The JSP files you provided contain HTML markup with embedded Java code snippets.

- edituser.jsp`: This JSP file is used for editing user information. It displays a form with fields for name and email, pre-filled with the existing user's data, and allows the user to update their information.
- addquestion.jsp`: This JSP file displays a table of subjects and provides links to add and view questions for each subject. In the provided snippet, it includes options for the "SQL" subject.
- addquestionsql.jsp`: This JSP file is used to add SQL questions. It displays a form with fields for question, options, and correct option. Upon submission, it likely sends the data to the server to be processed.
- admindashboardpage.jsp`: This JSP file represents the admin dashboard page. It displays options for adding questions to the quiz and viewing the user list.
- adminlogin.jsp`: This JSP file provides a login form for the admin user. It accepts an email and password and submits them to the server for authentication.
- editsqlquestion.jsp`: This JSP file is used for editing SQL questions. It displays a form pre-filled with the existing question's data, allowing the admin to modify the question, options, and correct option.
- edituser.jsp`: This JSP file is similar to `edituser.jsp` mentioned earlier. It allows the admin to edit user information.
- error.jsp`: This JSP file is displayed when an error occurs during the application's execution. It shows a simple error message.
- header.jsp`: This JSP file includes a common header section that is included in other JSP files. It likely contains navigation links or other common elements.
- index.jsp`: This JSP file represents the index page of the application. It displays options for user registration, user login, and admin login.
- loggedout.jsp`: This JSP file is displayed when a user or admin logs out. It shows a message indicating that the session has terminated.
- sqltest.jsp`: This JSP file is used for conducting an SQL test. It displays SQL questions retrieved from the database and provides options for selecting answers. Upon submission, it likely sends the answers to the server for evaluation.
- sqltestresult.jsp`: This JSP file is displayed after completing an SQL test. It shows the user's score and provides an option to go back to the dashboard.
- userdashboard.jsp`: This JSP file represents the user dashboard page. It likely displays options specific to the user, such as taking tests or viewing results.