Car Accessories Company Project

|  |  |
| --- | --- |
| **Students** | |
| 1. Riham Katout – 12029366 | 1. Shahd Hamad - 12028064 |

Table of Contents

[Main Classes 3](#_Toc152187221)

[a) Starter Class (main class) 3](#_Toc152187222)

[b) UserSession Class 3](#_Toc152187223)

[Database Classes 4](#_Toc152187224)

[a) DatabaseConnection Class 4](#_Toc152187225)

[b) RetrievingData Class 5](#_Toc152187226)

[c) InsertingData Class 6](#_Toc152187227)

[Helper Classes 7](#_Toc152187228)

[a) Generator Class 7](#_Toc152187229)

[b) DataValidation Class 8](#_Toc152187230)

[Model Classes 9](#_Toc152187231)

[a) User Class 9](#_Toc152187232)

[b) Address Class 10](#_Toc152187233)

[Authentication Classes 11](#_Toc152187234)

[a) UserSessionManager 11](#_Toc152187235)

[b) Login Class 12](#_Toc152187236)

[c) Register Class 13](#_Toc152187237)

[Task1 14](#_Toc152187238)

[Tester Classes 14](#_Toc152187239)

[a) DatabaseTester 14](#_Toc152187240)

[b) AuthenticationTester 14](#_Toc152187241)

[Feature 1: Database Connection 14](#_Toc152187242)

[Scenarios 14](#_Toc152187243)

[Steps definition Class 15](#_Toc152187244)

[Result 15](#_Toc152187245)

[Feature 2: Retrieving from DB 16](#_Toc152187246)

[Scenarios 16](#_Toc152187247)

[Steps definition Class 17](#_Toc152187248)

[Result 17](#_Toc152187249)

[Feature 3: Login 18](#_Toc152187250)

[Scenarios 18](#_Toc152187251)

[Steps definition Class 18](#_Toc152187252)

[Result 19](#_Toc152187253)

[Feature 4: Register 20](#_Toc152187254)

[Scenarios 20](#_Toc152187255)

[Steps definition Class 20](#_Toc152187256)

[Result 21](#_Toc152187257)

# Main Classes

## Starter Class (main class)

public class Starter extends Application {  
 public static DatabaseConnection *connector*;  
 public static UserSessionManager *sessionManager*;  
 public static UserSession *userSession*;

@Override  
 public void start(Stage stage) throws IOException {  
 *sessionManager* = new UserSessionManager();  
 *connector* = new DatabaseConnection();  
 System.*out*.println(*connector*.getStatus());  
  
  
 FXMLLoader fxmlLoader = new FXMLLoader(Starter.class.getResource("/FXMLFiles/login.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 608, 837);  
 stage.setTitle("Car Zone Company");  
 stage.setScene(scene);  
 stage.setResizable(false);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

* connector: instance of [DatabaseConnection](#_DatabaseConnection_Class) Class.
* sessionManager: instance of [UserSessionManager](#_UserSessionManager) Class.
* userSession: instance of [UserSession](#_UserSession_Class) Class, use it after logging in.

## UserSession Class

public class UserSession extends User{  
 private String sessionId;  
  
 public UserSession(User user) {  
 super(user);  
 }  
  
 public String getSessionId() {  
 return sessionId;  
 }  
  
 public void setSessionId(String sessionId) {  
 this.sessionId = sessionId;  
 }  
  
 @Override  
 public String toString() {  
 return "Classes.UserSession{" + getUsername() + ": " + "sessionId='" + sessionId + '\'' + '}';  
 }  
}

* Extends [User](#_User) Class to use its methods and attributes.
* Contains a user (create it using *super* conctructor)

# Database Classes

## DatabaseConnection Class

* To establish database connection given it databaseName, username, password, root, port.
* status will use for testing, same for all classes.

public class DatabaseConnection {  
 private String databaseName, username, password, status;  
 private int port;  
 private Connection con;  
   
 public DatabaseConnection(){  
 setPort(3306);  
 setDatabaseName("caraccessoriescompany");  
 setUsername("root");  
 setPassword("12345678password");  
 setCon();  
 }  
 public DatabaseConnection(int port, String databaseName, String username, String password){  
 setPort(port);  
 setDatabaseName(databaseName);  
 setUsername(username);  
 setPassword(password);  
 setCon();  
 }  
 public String getStatus() {return status;}  
 public Connection getCon() {return con;}   
 public void setDatabaseName(String databaseName) {this.databaseName = databaseName;}  
 public void setUsername(String username) {this.username = username;}  
 public void setPassword(String password) {this.password = password;}  
 public void setPort(int port) {this.port = port;}

public void setStatus(String status) {this.status = status;}  
 public void setCon() {  
 String url = "jdbc:mysql://localhost:" + port + "/" + databaseName;  
 try{  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 con= DriverManager.*getConnection*(url, username, password);  
 setStatus("Connected to the database successfully");  
 }catch(Exception e){  
 System.*out*.println(e);  
 setStatus("Couldn't connect to the database");  
 }  
 }  
}

## RetrievingData Class

* Not ready yet (we will add more entities to it)
* Retrieving records from database giving the condition and entity name to the suitable function.

public class RetrievingData{  
 private Connection con;  
 private String status;  
  
 public RetrievingData(Connection con){this.con = con;}  
  
 public String getStatus() {return status;}  
  
 public void setStatus(String status) {this.status = status;}  
  
 private ResultSet getFromData(String entity, String condition) throws Exception{  
 ResultSet rs = null;  
 String query = "SELECT \* FROM " + entity + " " + (condition.equals("") ? "":"where " + condition);  
 Statement st = con.createStatement();  
 rs = st.executeQuery(query);  
 return rs;  
 }

public List<User> selectUsers(String condition){  
 List<User> users = new ArrayList<>();  
 try {  
 ResultSet rs = getFromData("users", condition);  
 while (rs != null && rs.next())  
 users.add(Generator.*rsToUser*(rs));  
  
 setStatus("Retrieving users successfully");  
 return users;  
 }catch (Exception e){  
 setStatus("Error while retrieving users from database");  
 return null;  
 }  
 }  
  
 public List<Address> selectAddresses(String condition){  
 List<Address> addresses = new ArrayList<>();  
 try {  
 ResultSet rs = getFromData("addresses", condition);  
 while (rs != null && rs.next())  
 addresses.add(Generator.*rsToAddress*(rs));  
  
 setStatus("Retrieving addresses successfully");  
 return addresses;  
 }catch (Exception e){  
 setStatus("Error while retrieving addresses from database");  
 return null;  
 }  
 }  
}

* **connection**: connection to database.
* **getFromData**: from entity and condition, return the resultSet for retrieving records.
* same implementation for all entities, call ***getFromData*** then use [generator](#_Generator) class to generate the object from rs.

## InsertingData Class

* Not ready yet (we will add more entities to it)
* Inserting to database giving the object to the function

public class InsertingData {  
 private String status;  
 private Connection connection;

public InsertingData(Connection connection){connection = connection;}  
 public String getStatus() {return status;}  
  
 public void setStatus(String status) {this.status = status;}  
  
 public boolean insertUser(User user){  
 try {  
 String query = "insert into users " + " values (?, ?, ?, ?, ?, ?, ?);";  
 PreparedStatement preparedStmt = connection.prepareStatement(query);  
 preparedStmt = Generator.*userToPS*(preparedStmt, user);  
 preparedStmt.execute();  
 setStatus("User was inserted successfully");  
 return true;  
 } catch (Exception e) {  
 setStatus("Couldn't insert user");  
 return false;  
 }  
 }  
}

* same implementation for all entities, use [generator](#_Generator) class to generate the Prepared Statement from object.

# Helper Classes

## Generator Class

* Not ready yet

public class Generator {  
 public static User rsToUser(ResultSet rs) throws SQLException {  
 User tmpUser = new User();  
 tmpUser.setFirstName(rs.getString("firstName"));  
 tmpUser.setLastName(rs.getString("lastName"));  
 tmpUser.setUsername(rs.getString("username"));  
 tmpUser.setPhoneNumber(rs.getString("phone"));  
 tmpUser.setEmail(rs.getString("email"));  
 tmpUser.setPassword(rs.getString("userPassword"));  
 tmpUser.setImagePath(rs.getString("image"));  
 return tmpUser;  
 }  
  
 public static Address rsToAddress(ResultSet rs) throws SQLException {  
 Address tmpAddress = new Address();  
 tmpAddress.setCountry(rs.getString("country"));  
 tmpAddress.setCity(rs.getString("city"));  
 tmpAddress.setStreet(rs.getString("street"));  
 return tmpAddress;  
 }  
 public static PreparedStatement userToPS(PreparedStatement preparedStmt, User user) throws SQLException {  
 preparedStmt.setString(1, user.getFirstName());  
 preparedStmt.setString(2, user.getLastName());  
 preparedStmt.setString(3, user.getUsername());  
 preparedStmt.setString(4, user.getPhoneNumber());  
 preparedStmt.setString(5, user.getEmail());  
 preparedStmt.setString(6, user.getPassword());  
 preparedStmt.setString(7, "");  
 return preparedStmt;  
 }  
}

## DataValidation Class

public class DataValidation {  
  
 public static boolean regexMatcher(String regex, String value){  
 Pattern pattern = Pattern.*compile*(regex);  
 Matcher matcher = pattern.matcher(value);  
 return matcher.matches();  
 }  
  
 public static String userValidationTest(User user){  
 if(user.getEmail().equals("")) return "Email address can't be empty";  
 if(user.getPhoneNumber().equals("")) return "Phone number can't be empty";  
 if(user.getPassword().equals("")) return "Password can't be empty";  
 if(user.getUsername().equals("")) return "Username can't be empty";  
 if(user.getFirstName().equals("")) return "First name can't be empty";  
 if(user.getLastName().equals("")) return "Last name can't be empty";  
 if(*emailValidationTest*(user.getEmail())){  
 if(*phoneNumberValidationTest*(user.getPhoneNumber())) {  
 if (*passwordValidationTest*(user.getPassword())) return "Valid";  
 return "Invalid password";  
 }  
 return "Invalid phone number";  
 }  
 return "Invalid email address";  
 }  
  
 public static boolean phoneNumberValidationTest(String phoneNumber) {  
 String regex = "^[0-9]{10}$";  
 return *regexMatcher*(regex, phoneNumber);  
 }  
  
 public static boolean emailValidationTest(String email) {  
 String regex = "^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";  
 return *regexMatcher*(regex, email);  
 }  
  
 public static boolean passwordValidationTest(String password) {  
 String regex = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[@$!%\*?&])[A-Za-z\\d@$!%\*?&]{8,}$";  
 return *regexMatcher*(regex, password);  
  
 }

# Model Classes

## User Class

public class User {  
 private String username, firstName, lastName, phoneNumber, password, email, imagePath;  
 private Address address;

public User(){  
 this.username = ""; this.firstName = ""; this.lastName = ""; this.phoneNumber = "";  
 this.password = ""; this.email = ""; this.imagePath = ""; this.address = null;  
 }  
 public User(String username, String firstName, String lastName, String phoneNumber,

String password, String email, String imagePath, Address address) {  
 setUsername(username); setFirstName(firstName);  
 setLastName(lastName); setPhoneNumber(phoneNumber);  
 setPassword(password); setEmail(email);  
 setImagePath(imagePath); setAddress(address);  
 }  
 public User(User user){  
 this(user.getUsername(), user.getFirstName(), user.getLastName(),  
 user.getPhoneNumber(), user.getPassword(), user.getEmail(),  
 user.getImagePath(), user.getAddress());  
 }

public String getUsername() {return username;}  
  
 public void setUsername(String username) {this.username = username.toLowerCase();}  
  
 public String getFirstName() {return firstName;}  
  
 public void setFirstName(String firstName) {this.firstName = firstName;}  
  
 public String getLastName() {return lastName;}  
  
 public void setLastName(String lastName) {this.lastName = lastName;}  
  
 public String getPhoneNumber() {return phoneNumber;}  
  
 public void setPhoneNumber(String phoneNumber) {this.phoneNumber = phoneNumber;}  
  
 public String getPassword() {return password;}  
  
 public void setPassword(String password) {this.password = password;}  
  
 public String getEmail() {return email;}  
  
 public void setEmail(String email) {this.email = email;}  
  
 public String getImagePath() {return imagePath;}  
  
 public void setImagePath(String imagePath) {this.imagePath = imagePath;}  
  
 public Address getAddress() {return address;}  
  
 public void setAddress(Address address) {this.address = address;}  
  
 @Override  
 public String toString() {  
 return "User{" + "username='" + username + '\'' + ", firstName='" + firstName + '\''

+ ", lastName='" + lastName + '\'' + ", phoneNumber='" + phoneNumber + '\''

+ ", password='" + password + '\'' + ", email='" + email + '\''

+ ", imagePath='" + imagePath + '\'' + ", address=" + address + '}';  
 }  
}

## Address Class

public class Address {  
 private String country, city, street;  
  
 public String getCountry() {  
 return country;  
 }  
  
 public void setCountry(String country) {  
 this.country = country;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public String getStreet() {  
 return street;  
 }  
  
 public void setStreet(String street) {  
 this.street = street;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "country='" + country + '\'' +  
 ", city='" + city + '\'' +  
 ", street='" + street + '\'' +  
 '}';  
 }  
}

# Authentication Classes

## UserSessionManager

public class UserSessionManager {  
 private static Map<String, String> *userSessions* = new HashMap<>();  
  
 // Method to create a session for a user upon successful login  
 public static String createSession(String username) {  
 String sessionId = *generateSessionId*();  
 *userSessions*.put(sessionId, username);  
 System.*out*.println("Session created for user: " + username + ", Session ID: " + sessionId);  
 return sessionId;  
 }  
  
 // Method to check if a session exists for a given session ID  
 public static boolean isValidSession(String sessionId) {  
 return *userSessions*.containsKey(sessionId);  
 }  
  
 // Method to retrieve username from a session ID  
 public static String getUsernameFromSession(String sessionId) {  
 return *userSessions*.get(sessionId);  
 }  
  
 // Simulated session ID generation method  
 private static String generateSessionId() {  
 String tmpID;  
 do{  
 tmpID = "SESSION\_" + System.*currentTimeMillis*();  
 }while (*isValidSession*(tmpID));  
 return tmpID; // Not a secure way, just for demonstration  
 }  
  
 // Method to invalidate a session (logout)  
 public static void invalidateSession(String sessionId) {  
 *userSessions*.remove(sessionId);  
 System.*out*.println("Session invalidated for Session ID: " + sessionId);  
 }  
  
 // For demonstration: displaying all active sessions  
 public static void displayActiveSessions() {  
 System.*out*.println("Active Sessions:");  
 for (Map.Entry<String, String> entry : *userSessions*.entrySet()) {  
 System.*out*.println("Session ID: " + entry.getKey() + ", Username: " + entry.getValue());  
 }  
 }  
  
}

* To generate a session when logging in, or delete it when signing out.

## Login Class

public class Login {  
 private String status;  
 private RetrievingData usersRetriever;  
 public Login(Connection connection) {  
 usersRetriever = new RetrievingData(connection);  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public boolean loginUser(String username, String password){  
 username = username.toLowerCase();  
  
 List<User> allUsers = usersRetriever.selectUsers("username = '" + username + "'");  
 if(allUsers != null && allUsers.size() != 0 ) {  
 User tmpUser = allUsers.get(0);  
 if (tmpUser.getPassword().equals(password)) {  
 setStatus("Valid username and password");  
 Starter.*userSession* = new UserSession(tmpUser);  
 Starter.*userSession*.setSessionId(Starter.*sessionManager*.*createSession*(username));  
 return true;  
 }  
 }  
 setStatus("Invalid username or password");  
 return false;  
 }  
  
  
}

* **loginUser**: the main method in this class, it’s used to check if the username is existed using ***usersRetriever*** (an

instance of [RetrievingData](#_RetrievingData_Class) Class), if yes, check the password, if it’s correct, then create a session

using [UserSession](#_UserSession_Class) and [UserSessionManager](#_UserSessionManager) classes.

## Register Class

public class Register {  
 private String status;  
 private InsertingData userInserter;  
 private RetrievingData userRetriever;  
  
 public Register(Connection connection){  
 userInserter = new InsertingData(connection);  
 userRetriever = new RetrievingData(connection);  
 }  
 public String getStatus() {return status;}  
  
 public void setStatus(String status) {this.status = status;}  
  
 public void registerUserTest(User user){  
 String st = DataValidation.*userValidationTest*(user);  
 if(st.equals("Valid")){  
 List<User> allUsers = userRetriever.selectUsers("username = '" + user.getUsername() + "'");  
 if(allUsers == null || allUsers.size() == 0 )  
 setStatus("User was registered successfully");  
 else  
 setStatus("Username is already taken");  
 }  
 else  
 setStatus(st);  
 }  
 public boolean registerUser(User user){  
 registerUserTest(user);  
 if(getStatus().equals("User was registered successfully")){  
 if(userInserter.insertUser(user)) {  
 setStatus("User was registered successfully");  
 return true;  
 }  
 setStatus("Couldn't register user");  
  
 }  
 return false;  
 }  
}

* **registerUserTest**: it’s used to check if user’s fields are valid using [DataValidation](#_DataValidation_Class) Class, if yes, check if

the user is already exist using [RetrievingData](#_RetrievingData_Class) Class, if not, we can register it successfully.

* **registerUser**: the main method in this class, it uses ***registerUserTest*** to check the availability to register the user

then use an instance of [InsertingData](#_InsertingData_Class) Class to add the user to the database.

# Task1

* Features (1, 2) for Database
* Features (3, 4) for Authentication

## Tester Classes

### DatabaseTester

@RunWith(Cucumber.class)  
@CucumberOptions(features = "src/test/resources/databaseFeatures",  
 monochrome = true,snippets = CucumberOptions.SnippetType.*CAMELCASE*,

glue = {"database"})  
public class DatabaseTester {  
  
}

### AuthenticationTester

@RunWith(Cucumber.class)  
@CucumberOptions(features = "src/test/resources/authenticationFeatures",  
 monochrome = true,snippets = CucumberOptions.SnippetType.*CAMELCASE*,

glue = {"authentication"})  
public class AuthenticationTester {

}

## Feature 1: Database Connection

* Test the connection of the database given (port, database name, username, password).

### Scenarios

Feature: Connecting to a given database  
  
 Scenario: Successful connection  
 When I want to connect to database  
 And I fill in **'port'** with **'3306'** And I fill in **'databaseName'** with **'caraccessoriescompany'** And I fill in **'username'** with **'root'** And I fill in **'password'** with **'12345678password'** Then I should see **"Connected to the database successfully"** for connection

Scenario: Failure connection  
 When I want to connect to database  
 And I fill in **'port'** with **'3300'** And I fill in **'databaseName'** with **'invalidName'** And I fill in **'username'** with **'invalidRoot'** And I fill in **'password'** with **'invalidPassword'** Then I should see **"Couldn't connect to the database"** for connection

### Steps definition Class

public class DBConnectionTest {

private String databaseName, username, password;  
 private int port;  
 private DatabaseConnection testConnection;

@When("I want to connect to database")  
 public void iWantToConnectToDatabase() {  
 assert(true);  
 }

* **databaseName, username, password, port**: are used as parameters to set the connection
* **testConnection**: instance of [DatabaseConnection](#_DatabaseConnection_Class) Class (using ***getStatus*** as a result to database connection in step definition class)

@When("I fill in {string} with {string}")  
 public void iFillInWith(String field, String input) {  
 if(field.equals("databaseName"))  
 databaseName = input;  
 else if (field.equals("username"))  
 username = input;  
 else if (field.equals("password"))  
 password = input;  
 else if (field.equals("port"))  
 port = Integer.*parseInt*(input);  
 else assert(false);  
 assert(true);  
 }

@Then("I should see {string} for connection")  
 public void iShouldSee(String message) {  
 testConnection = new DatabaseConnection(port, databaseName, username, password);  
 String status = testConnection.getStatus();  
 *assertEquals*(status, message);  
 }  
}

* **iFillInWith**: set the value of each field necessary to establish the connection, assert false if invalid field is entered.
* **iShouldSee**: compare the expected result with the status of ***testConnection***.

### Result

## Feature 2: Retrieving from DB

### Scenarios

Feature: Retrieve data from database  
  
 Scenario Outline: I retrieve from users entity  
 Given I'm connected to a database  
 When I fill in condition with **"<condition>"** And I want to retrieve **'users'** Then I should see **"<message>"** for retrieving data  
 And close the connection  
 Examples:  
 | **condition** | **message** |  
 | | **Retrieving users successfully** |  
 | **username = 'rihamkatout'** | **Retrieving users successfully** |  
 | **username = rihamkatout** | **Error while retrieving users from database** |  
 | **username = 'nousername'** | **Retrieving users successfully** |  
 | **phone = '0599119482'** | **Retrieving users successfully** |  
 | **email = 'rihamk@gm.c'** | **Retrieving users successfully** |  
 | **firstName = 'Riham'** | **Retrieving users successfully** |  
 | **lastName = 'Katout'** | **Retrieving users successfully** |  
 | **username = 'rihamkatout' AND firstName = 'Riham'** | **Retrieving users successfully** |  
 | **latName = 'Katout'** | **Error while retrieving users from database** |  
 | **lastName = 'Katout' ANDD firstName = 'Riham'** | **Error while retrieving users from database** |  
  
  
 Scenario Outline: I retrieve from addresses entity  
 Given I'm connected to a database  
 When I fill in condition with **"<condition>"** And I want to retrieve **'addresses'** Then I should see **"<message>"** for retrieving data  
 And close the connection  
 Examples:  
 | **condition** | **message** |  
 | | **Retrieving addresses successfully** |  
 | **city = 'Nablus'** | **Retrieving addresses successfully** |  
 | **country = 'Palestine'** | **Retrieving addresses successfully** |  
 | **country = 'FakeCountry'** | **Retrieving addresses successfully** |  
 | **street = 'Sikkah street'** | **Retrieving addresses successfully** |  
 | **country = 'Palestine' AND city = 'Nablus'** | **Retrieving addresses successfully** |  
 | **country = 'Palestine' ANDDDD city = 'Nablus'** | **Error while retrieving addresses from database** |

\* We will use the same feature to test retrieving other entities in the future\*

### Steps definition Class

public class DBRetrievingTest {  
 private String condition, status;  
 private DatabaseConnection connection;  
 private RetrievingData retrievingData;  
  
 @BeforeAll  
 @Given("I'm connected to a database")  
 public void iMConnectedToADatabase() {  
 connection = new DatabaseConnection();  
 retrievingData = new RetrievingData(connection.getCon());  
 }

* **connection**: temporary connection to database
* **retrievingData**: instance of [RetrievingData](#_RetrievingData_Class) Class.
* **condition**: to store the condition when user enters it.
* **status**: to store the result of retrieving then use it in ***iShouldSee*** step

@When("I fill in condition with {string}")  
 public void iFillInConditionWith(String string) {  
 condition = string;  
 }  
 @When("I want to retrieve {string}")  
 public void iWantToRetrieve(String entity) {  
 if(entity.equals("users")) {  
 retrievingData.selectUsers(condition);  
 status = retrievingData.getStatus();  
 }  
 else if(entity.equals("addresses")){  
 retrievingData.selectAddresses(condition);  
 status = retrievingData.getStatus();  
 }  
 else  
 status = "Error while retrieving from database";  
 }  
 @Then("I should see {string} for retrieving data")  
 public void iShouldSee(String message) {  
 *assertEquals*(status, message);  
 }  
 @AfterAll  
 @Then("close the connection")  
 public void closeTheConnection() throws SQLException {  
 connection.getCon().close();  
 }  
}

* **iWantToRetrieve**: call the suitable function according to entered entity, then store the result in status.
* **iShouldSee**: compare the result with the expected value, if they are equal assert true, otherwise, false.
* **closeTheConnection**: close the connection after finishing all steps.

### Result

## Feature 3: Login

### Scenarios

Feature: Login feature  
 I want to login to car accessories  
  
 Scenario Outline: login scenarios  
 Given user is connected to the database  
 When he fills in **'username'** with **'<username>'** for login  
 And he fills in **'password'** with **'<password>'** for login  
 And user clicks on login  
 Then user should see **'<message>'** for login  
 And close the connection  
  
 Examples:  
 | **username** | **password** | **message** |  
 | **rihamkatout** | **1234\*\*Aa** | **Valid username and password** |  
 | **rihamkatout2** | **1234\*\*Aa** | **Valid username and password** |  
 | **rihamkatout3** | **1234\*\*Aa** | **Valid username and password** |  
 | **rihamkatout9** | **123456** | **Invalid username or password** |  
 | **rihamkatout** | **12de456** | **Invalid username or password** |  
 | | **12de456** | **Invalid username or password** |  
 | **rihamkatout** | | **Invalid username or password** |

### Steps definition Class

public class LoginTester {  
  
 private String status, username, password;  
 private DatabaseConnection connection;  
 private Login login;  
  
  
 @BeforeAll  
 @Given("user is connected to the database")  
 public void userIsConnectedToTheDatabase() {  
 connection = new DatabaseConnection();  
 login = new Login(connection.getCon());  
 }

* **connection**: temporary connection to database.
* **status**: to store the result of retrieving then use it in ***userShouldSee*** step.
* **username, password**: to store input from user.
* **login**: instance of [Login](#_Login_Class) Class to test logging in.

@When("he fills in {string} with {string} for login")  
 public void heFillsInWith(String field, String input) {  
 if(field.equals("username"))  
 username = input;  
 else  
 password = input;  
 }

@When("user clicks on login")  
 public void userClicksOnLogin() {  
 login.loginUser(username, password);  
 status = login.getStatus();  
 }

@Then("user should see {string} for login")  
 public void userShouldSee(String message) {  
 *assertEquals*(status,message);  
 }

@AfterAll  
 @Then("close the connection")  
 public void closeTheConnection() throws SQLException {  
 connection.getCon().close();  
 }  
}

### Result

## Feature 4: Register

### Scenarios

Feature: User sign-Up  
 I want to sign up for car accessories  
  
 Scenario Outline: User sign-up with various inputs  
 When user is in sign-up page  
 And he fills in **'username'** with **"<Username>"** for register  
 And he fills in **'firstName'** with **"<FirstName>"** for register  
 And he fills in **'lastName'** with **"<LastName>"** for register  
 And he fills in **'phoneNumber'** with **"<PhoneNumber>"** for register  
 And he fills in **'password'** with **"<Password>"** for register  
 And he fills in **'email'** with **"<Email>"** for register  
 And he submits the registration form  
 Then he should see **"<Message>"** for register  
  
 Examples:  
 | **Username**| **FirstName**| **LastName**| **PhoneNumber**| **Password** | **Email** | **Message** |  
 | **shahd28** | **Shahd** | **Hamad** | **0595014020** | **1234\*\*Aa** | **shahd22@gmail.com**| **User was registered successfully** |  
 | **shahd18** | **Shahd** | **Hamad** | **059501402** | **1234\*\*Aa** | **shahd18@gmail.com**| **Invalid phone number** |  
 | **shahd12** | **Shahd** | **Hamad** | **059501402a** | **1234\*\*Aa** | **shahd12@gmail.com**| **Invalid phone number** |  
 | **shahd28** | **Shahd** | **Hamad** | **0595014020** | **weakPassword**| **shahd28@gmail.com**| **Invalid password** |  
 | **shahd20** | **Shahd** | **Hamad** | **0595014020** | **1234\*\*Aa** | **shahd22ail.com** | **Invalid email address** |  
 | **shahd11** | **Shahd** | **Hamad** | **0595014020** | **1234\*\*Aa** | **shahd11@gmail.com**| **Username is already taken** |  
 | | **Shahd** | **Hamad** | **0595014020** | **1234\*\*Aa** | **shahd29@gmail.com**| **Username can't be empty** |  
 | **shahd19** | | **Hamad** | **0595014020** | **1234\*\*Aa** | **shahd29@gmail.com**| **First name can't be empty** |  
 | **shahd19** | **Shahd** | | **0595014020** | **1234\*\*Aa** | **shahd29@gmail.com**| **Last name can't be empty** |  
 | **shahd19** | **Shahd** | **Hamad** | | **1234\*\*Aa** | **shahd29@gmail.com**| **Phone number can't be empty** |  
 | **shahd19** | **Shahd** | **Hamad** | **0595014020** | | **shahd29@gmail.com**| **Password can't be empty** |  
 | **shahd19** | **Shahd** | **Hamad** | **0595014020** | **1234\*\*Aa** | | **Email address can't be empty** |

### Steps definition Class

public class SignupTester {  
 private Register userRegisterer;  
 private User user;  
 private DatabaseConnection connection;  
 private String status;  
  
 @BeforeAll  
 @When("user is in sign-up page")  
 public void userIsOnTheSignUpPage() {  
 connection = new DatabaseConnection();  
 userRegisterer = new Register(connection.getCon());  
 user = new User();  
 }

* **connection**: temporary connection to database.
* **status**: to store the result of retrieving then use it in ***heShouldSee*** step.
* **user**: to store input from user.
* **userRegisterer**: instance of [Register](#_Register_Class) Class.

@When("he fills in {string} with {string} for register")  
 public void heFillsInWithForRegister(String field, String input) {  
 if(field.equals("username"))  
 user.setUsername(input);  
 else if(field.equals("firstName"))  
 user.setFirstName(input);  
 else if(field.equals("lastName"))  
 user.setLastName(input);  
 else if(field.equals("phoneNumber"))  
 user.setPhoneNumber(input);  
 else if(field.equals("password"))  
 user.setPassword(input);  
 else if(field.equals("email"))  
 user.setEmail(input);  
 else  
 assert(false);  
 assert(true);  
 }  
  
 @When("he submits the registration form")  
 public void heSubmitsTheRegistrationForm() {  
 userRegisterer.registerUserTest(user);  
 status = userRegisterer.getStatus();  
 }  
 @Then("he should see {string} for register")  
 public void heShouldSee(String message) {  
 System.*out*.println(status);  
 *assertEquals*(message, status);  
 }  
  
}

### Result