

|  |  |
| --- | --- |
| **Qualification Name** | Bachelor in Software Engineering (Honours) (Application Development) |
| **Module Name** | Application Development and Process |
| **Assignment Title** | Project |
| **Start Date** | 22 November 2024 |
| **Due Date** | 25 November 2024 |
| **Learner Name** | Nicole Ooi Jinn Faye |
| **Course Lead** | Mr Muhammad Norhadri bIn Mohd Hilmi |

|  |
| --- |
| **Learner declaration** |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  Student signature: Nicole Date:25-11-24 |

**Project Background**

Hi-fi Cars Pte Ltd, a prominent dealership specializing in quality pre-owned vehicles, has commissioned a new initiative to enhance their customer engagement through a modern online platform. Recognizing the growing trend towards digital transactions in the automotive industry, Hi-fi Cars aims to establish a user-friendly Used Car Sales portal. This portal will cater to both individual users looking to buy or sell cars and administrators tasked with managing the platform.

**Project Objectives**

1. **Enhanced User Registration:** Develop a seamless registration process that allows users to create accounts securely and efficiently.
2. **Advanced Search Functionality:** Implement robust search capabilities based on car make, model, registration details, and price range to facilitate quick and precise vehicle discovery.
3. **Detailed Car Listings:** Provide comprehensive car details and images for users to review once they find vehicles of interest.
4. **User Authentication and Management:** Enable secure login functionality for both users and administrators, each with distinct permissions and access levels.
5. **Listing Management:** Allow users to post their cars for sale, manage listings, and update car information as needed.

Project Introduction

The Used Car Sales portal development project aims to transform Hi-fi Cars Pte Ltd's online presence by offering a dynamic platform for buying and selling pre-owned vehicles. This initiative responds to the increasing demand for digital solutions in the automotive retail sector, providing a user-centric interface that enhances the overall customer experience. By integrating advanced search functionalities, streamlined user registration, and secure authentication mechanisms, the portal will empower users to navigate through a wide selection of cars efficiently.

Through collaboration with Hi-fi Cars Pte Ltd, the project will prioritize usability, security, and scalability to ensure the portal meets current industry standards and supports future growth. By facilitating direct interactions between sellers and potential buyers, the platform will streamline transactions and foster a transparent marketplace for used cars. The project's success will be measured by user engagement metrics, platform stability, and feedback from both users and administrators, ensuring continuous improvement and optimization of the portal's functionalities.

Task 1. Risk- Based Testing Concepts and Requirements

1. Risk-Based Testing Concepts and Requirements

Risk-based testing is a systematic approach that prioritizes testing efforts based on the potential risks to the system. This methodology ensures that testing resources are allocated efficiently to areas of the system that pose the highest risk of failure or impact. Here's how you can structure and explain this section:

****a. Stages of Risk-****Based ****Testing Model:****

Risk-driven testing typically involves several key stages:

* **Risk Identification and Assessment Stage:** This stage involves identifying potential risks within the system. Risks can include functional risks (related to the system's functionality not meeting requirements), security risks (related to vulnerabilities and threats), performance risks (related to system response times and scalability), and others. The goal is to comprehensively assess these risks to understand their impact on the system if they were to occur.
* **Test Strategy Formulation Stage:** Once risks are identified and assessed, the next stage is to formulate a test strategy. This strategy outlines how testing will be conducted to address identified risks effectively. It includes deciding which types of tests (e.g., functional testing, security testing, performance testing) will be prioritized based on the severity and likelihood of risks.

****b. Types of Risks and Their Impact:****

It's important to categorize risks based on their nature and potential impact on the system:

* **Functional Risks:** These risks involve the possibility of system functionalities not performing as expected, leading to incorrect outputs or failures in meeting user requirements.
* **Security Risks:** These risks involve vulnerabilities in the system that could be exploited by malicious actors, potentially leading to unauthorized access, data breaches, or service disruptions.
* **Performance Risks:** These risks involve issues related to system performance, such as slow response times, high resource consumption, or inability to handle expected load levels.

Understanding these risk types helps in prioritizing testing efforts. For example, critical security risks might require more intensive security testing measures like penetration testing, while functional risks might necessitate comprehensive functional testing to ensure all user requirements are met.

****c. Benefits of Risk-Based Testing:****

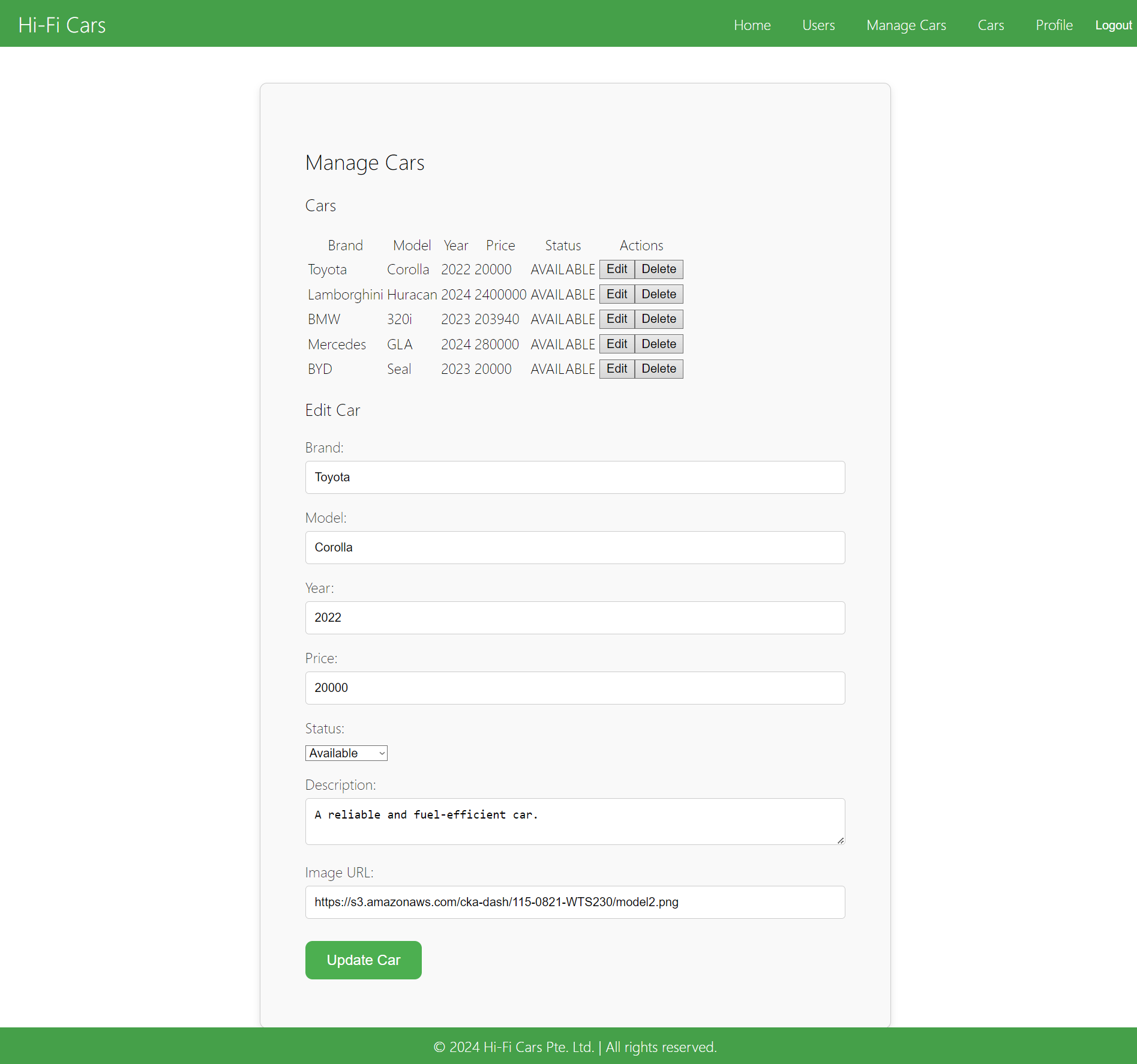
Adopting a risk-driven testing approach offers several benefits:

* **Focused Testing Efforts:** By concentrating testing efforts on high-risk areas identified during risk assessment, teams can increase the likelihood of detecting critical defects early in the development lifecycle.
* **Cost Efficiency:** Risk-driven testing optimizes resource allocation by reducing unnecessary testing in low-risk areas, thereby saving time and costs associated with exhaustive testing approaches.
* **Enhanced System Stability:** Addressing high-impact risks proactively during testing reduces the likelihood of critical issues affecting system stability in production, leading to improved overall system reliability and user satisfaction.

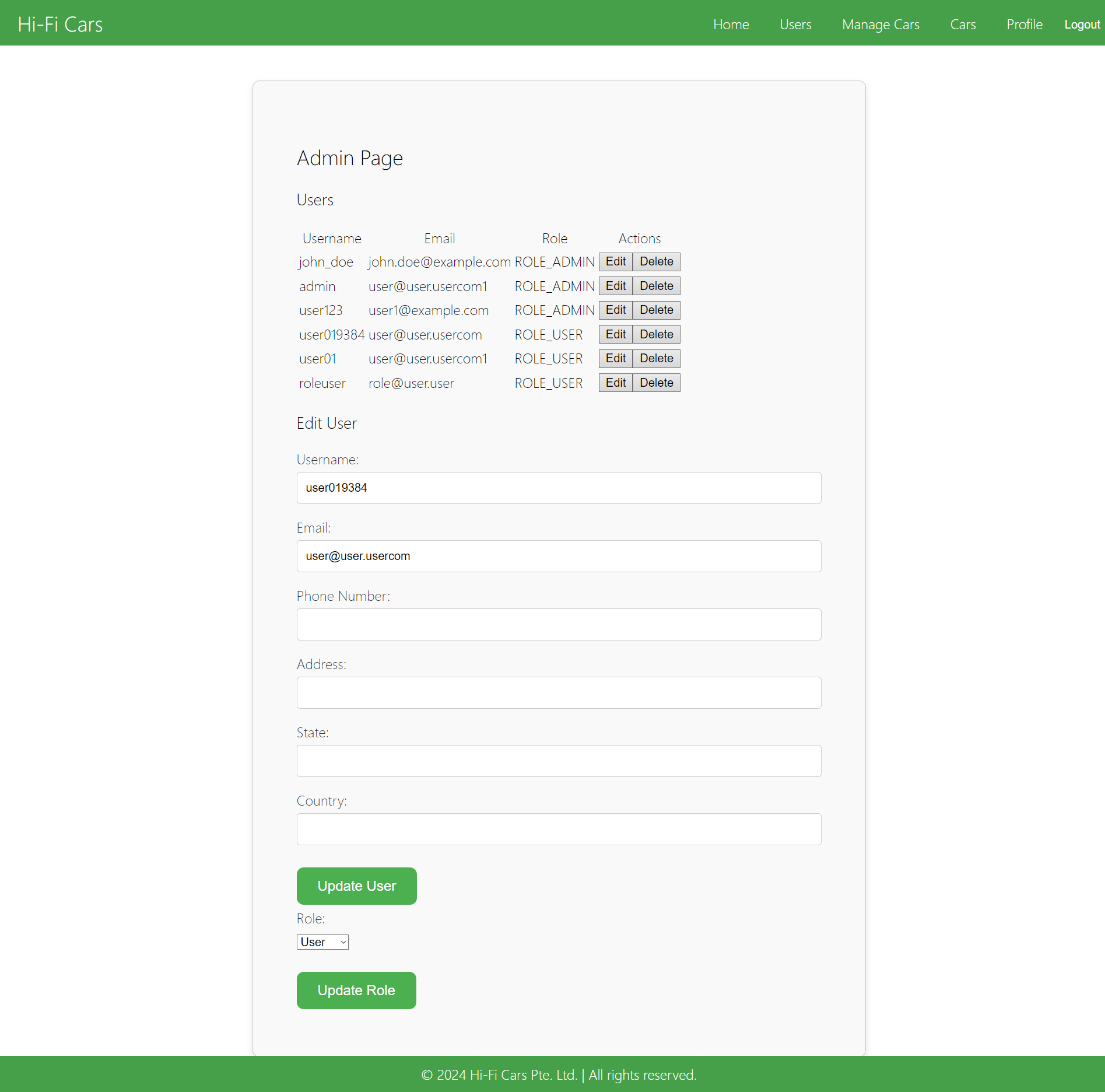
Task 2 Establish the test phases of security testing with its prioritization

**Administrator permissions**

Administrator users can register users, edit a user’s details and set permissions, and delete users

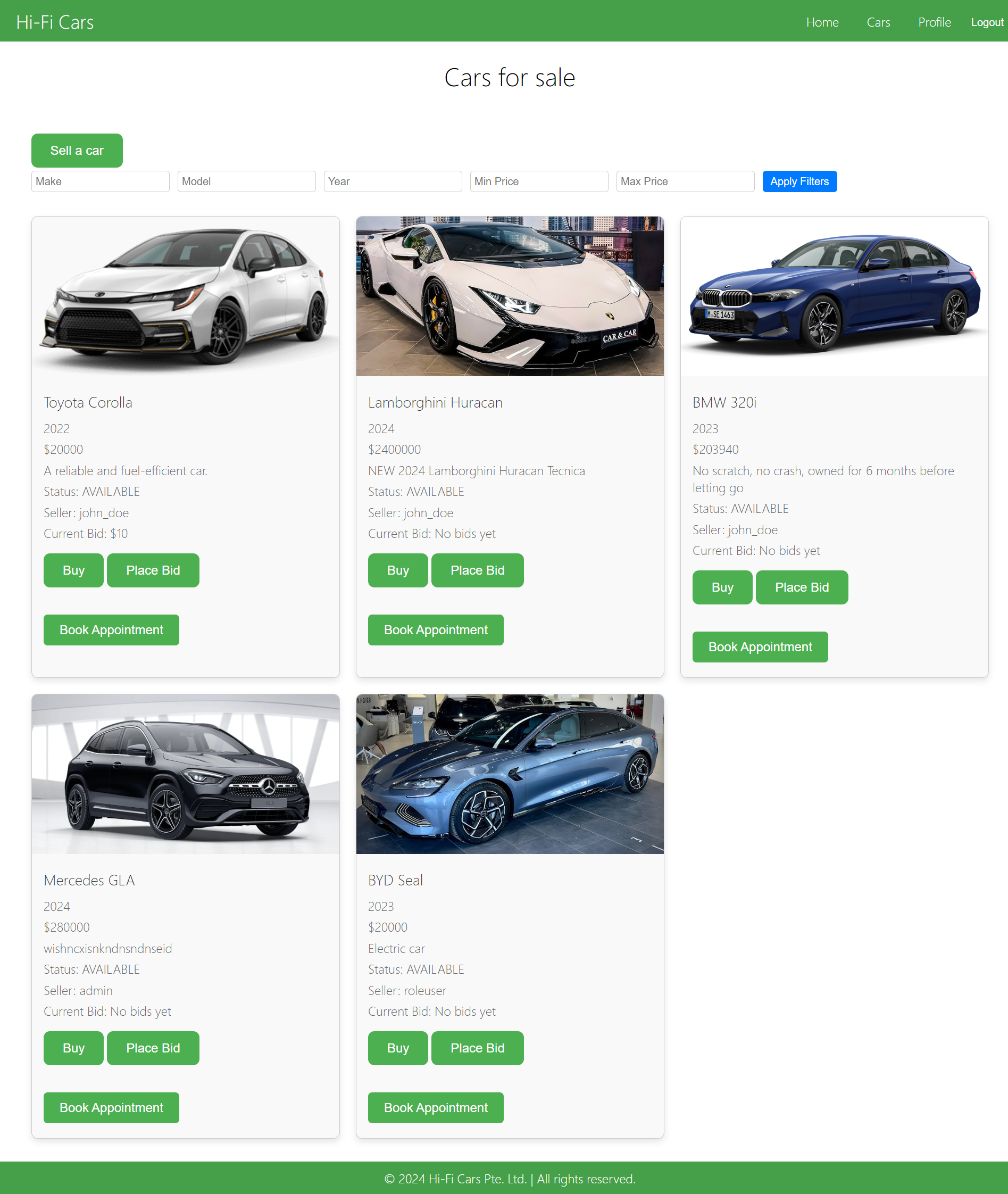


Administrator users can access all site functions as well as, update and set availability of cars, and delete cars



**User permissions**

Users can register a new user, edit their profile, sell a car, buy cars, bid for cars, and book appointments only.

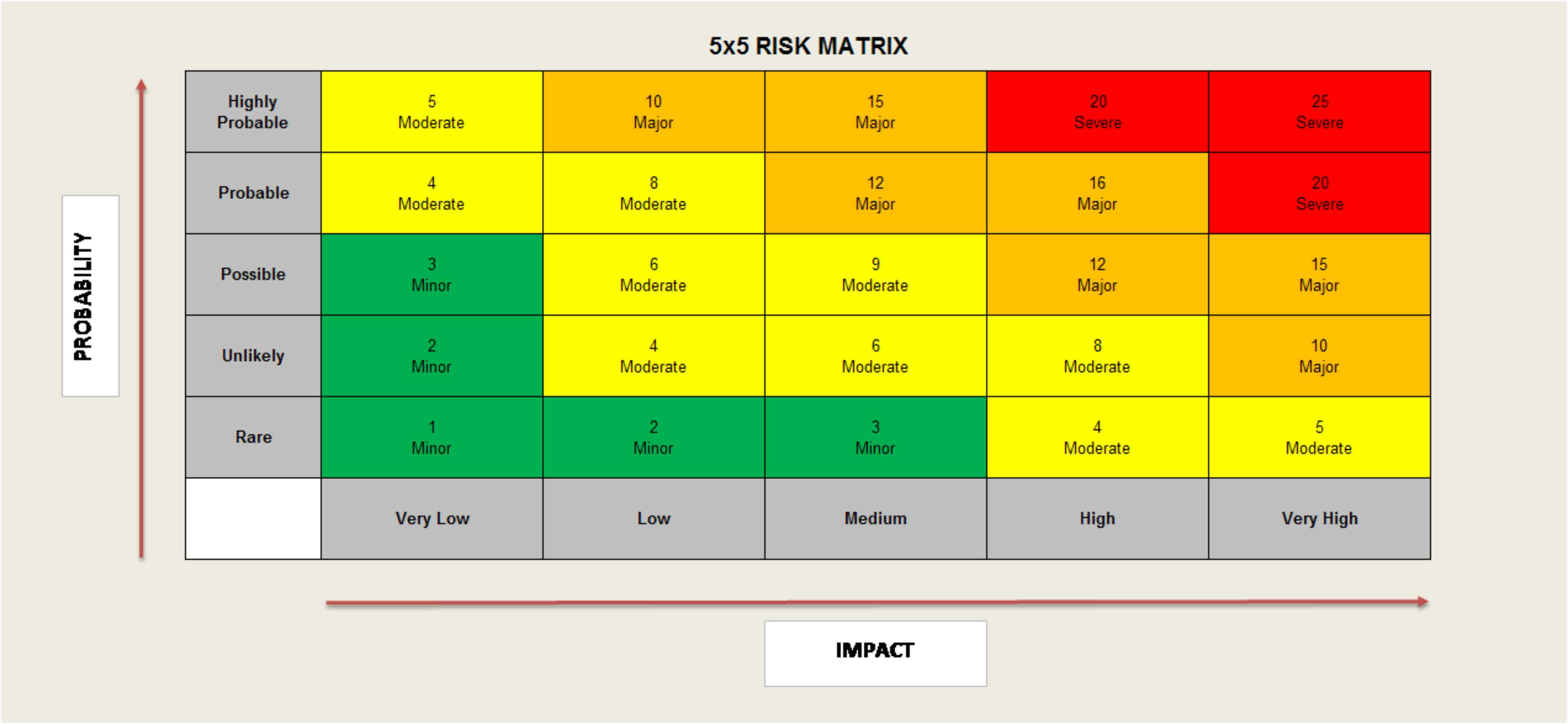


Task 3 Develop the detailed test plan with test cases based on quantitative risk analysis

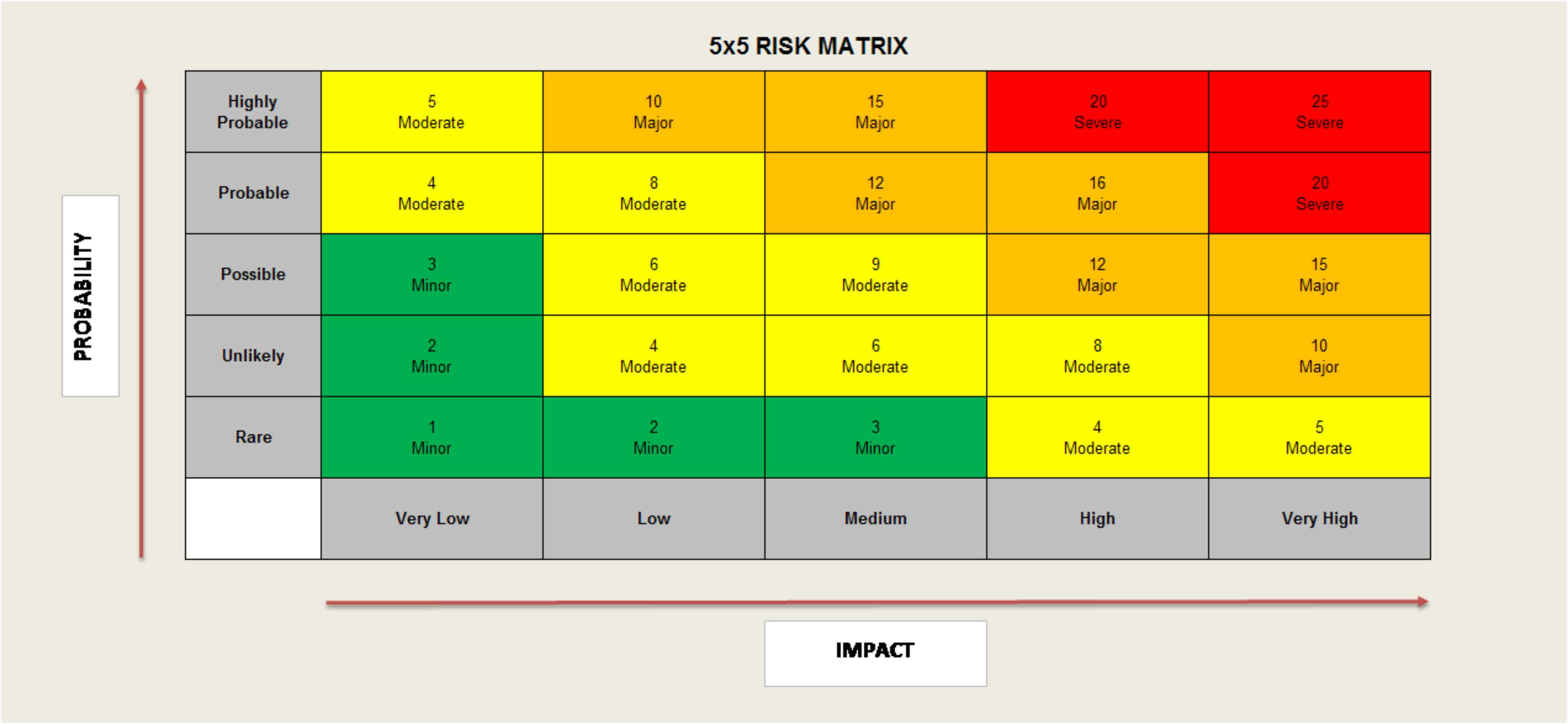
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Test Scenario** | **Test Objective** | **Test Case ID** | **Test Case** | **Test Priority** | **Risk** | **Techniqu**e | **Expected Results** |
| 1 | S001 | Registration Flow | RTC001 | Verify registration page loads correctly | High | Medium | User acceptance testing | Registration page should load without errors |
| 2 | S001 | Registration Flow | RTC002 | Registration unit test - valid inputs | High | Medium | Unit Testing | User should be successfully registered and redirected to the welcome page |
| 3 | L001 | Login Flow | LTC001 | Verify login page loads correctly | High | High | User acceptance testing | Login page should load without errors |
| 4 | L001 | Login Flow | LTC002 | Login unit test - valid credentials | High | High | Unit Testing | User should be successfully logged in and redirected to the home page |
| 5 | AD001 | Add Car Flow | TC\_ADD\_CAR\_001 | Verify adding a car with valid details | High | Medium | User acceptance testing | Car should be successfully added to the system and listed in the car inventory |
| 6 | UCP001 | User Change Profile | TC\_CHANGE\_EMAIL\_001 | Verify user can change email with valid new email | Medium | Low | User acceptance testing | Email should be successfully updated and reflected in the user's profile |
| 7 | UCP001 | User Change Profile | TC\_CHANGE\_PASSWORD\_001 | Verify user can change password with valid current and new passwords | Medium | Low | User acceptance testing | Password should be successfully updated and user notified of the change |
| 8 | TA001 | Toggle Car Activation Status | TC\_TOGGLE\_ACTIVATE\_001 | Verify toggling availability status of a car | High | High | User acceptance testing | Car's availability status should be successfully toggled and reflected immediately |

Task 4 Create a test risk matrix with the evaluation of the impact

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Test Scenario** | **Test Case ID** | **Impact** | **Probability** | **Risk level** | **Description** |
| 1 | Registration Flow | RTC001 | High (4) | Medium (3) | Major (12) | Incorrect registration page load affects new user registration, hindering users from accessing the system. |
| 2 | Registration Flow | RTC002 | High (4) | Medium  (3) | Major (12) | Registration function failure prevents users from creating new accounts, affecting user growth. |
| 3 | Login Flow | LTC001 | High  (4) | High  (4) | Major (16) | Incorrect login page load affects user access to the system, hindering users from logging in. |
| 4 | Login Flow | LTC002 | High  (4) | High  (4) | Major (16) | Login failure prevents users from accessing the system, affecting user experience and business operations. |
| 5 | Add Car Flow | TC\_ADD\_CAR\_001 | High  (4) | Medium  (3) | Major (12) | Failure to add a car affects user management of car inventory, impacting the system's core functionality. |
| 6 | User Change Profile | TC\_CHANGE\_EMAIL\_001 | Medium  (3) | Low  (2) | Moderate (6) | Inability to change email affects user information updates and notification delivery. |
| 7 | User Change Profile | TC\_CHANGE\_PASSWORD\_001 | Medium  (3) | Low (2) | Moderate (6) | Inability to change password affects account security, increasing security risks. |
| 8 | Toggle Car Availability Status | TC\_TOGGLE\_ACTIVATE\_001 | High  (4) | High  (4) | Major (16) | Failure to toggle car availability status affects user management of car visibility and status, impacting the system's core functionality. |



The 5x5 risk matrix is used for risk assessment and prioritization. Risk ratings range from 1 to 25, determined by multiplying risk probability with risk impact. Ratings 1 to 3 indicate minor risk levels, 4 to 9 denote moderate risk levels, 10 to 16 signify major risk levels, and 20 and 25 represent severe risk levels, which are the most critical.

Task 5 Design suitable RBT cycle content and prioritize further test cycles

TC\_CHANGE\_EMAIL\_001

TC\_CHANGE\_PASSWORD\_001

RTC001

RTC002

TC\_ADD\_CAR\_001

LTC001

LTC002

TC\_TOGGLE\_ACTIVATE\_001

Task 6 Communicate the details of the strategy and test planning to the key stakeholders with proper justifications

**Strategy and Test Planning Communication**

Dear Stakeholders,

I am pleased to present the strategy and test planning for our upcoming software testing phase. Below, you will find the detailed test scenarios, objectives, priorities, risks, techniques, and expected results for critical functionalities in our system:

1. **Registration Flow**
   * **RTC001 - Verify registration page loads correctly**
     + **Priority**: High
     + **Risk**: Medium
     + **Technique**: User acceptance testing
     + **Objective**: Ensure the registration page loads without errors to facilitate smooth user onboarding.
     + **Expected Results**: Registration page loads correctly without any issues.
   * **RTC002 - Registration unit test - valid inputs**
     + **Priority**: High
     + **Risk**: Medium
     + **Technique**: Unit Testing
     + **Objective**: Validate that users can successfully register with valid inputs.
     + **Expected Results**: Users are registered successfully and redirected to the welcome page.
2. **Login Flow**
   * **LTC001 - Verify login page loads correctly**
     + **Priority**: High
     + **Risk**: High
     + **Technique**: User acceptance testing
     + **Objective**: Ensure the login page loads without errors to facilitate user access.
     + **Expected Results**: Login page loads correctly without any issues.
   * **LTC002 - Login unit test - valid credentials**
     + **Priority**: High
     + **Risk**: High
     + **Technique**: Unit Testing
     + **Objective**: Validate that users can successfully log in with valid credentials.
     + **Expected Results**: Users are logged in successfully and redirected to the home page.
3. **Add Car Flow**
   * **TC\_ADD\_CAR\_001 - Verify adding a car with valid details**
     + **Priority**: High
     + **Risk**: Medium
     + **Technique**: User acceptance testing
     + **Objective**: Ensure cars can be added to the system correctly to manage car inventory effectively.
     + **Expected Results**: Car is added successfully and appears in the car inventory.
4. **User Change Profile**
   * **TC\_CHANGE\_EMAIL\_001 - Verify user can change email with valid new email**
     + **Priority**: Medium
     + **Risk**: Low
     + **Technique**: User acceptance testing
     + **Objective**: Validate that users can update their email addresses successfully.
     + **Expected Results**: Email is updated and reflects correctly in the user's profile.
   * **TC\_CHANGE\_PASSWORD\_001 - Verify user can change password with valid current and new passwords**
     + **Priority**: Medium
     + **Risk**: Low
     + **Technique**: User acceptance testing
     + **Objective**: Ensure users can update their passwords securely.
     + **Expected Results**: Password is updated successfully and user is notified.
5. **Toggle Car Activation Status**
   * **TC\_TOGGLE\_ACTIVATE\_001 - Verify toggling activation status of a car**
     + **Priority**: High
     + **Risk**: High
     + **Technique**: User acceptance testing
     + **Objective**: Validate that car activation status can be toggled effectively.
     + **Expected Results**: Car's activation status is toggled successfully and reflects immediately.

Task 7 Execute Risk-Driven Testing Plan

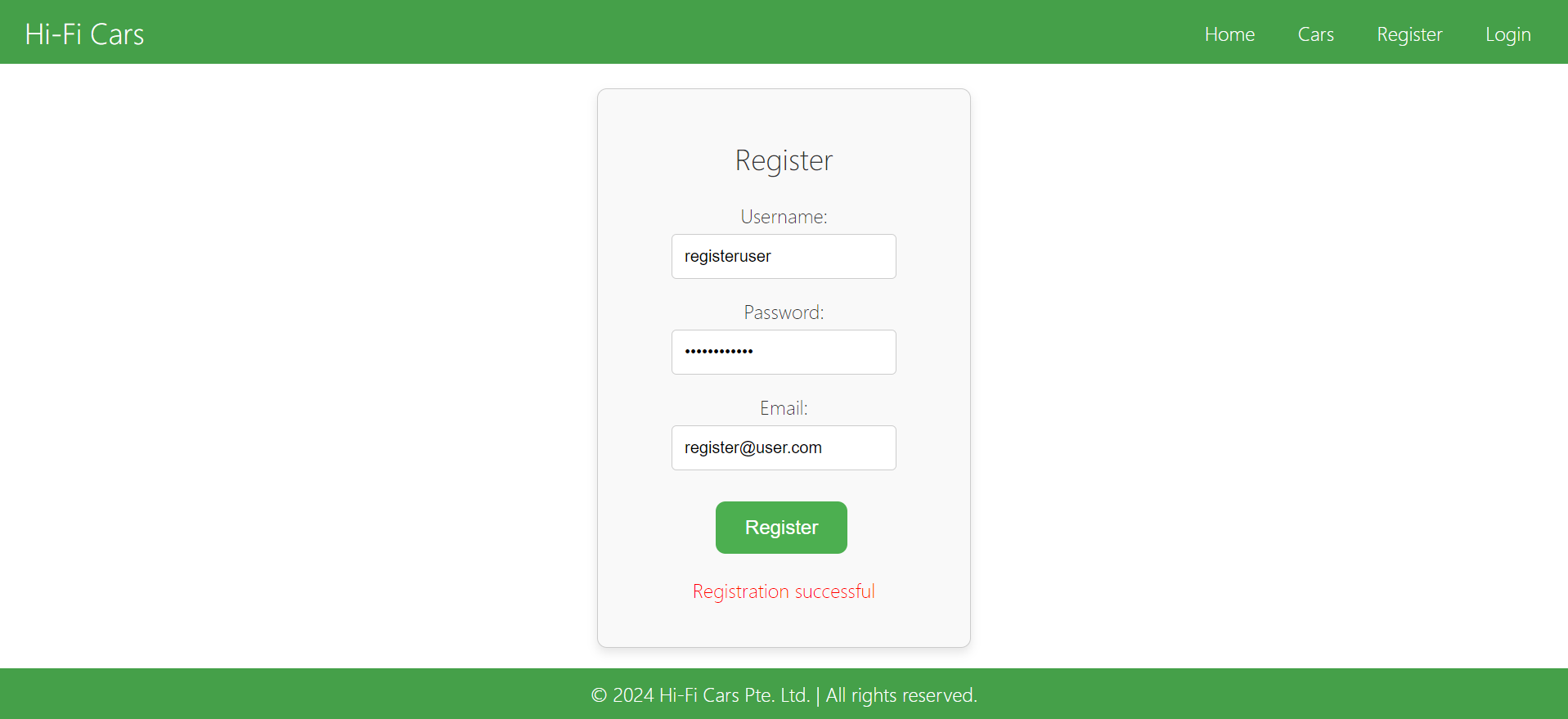
Test Scenario: S001 – Registration Flow

Test Objective: Test that the registration process on the used car sales portal works as expected.

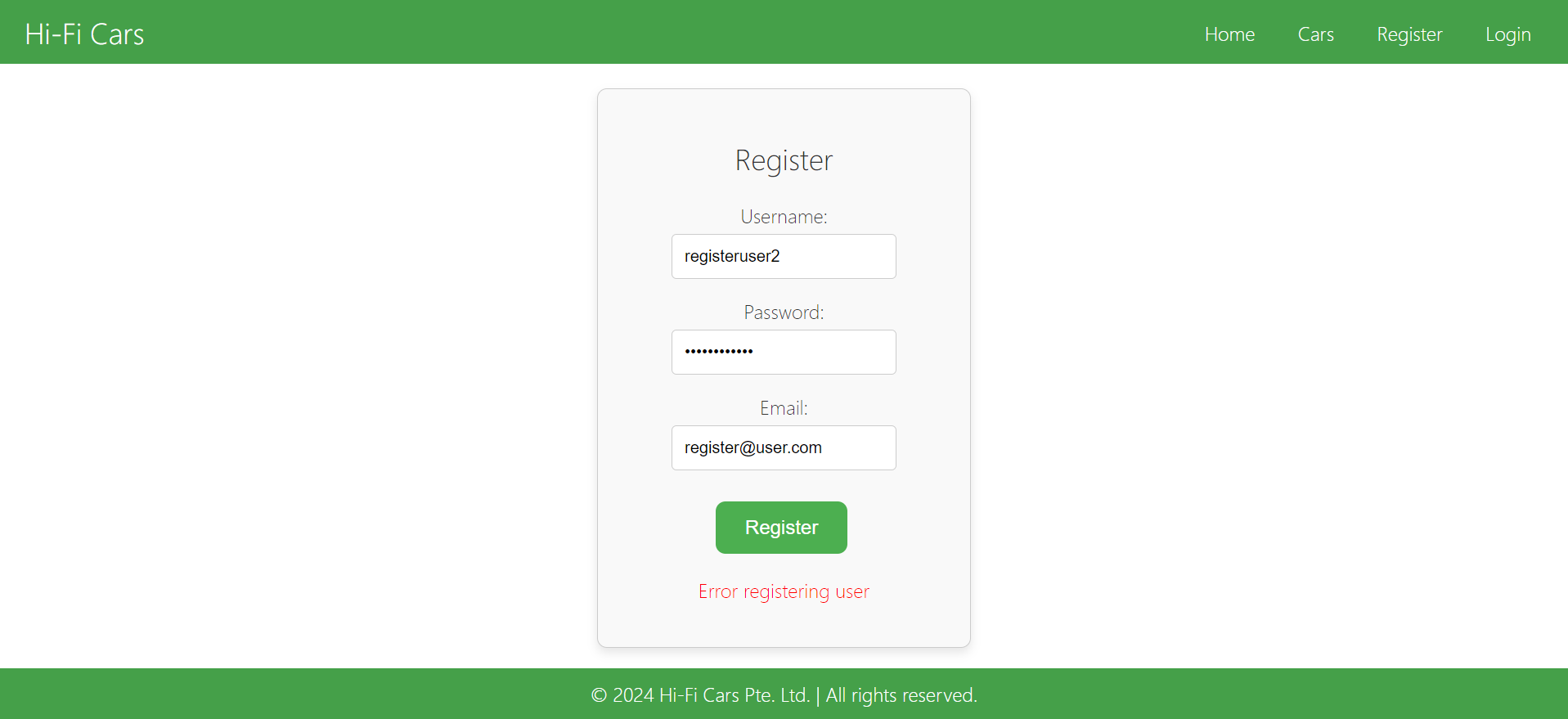
Test Data: email and password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | S001 | Test Scenario Name | Registration Flow | |
| Test Case ID | | TC001 | Risk | Medium | |
| Pre-Requisites | | User Service Setup | Test Priority | High | |
| Test Case Description | | Verify user registration flow | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Run the test script | Execute the user registration script. username: registeruser password: registeruser | User is successfully registered, permissions assigned. | User is successfully registered, permissions assigned. | pass |
| 2 | Check for email duplication | Register with an existing email address. | Error message "Error registering user” is displayed. | Error message "Error registering user” is displayed. | pass |
| 3 | Validate password encryption | Check if the password is encrypted in the database. | Password is stored in encrypted format. | Password is stored in encrypted format. | pass |
| 4 | Verify permission assignment | Check if the "User" permission is assigned to the new user. | User has "User" permission assigned. | User has "User" permission assigned. | pass |

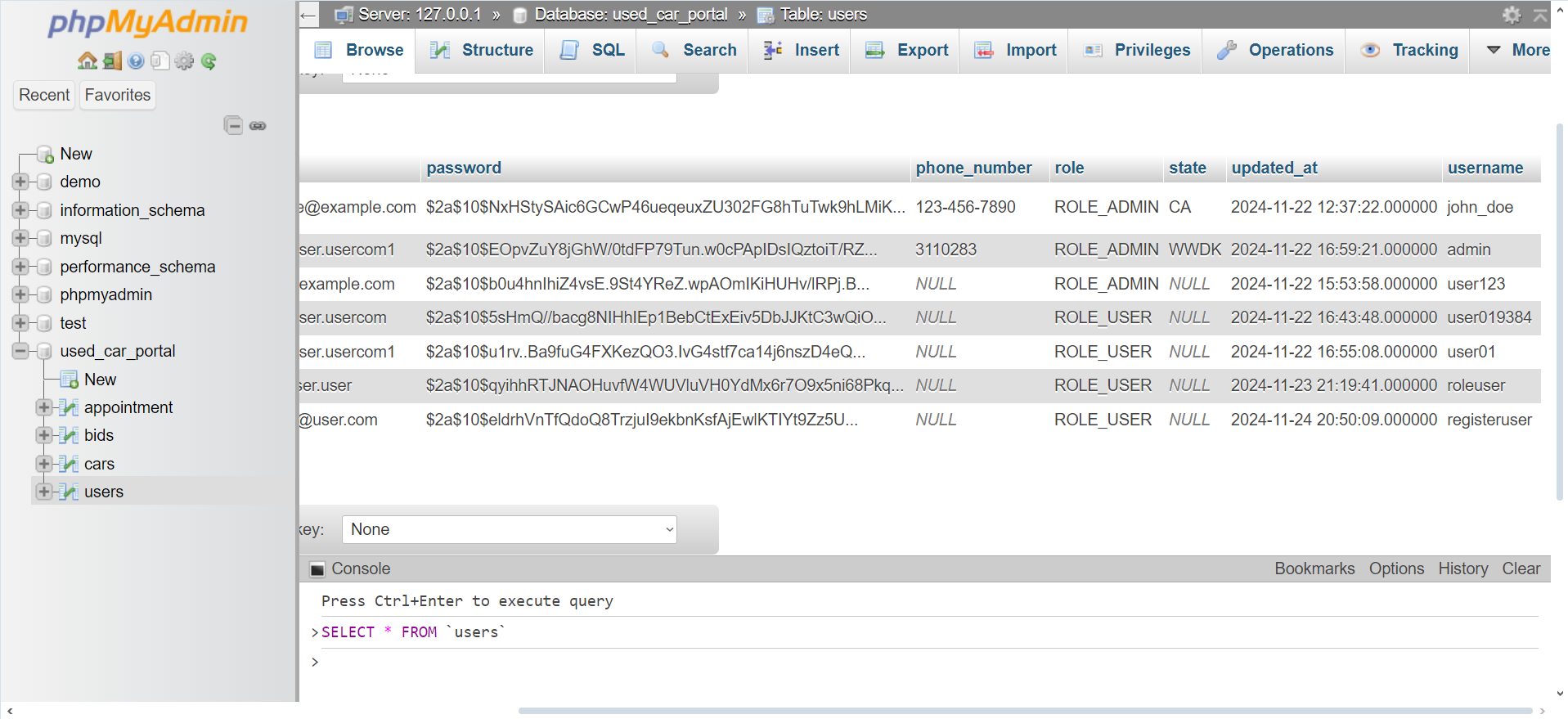
Run the test script



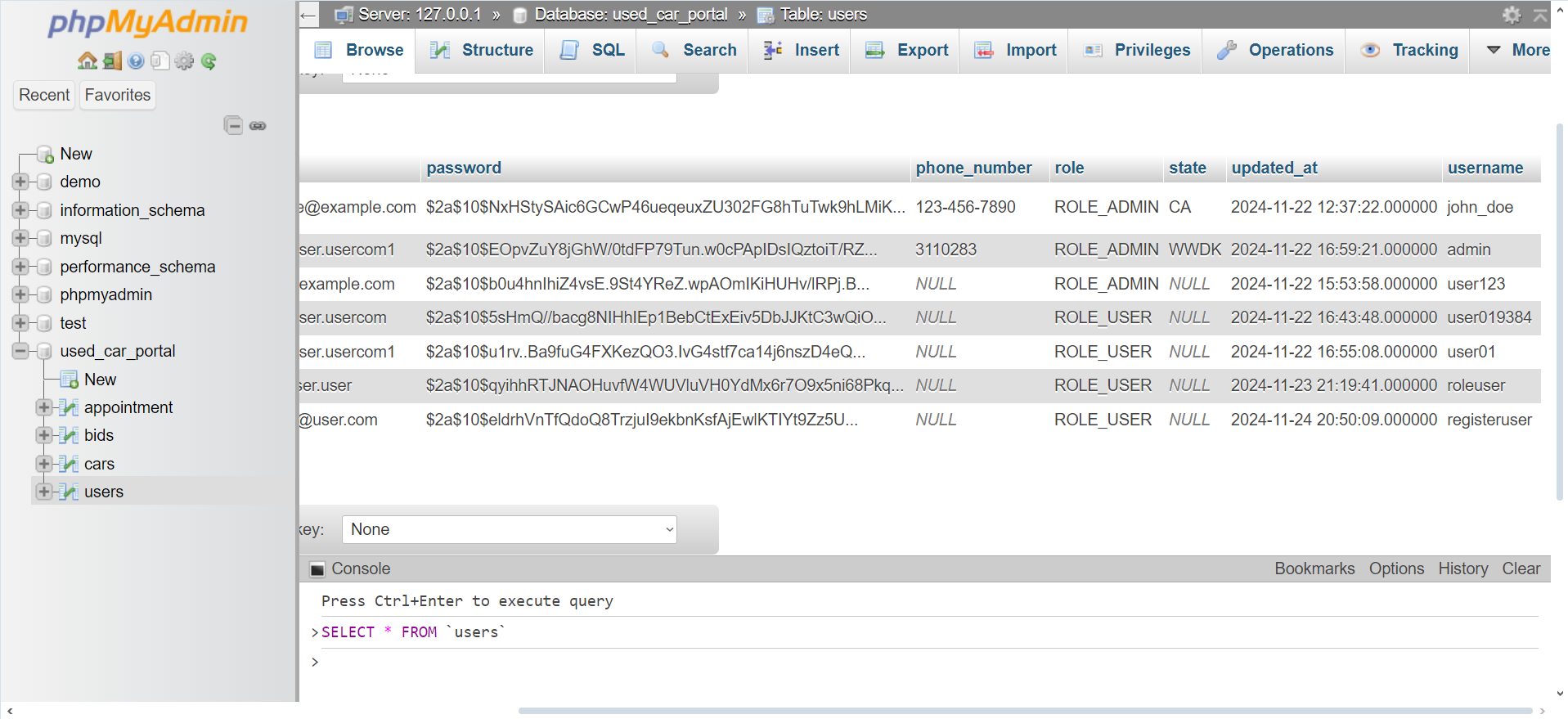
Check for email duplication



Validate password encryption



Verify permission assignment



Test Scenario: S001 – Registration Flow

Test Objective: Test that the registration process on the used car sales portal works as expected.

Test Data: unit test

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | S001 | Test Scenario Name | Registration Flow | |
| Test Case ID | | TC002 | Risk | Medium | |
| Pre-Requisites | | User Service Setup | Test Priority | High | |
| Test Case Description | | Verify user registration flow | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Run the test script | Execute the user registration script. email: [test@example.com](mailto:test@example.com) Password: password123 | User is successfully registered, permissions assigned. | User is successfully registered, permissions assigned. | pass |
| 2 | Check for email duplication | Register with an existing email address. | Error message "Error registering user” is displayed. | Error message "Error registering user” is displayed. | pass |
| 3 | Validate password encryption | Check if the password is encrypted in the database. | Password is stored in encrypted format. | Password is stored in encrypted format. | pass |
| 4 | Verify permission assignment | Check if the "User" permission is assigned to the new user. | User has "User" permission assigned. | User has "User" permission assigned. | pass |

package com.example.dse20503\_project;

import com.example.dse20503\_project.entity.User;

import com.example.dse20503\_project.repository.UserRepository;

import com.example.dse20503\_project.service.UserService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import org.springframework.security.crypto.password.PasswordEncoder;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.ArgumentMatchers.any;

import static org.mockito.Mockito.\*;

public class UserRegistrationTest {

    @Mock

    private UserRepository userRepository;

    @Mock

    private PasswordEncoder passwordEncoder;

    @InjectMocks

    private UserService userService;

    @BeforeEach

    public void setUp() {

        MockitoAnnotations.openMocks(this);

    }

    @Test

    public void testUserRegistration() {

        User user = new User();

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("password123");

        when(userRepository.findByUsername(user.getUsername())).thenReturn(null);

        when(userRepository.findByEmail(user.getEmail())).thenReturn(null);

        when(passwordEncoder.encode(user.getPassword())).thenReturn("encryptedPassword");

        when(userRepository.save(any(User.class))).thenReturn(user);

        User registeredUser = userService.registerUser(user);

        assertNotNull(registeredUser);

        assertEquals("testuser", registeredUser.getUsername());

        assertEquals("test@example.com", registeredUser.getEmail());

        assertEquals("encryptedPassword", registeredUser.getPassword());

        assertEquals("ROLE\_USER", registeredUser.getRole().name());

    }

    @Test

    public void testEmailDuplication() {

        User user = new User();

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("password123");

        when(userRepository.findByEmail(user.getEmail())).thenReturn(user);

        Exception exception = assertThrows(IllegalArgumentException.class, () -> {

            userService.registerUser(user);

        });

        assertEquals("Email already exists", exception.getMessage());

    }

    @Test

    public void testPasswordEncryption() {

        User user = new User();

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("password123");

        when(userRepository.findByUsername(user.getUsername())).thenReturn(null);

        when(userRepository.findByEmail(user.getEmail())).thenReturn(null);

        when(passwordEncoder.encode(user.getPassword())).thenReturn("encryptedPassword");

        when(userRepository.save(any(User.class))).thenReturn(user);

        User registeredUser = userService.registerUser(user);

        assertNotNull(registeredUser);

        assertEquals("encryptedPassword", registeredUser.getPassword());

    }

    @Test

    public void testPermissionAssignment() {

        User user = new User();

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("password123");

        when(userRepository.findByUsername(user.getUsername())).thenReturn(null);

        when(userRepository.findByEmail(user.getEmail())).thenReturn(null);

        when(passwordEncoder.encode(user.getPassword())).thenReturn("encryptedPassword");

        when(userRepository.save(any(User.class))).thenReturn(user);

        User registeredUser = userService.registerUser(user);

        assertNotNull(registeredUser);

        assertEquals("ROLE\_USER", registeredUser.getRole().name());

    }

}

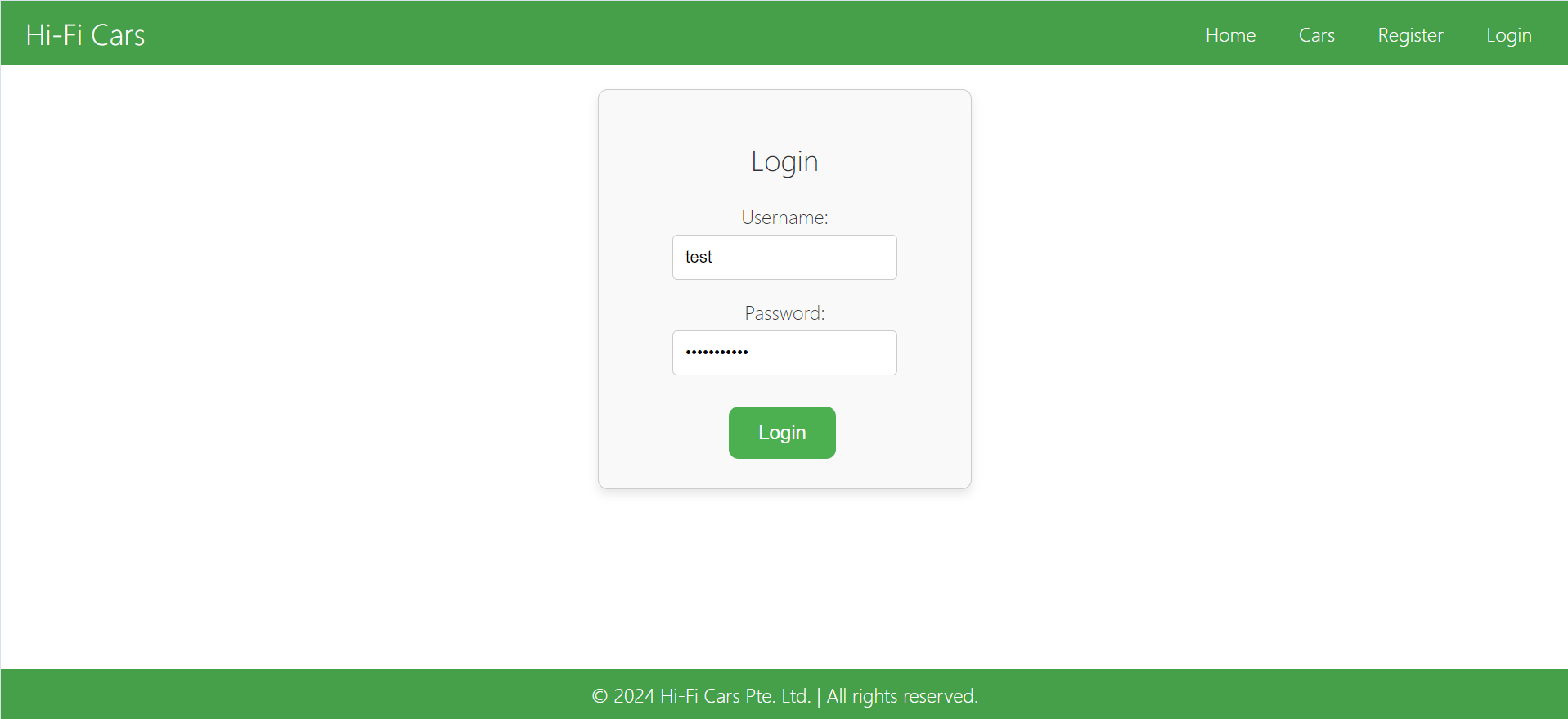
Test Scenario: S001 – Login Flow

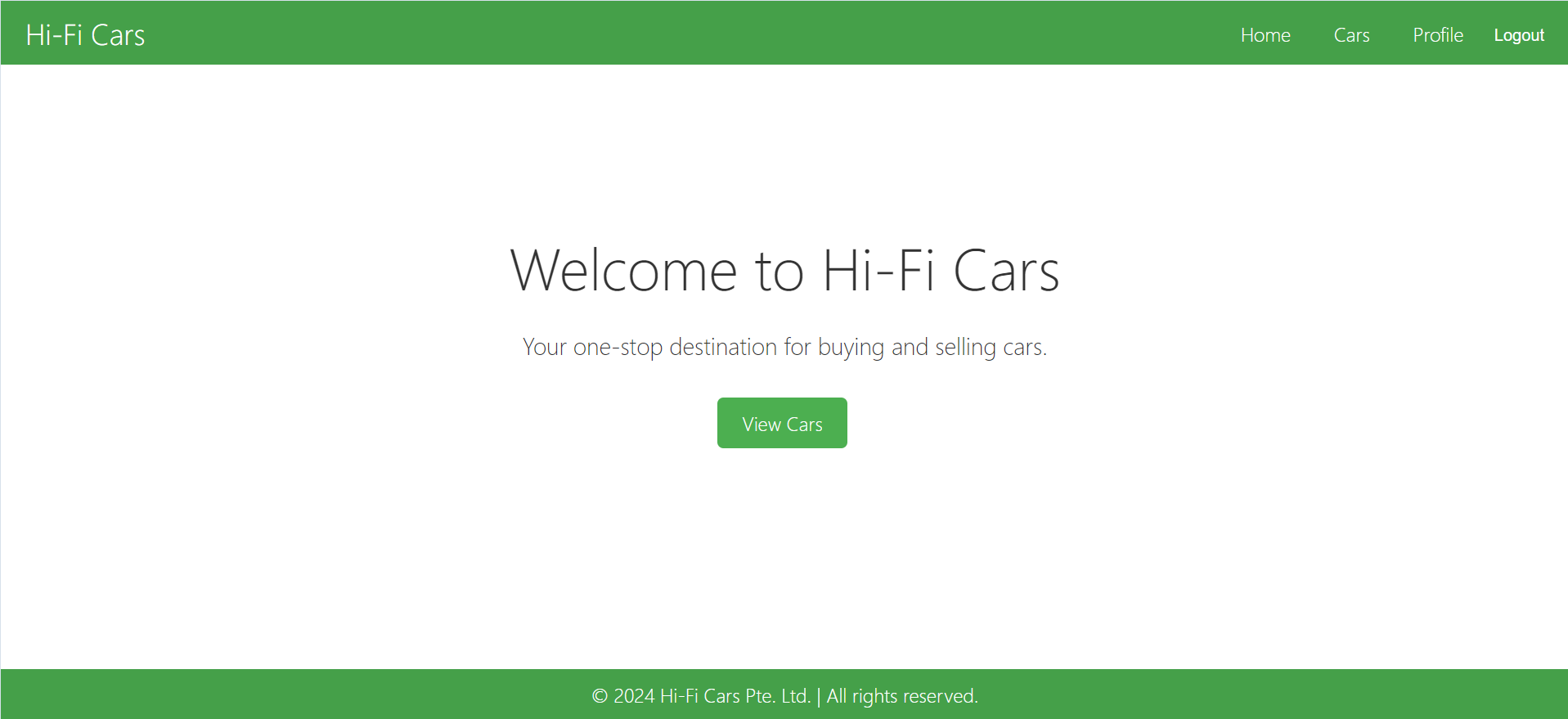
Test Objective: Test that the login process on the used car sales portal works as expected.

Test Data: email and Password

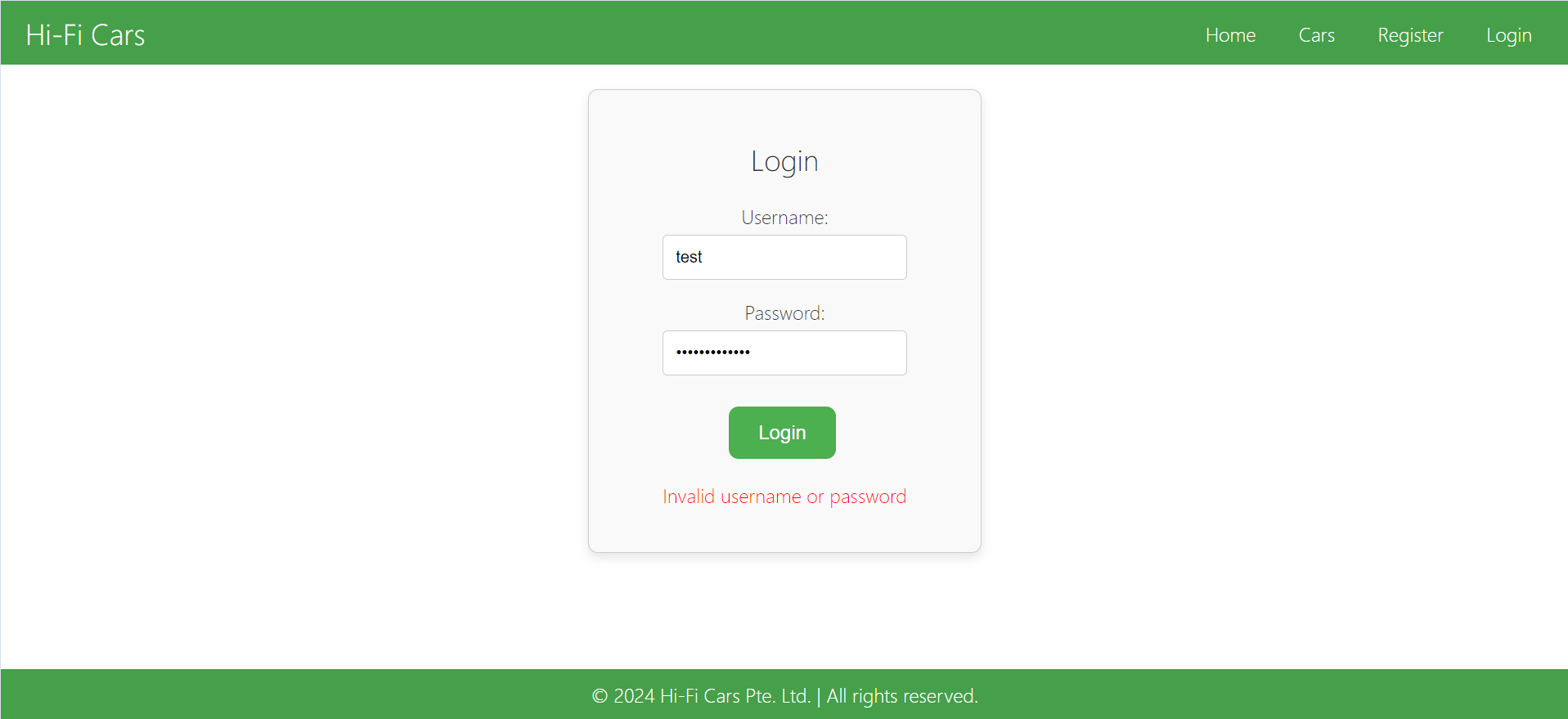
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | L001 | Test Scenario Name | Login Flow | |
| Test Case ID | | TC001 | Risk | High | |
| Pre-Requisites | | User Service and Security Configuration Setup | Test Priority | High | |
| Test Case Description | | Verify user login flow | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Valid user login | Execute the login script with inputs:  username: test  password: password123 | User is successfully logged in and redirected to the home page. | User is successfully logged in and redirected to the home page. | pass |
| 2 | Invalid username login | Execute the login script with inputs:  username: wronguser password: password123 | Error message "Invalid username or password." is displayed. | Error message "Invalid username or password." is displayed. | pass |
| 3 | Incorrect password login | Execute the login script with inputs:  username: test Password: wrongpassword | Error message "Invalid username or password." is displayed. | Error message "Invalid username or password." is displayed. | pass |
| 4 | Check password encryption during login | Check if the password is encrypted during the login process. | Password comparison is done using encrypted password. | Password comparison is done using encrypted password. | pass |
| 5 | Verify session creation | Execute the login script and check if a session is created. | User session is created successfully with jwt token. | User session is created successfully with jwt token. | pass |

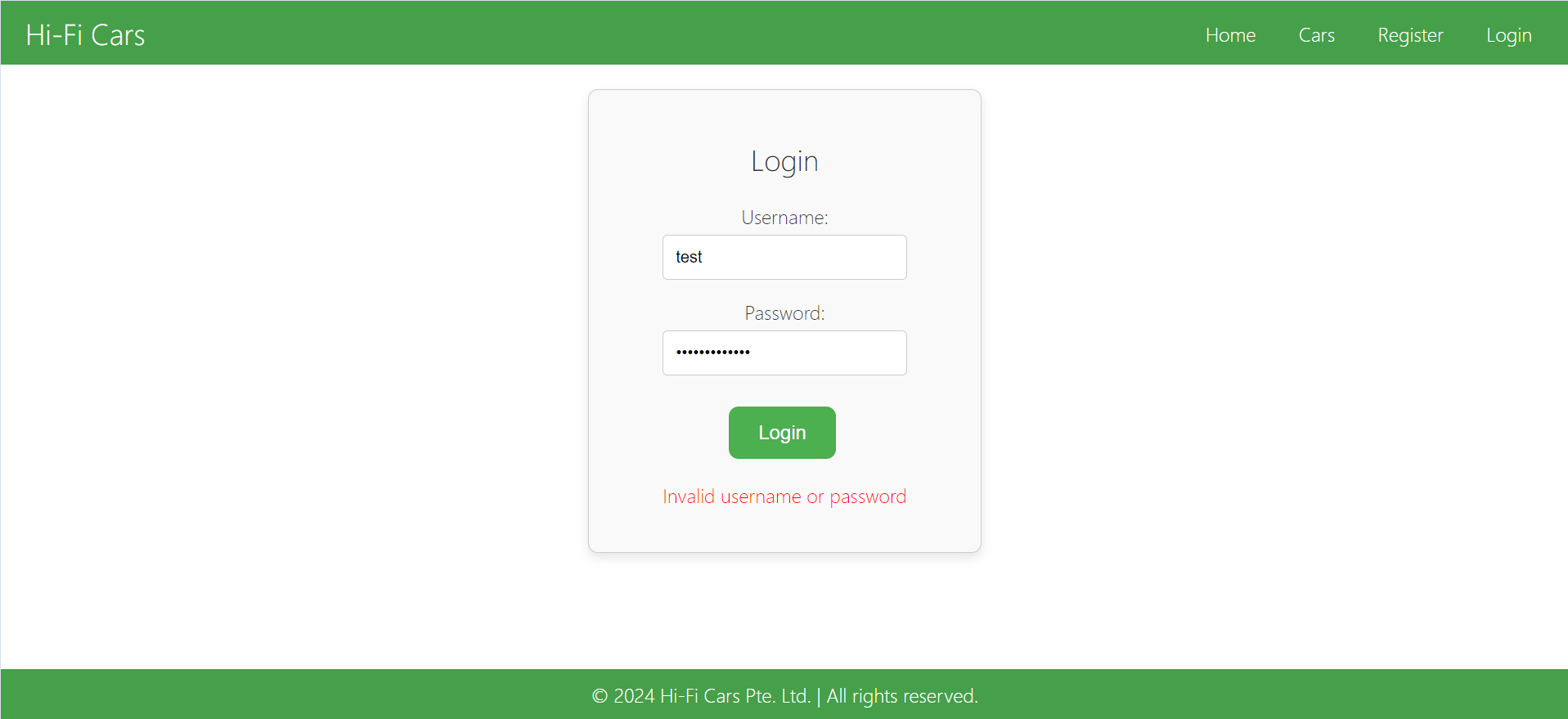
Valid user login





Invalid username login



Incorrect password login  


Check password encryption during login

2024-11-24T21:36:25.481+08:00 DEBUG 1712 --- [dse20503-project] [nio-8080-exec-7] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to com.example.dse20503\_project.controller.UserController#loginUser(LoginRequest)

2024-11-24T21:36:25.514+08:00 DEBUG 1712 --- [dse20503-project] [nio-8080-exec-7] m.m.a.RequestResponseBodyMethodProcessor : Read "application/json;charset=UTF-8" to [com.example.dse20503\_project.util.LoginRequest@216bb398]

Hibernate: select u1\_0.id,u1\_0.address,u1\_0.country,u1\_0.created\_at,u1\_0.email,u1\_0.password,u1\_0.phone\_number,u1\_0.role,u1\_0.state,u1\_0.updated\_at,u1\_0.username from users u1\_0 where u1\_0.username=?

Hibernate: select u1\_0.id,u1\_0.address,u1\_0.country,u1\_0.created\_at,u1\_0.email,u1\_0.password,u1\_0.phone\_number,u1\_0.role,u1\_0.state,u1\_0.updated\_at,u1\_0.username from users u1\_0 where u1\_0.username=?

Hibernate: select u1\_0.id,u1\_0.address,u1\_0.country,u1\_0.created\_at,u1\_0.email,u1\_0.password,u1\_0.phone\_number,u1\_0.role,u1\_0.state,u1\_0.updated\_at,u1\_0.username from users u1\_0 where u1\_0.username=?

2024-11-24T21:36:25.939+08:00 DEBUG 1712 --- [dse20503-project] [nio-8080-exec-7] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Using 'application/json', given [application/json, text/plain, \*/\*] and supported [application/json, application/\*+json]

2024-11-24T21:36:25.948+08:00 DEBUG 1712 --- [dse20503-project] [nio-8080-exec-7] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Writing [{token=eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIjoiUk9MRV9VU0VSIiwidXNlcklkIjoxMSwic3ViIjoidGVzdCIsImlhdCI6MTcz (truncated)...]

    // Endpoint for logging in a user

    @PostMapping("/login")

    public ResponseEntity<Map<String, String>> loginUser(@RequestBody LoginRequest loginRequest) {

        String username = loginRequest.getUsername();

        String password = loginRequest.getPassword();

        try {

            // Authenticate the user

            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

            // Load user details and generate JWT token

            final UserDetails userDetails = userService.loadUserByUsername(username);

            final User user = userService.getUserByUsername(username);

            final String jwt = jwtUtil.generateToken(userDetails, user.getId());

            // Create a response body with the token

            Map<String, String> responseBody = new HashMap<>();

            responseBody.put("token", jwt);

            // Return the JWT token in a JSON object

            return ResponseEntity.ok(responseBody); // Return 200 OK status code

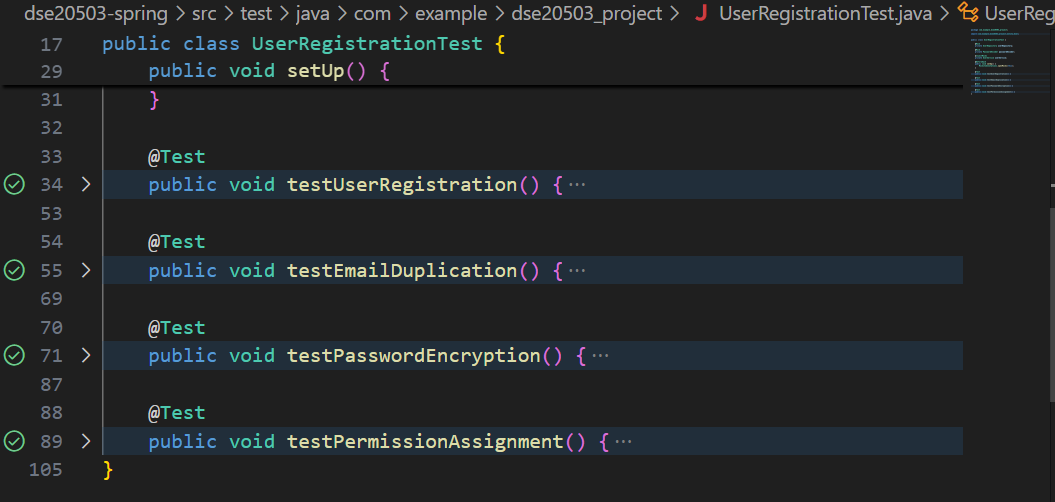
        } catch (AuthenticationException e) {

            return new ResponseEntity<>(Collections.singletonMap("error", "Invalid username or password"),

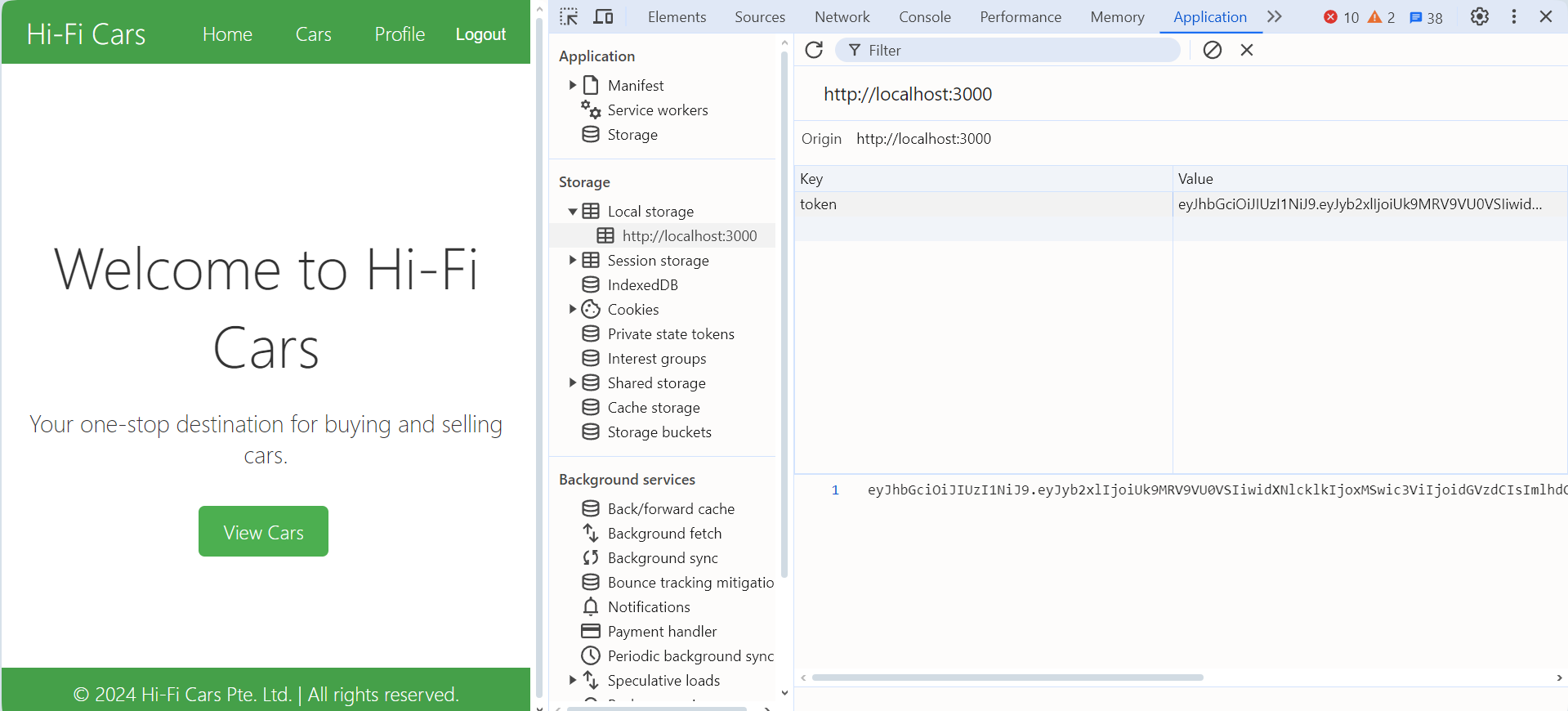
                    HttpStatus.UNAUTHORIZED); // Return 401 Unauthorized

        }

    }



Verify session creation



Test Scenario: S001 – Login Flow

Test Objective: Test that the login process on the used car sales portal works as expected.

Test Data: unit test

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | L001 | Test Scenario Name | Login Flow | |
| Test Case ID | | TC002 | Risk | High | |
| Pre-Requisites | | User Service and Security Configuration Setup | Test Priority | High | |
| Test Case Description | | Verify user login flow | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Valid user login | Execute the login script with inputs: email: test@example.com Password: password123 | User is successfully logged in and redirected to the home page. | User is successfully logged in and redirected to the home page. | - |
| 2 | Invalid email login | Execute the login script with inputs: email: wrong@example.com Password: password123 | Error message "Invalid username or password." is displayed. | Error message "Invalid username or password." is displayed. | - |
| 3 | Incorrect password login | Execute the login script with inputs: email: test@example.com Password: wrongpassword | Error message "Invalid username or password." is displayed. | Error message "Invalid username or password." is displayed. | - |
| 4 | Check password encryption during login | Execute the login script with inputs: email: test@example.com Password: password123 Monitor the login process using tools like Wireshark or application logs to ensure that the password is encrypted when sent over the network and during the process | Password is encrypted during the process and not stored/processed in plain text. | Password is encrypted during the process and not stored/processed in plain text. | - |
| 5 | Verify session creation | Execute the login script with inputs: email: test@example.com Password: password123 After login, check the browser's developer tools (usually under the 'Application' or 'Storage' tab) to see if a session cookie is created and is valid. | User session is created successfully. | User session is created successfully. | - |

package com.example.dse20503\_project;

import com.example.dse20503\_project.controller.UserController;

import com.example.dse20503\_project.entity.User;

import com.example.dse20503\_project.repository.UserRepository;

import com.example.dse20503\_project.service.UserService;

import com.example.dse20503\_project.util.JwtUtil;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.MockitoAnnotations;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import static org.mockito.ArgumentMatchers.any;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class)

public class UserLoginTest {

    @Autowired

    private MockMvc mockMvc;

    @MockBean

    private UserService userService;

    @MockBean

    private UserRepository userRepository;

    @MockBean

    private PasswordEncoder passwordEncoder;

    @MockBean

    private AuthenticationManager authenticationManager;

    @MockBean

    private JwtUtil jwtUtil;

    @BeforeEach

    public void setUp() {

        MockitoAnnotations.openMocks(this);

    }

    @Test

    public void testValidUserLogin() throws Exception {

        User user = new User();

        user.setId(1L);

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("encryptedPassword");

        when(userRepository.findByUsername("testuser")).thenReturn(user);

        when(passwordEncoder.matches("password123", user.getPassword())).thenReturn(true);

        when(jwtUtil.generateToken(any(UserDetails.class), any(Long.class))).thenReturn("dummyToken");

        mockMvc.perform(MockMvcRequestBuilders.post("/users/login")

                .contentType(MediaType.APPLICATION\_JSON)

                .content("{\"username\":\"testuser\", \"password\":\"password123\"}"))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$.token").value("dummyToken"));

    }

    @Test

    public void testInvalidEmailLogin() throws Exception {

        when(userRepository.findByUsername("wronguser")).thenReturn(null);

        mockMvc.perform(MockMvcRequestBuilders.post("/users/login")

                .contentType(MediaType.APPLICATION\_JSON)

                .content("{\"username\":\"wronguser\", \"password\":\"password123\"}"))

                .andExpect(status().isUnauthorized())

                .andExpect(jsonPath("$.error").value("Invalid username or password"));

    }

    @Test

    public void testIncorrectPasswordLogin() throws Exception {

        User user = new User();

        user.setId(1L);

        user.setUsername("testuser");

        user.setEmail("test@example.com");

        user.setPassword("encryptedPassword");

        when(userRepository.findByUsername("testuser")).thenReturn(user);

        when(passwordEncoder.matches("wrongpassword", user.getPassword())).thenReturn(false);

        mockMvc.perform(MockMvcRequestBuilders.post("/users/login")

                .contentType(MediaType.APPLICATION\_JSON)

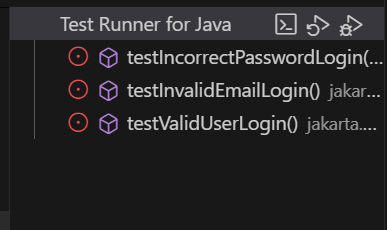
                .content("{\"username\":\"testuser\", \"password\":\"wrongpassword\"}"))

                .andExpect(status().isUnauthorized())

                .andExpect(jsonPath("$.error").value("Invalid username or password"));

    }

}



I was not able to make the test class work as I don’t have any more time to debug an error in user being null.

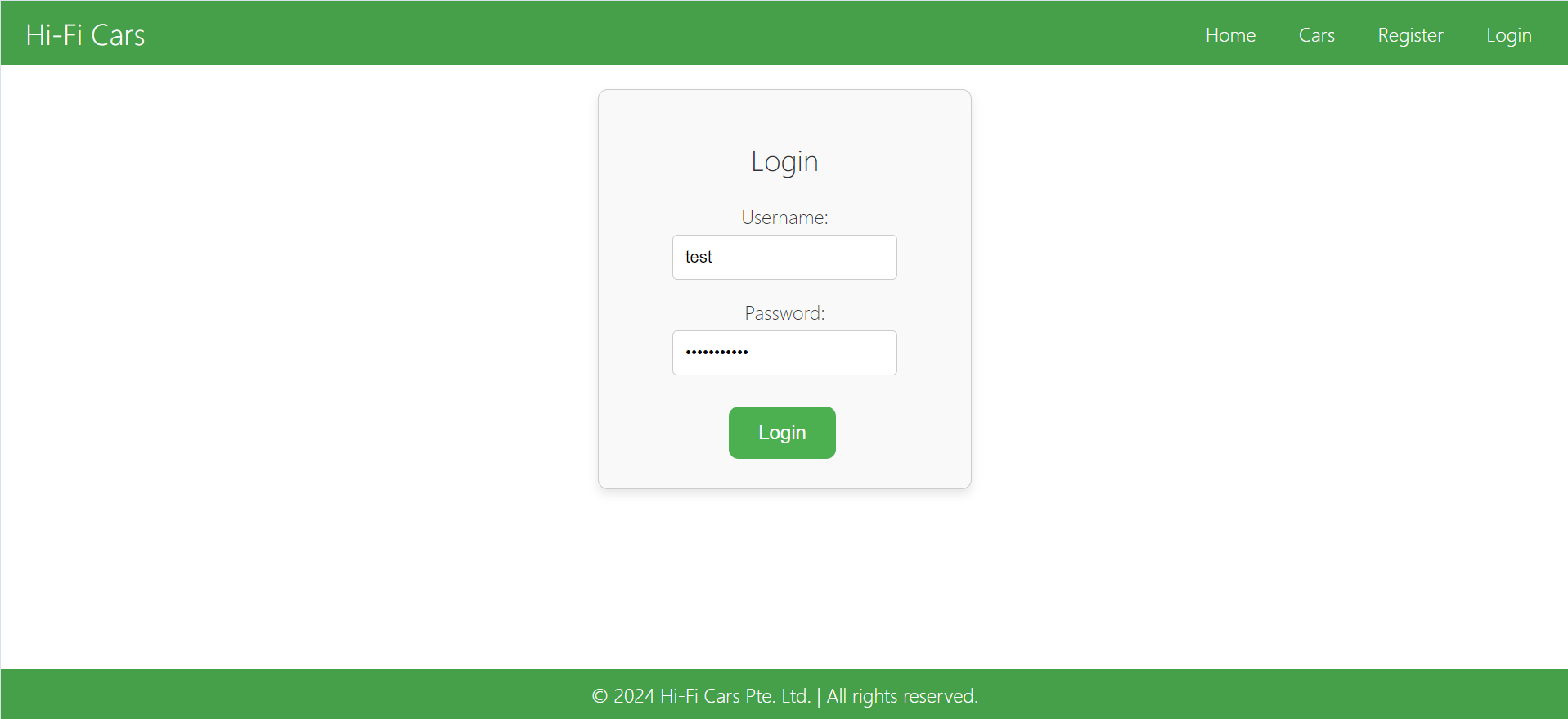
Test Scenario: AD001– Add Car Flow

Test Objective: Test that the add car process on the used car sales portal works as expected.

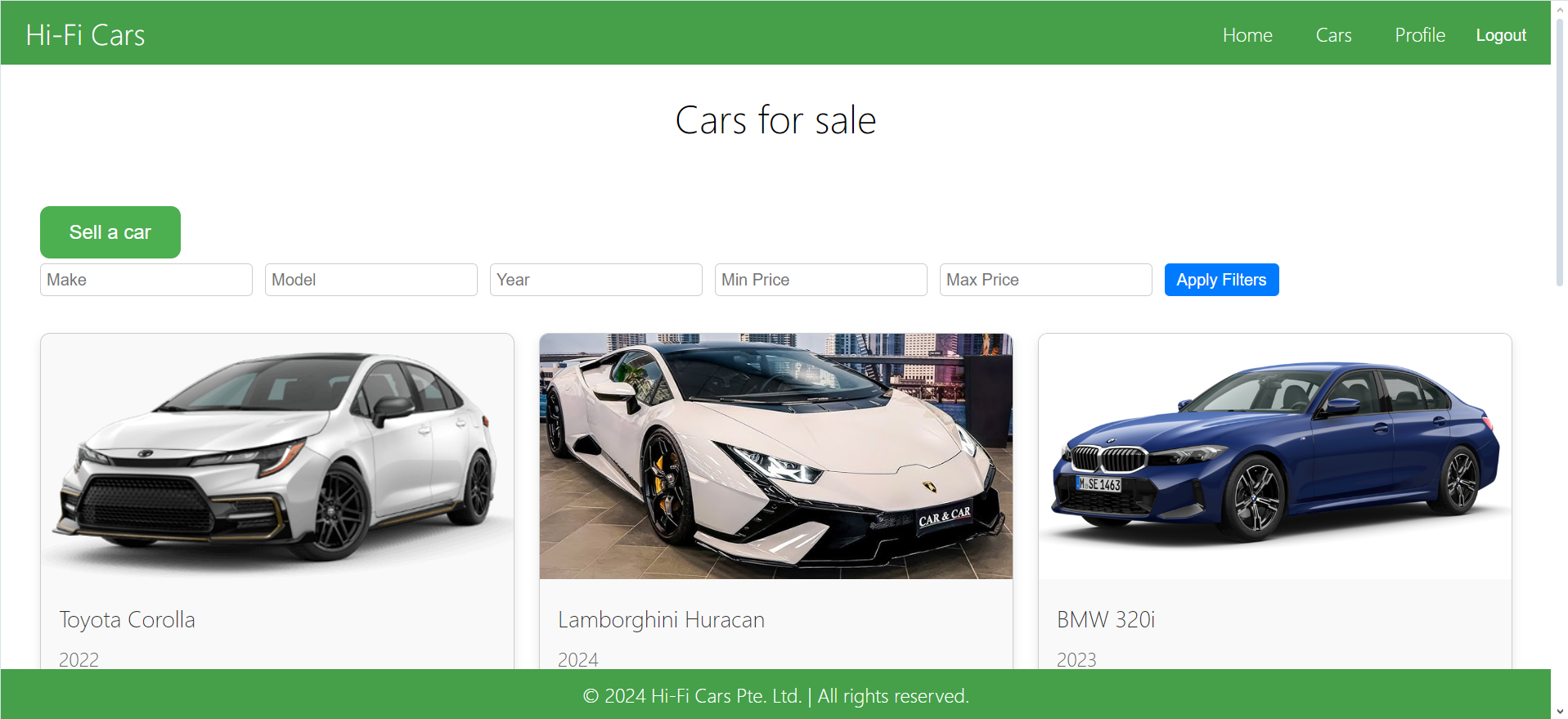
Test Data: car plate number, car brand, car model, car registration, car image, car price

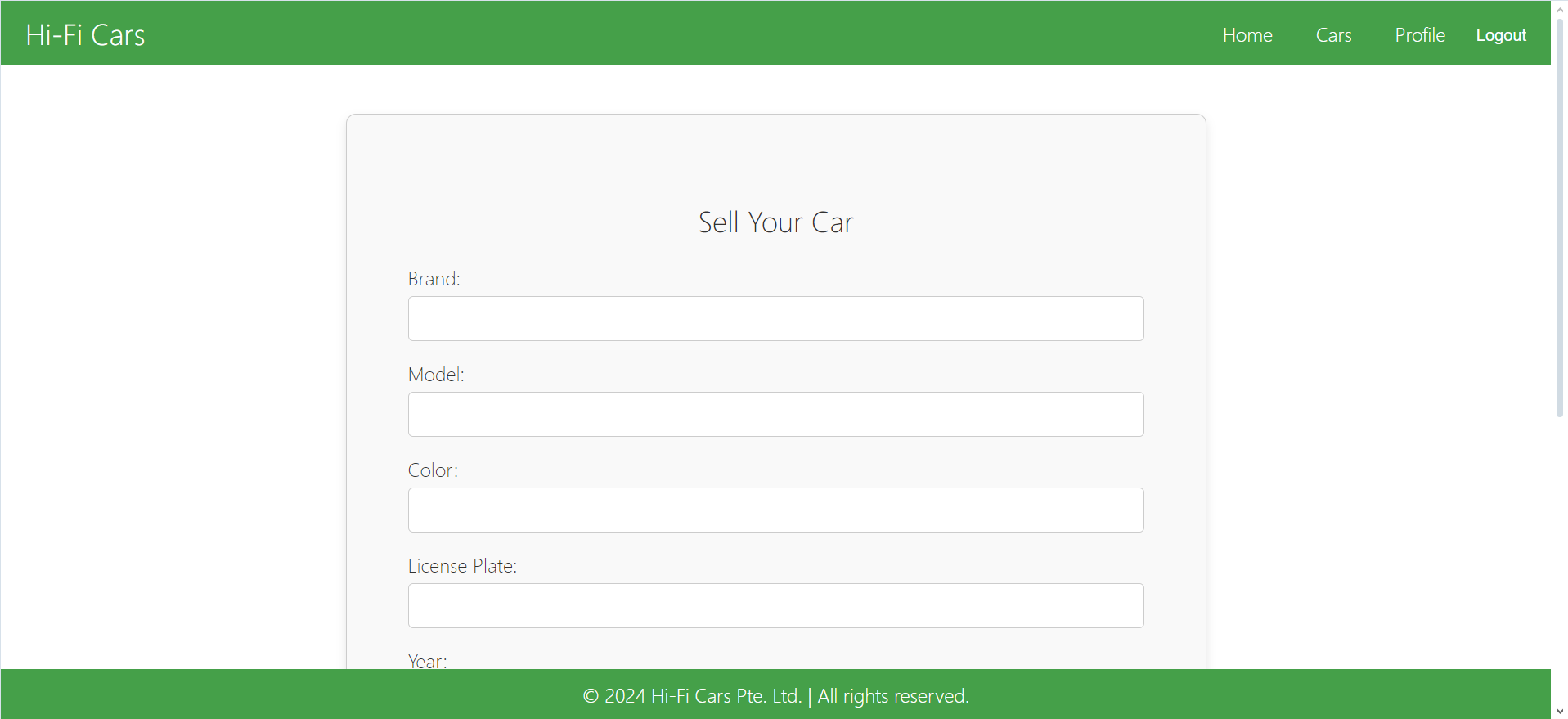
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | AD001 | Test Scenario Name | Add Car Flow | |
| Test Case ID | | TC\_ADD\_CAR\_001 | Risk | High | |
| Pre-Requisites | | User Service and Security Configuration Setup | Test Priority | High | |
| Test Case Description | | Verify user can successfully add a car | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Open add car page | Open the add car page URL | The page loads successfully and displays the add car form | The page loads successfully and displays the add car form | pass |
| 2 | Enter car plate number | Car Plate Number: ABC1234 | The entered car plate number is displayed in the form | The entered car plate number is displayed in the form | pass |
| 3 | Enter car brand | Car Brand: Toyota | The entered car brand is displayed in the form | The entered car brand is displayed in the form | pass |
| 4 | Enter car model | Car Model: Camry | The entered car model is displayed in the form | The entered car model is displayed in the form | pass |
| 5 | Enter car registration | Car Registration: 2011 | The entered car registration is displayed in the form | The entered car registration is displayed in the form | pass |
| 6 | Upload car image | Upload a valid image file | The image is uploaded successfully | The image is uploaded successfully | pass |
| 7 | Enter car price | Car Price: 20000.00 | The entered car price is displayed in the form | The entered car price is displayed in the form | pass |
| 8 | Click "Submit" button | Click "Submit" | The form is successfully submitted, and a success message is displayed | The form is successfully submitted, and a success message is displayed | pass |
| 9 | Verify if the car is added | Verify if the car appears in the car list | The new car information is displayed on the car list page | The new car information is displayed on the car list page | pass |
| 10 | Check car image display | Verify if the uploaded car image is displayed correctly | The uploaded car image is displayed correctly | The uploaded car image is displayed correctly | pass |
| 11 | Verify error message for no image | Try to submit the form without uploading an image | An error message "Car image is required" is displayed | An error message "Car image is required" is displayed | pass |
| 12 | Verify form validation | Try to submit the form with invalid or missing data | Relevant error messages are displayed for each invalid or missing input | Relevant error messages are displayed for each invalid or missing input | pass |
| 13 | Check database entry | Check if the car details are correctly saved in the database | The car details are correctly saved in the database | The car details are correctly saved in the database | pass |
| 14 | Verify car activation status | Verify if the car is automatically set to "Activated" after being added | The car is automatically set to "Activated" after being added | The car is automatically set to "Activated" after being added | pass |

Log in

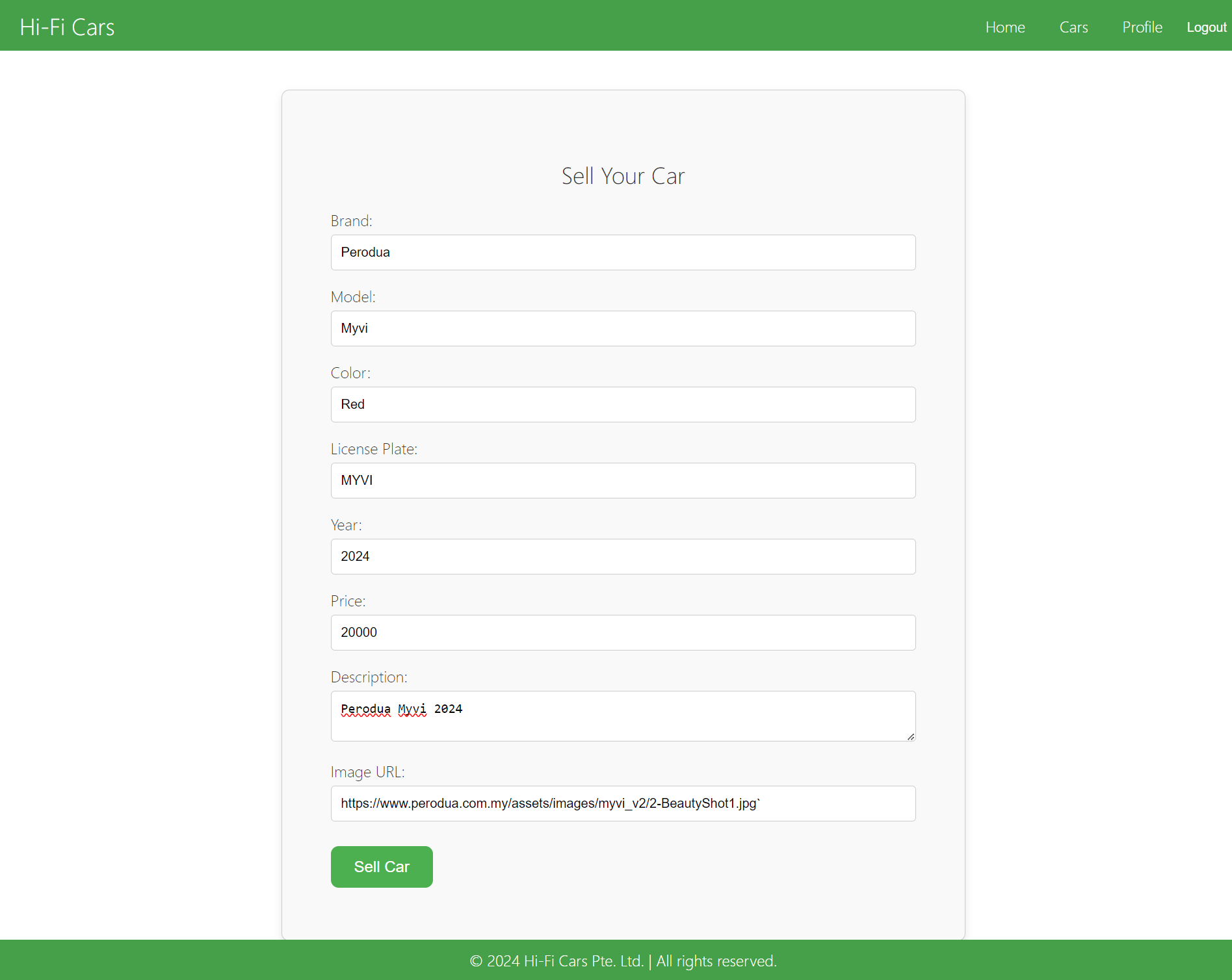


Open add car page





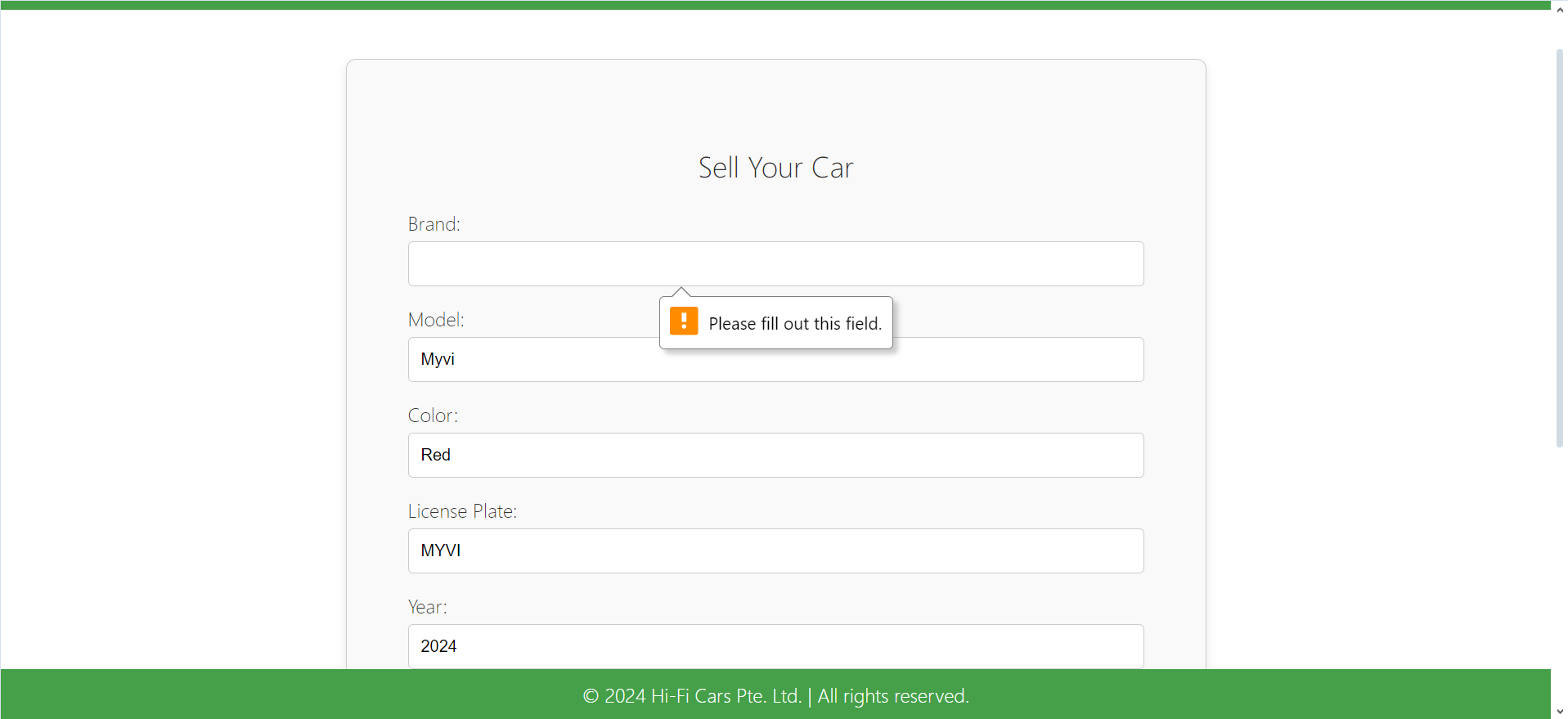
Enter data



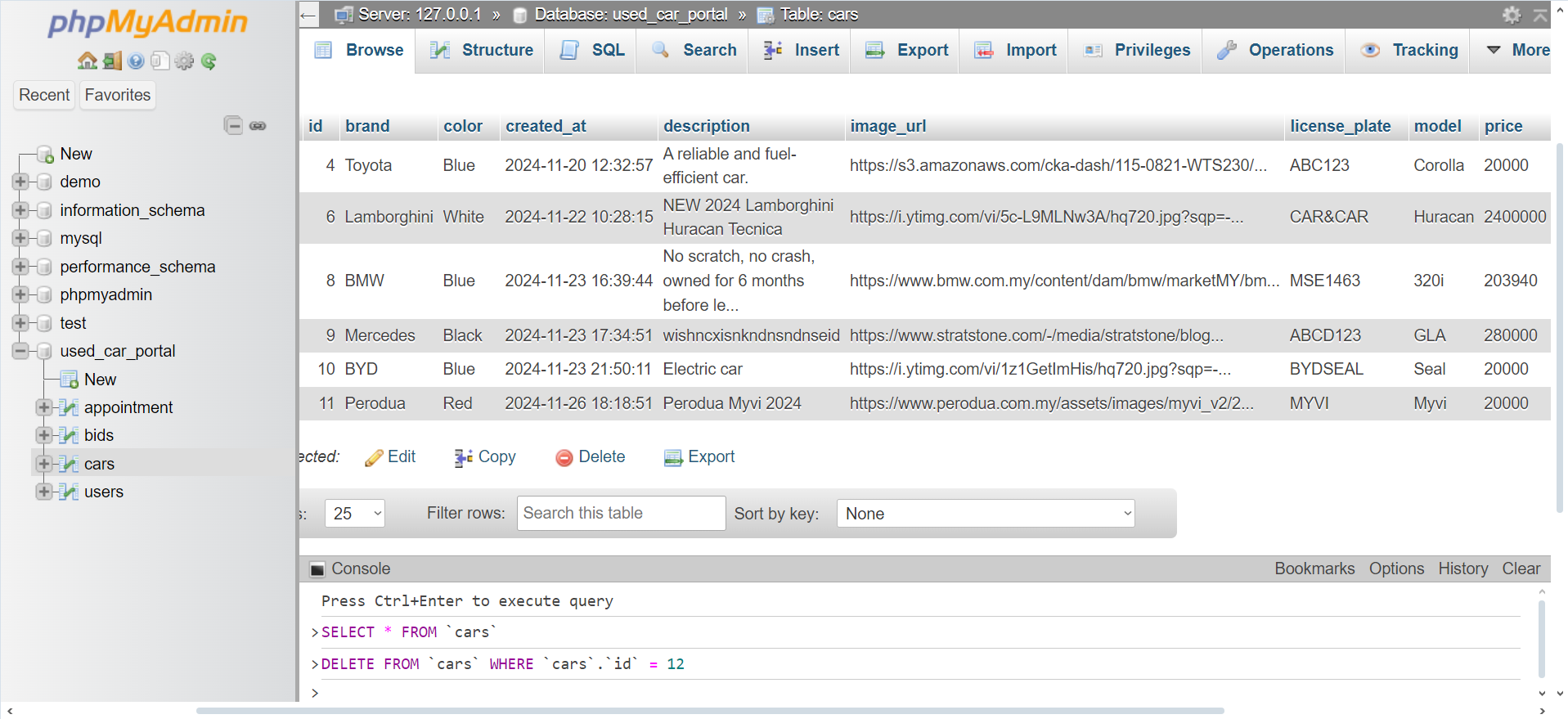
Verify error message for no image



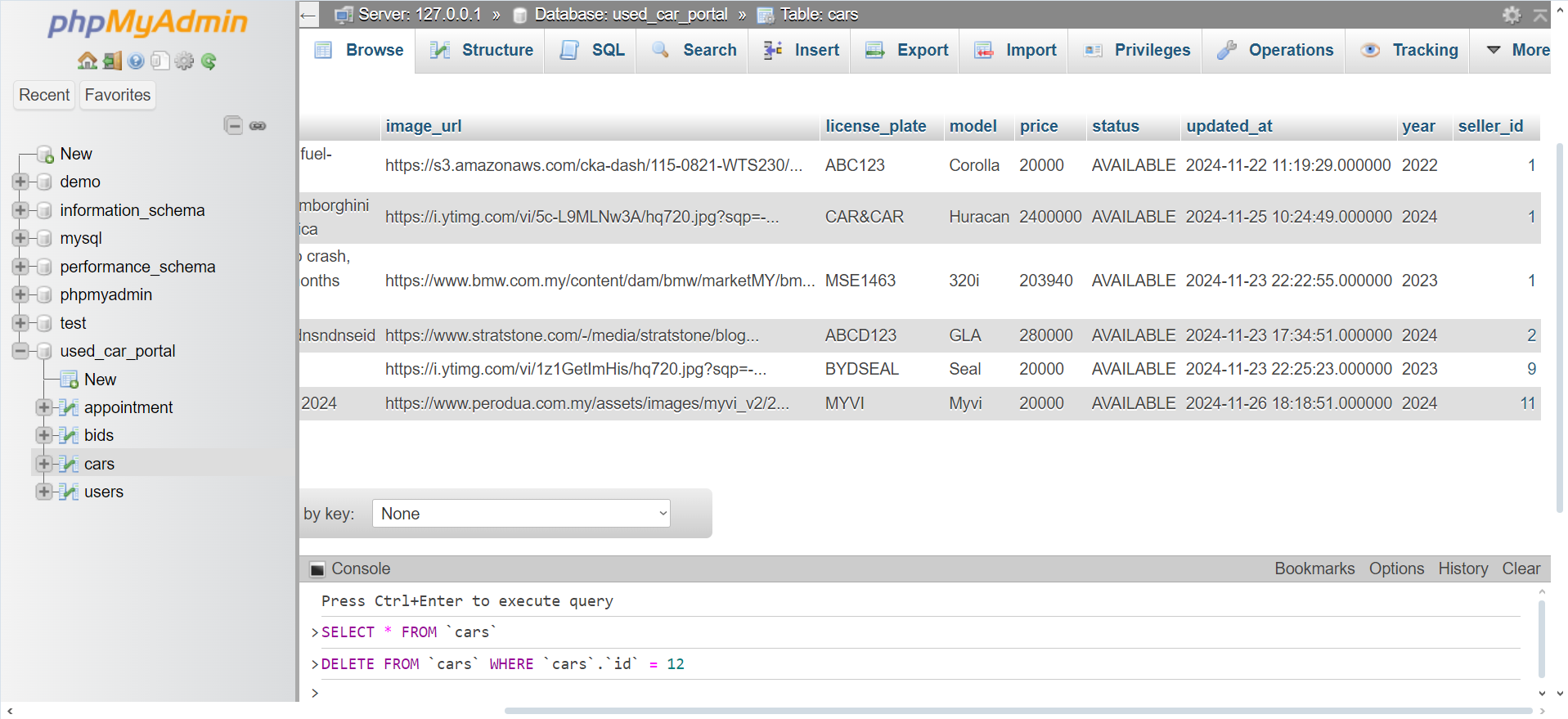
Verify form validation



Check database entry



Verify car activation status



Test Scenario: UCP001- User Change Profile

Test Objective: To validate that the user can successfully change their email and password on the user profile page, ensuring data integrity and proper error handling.

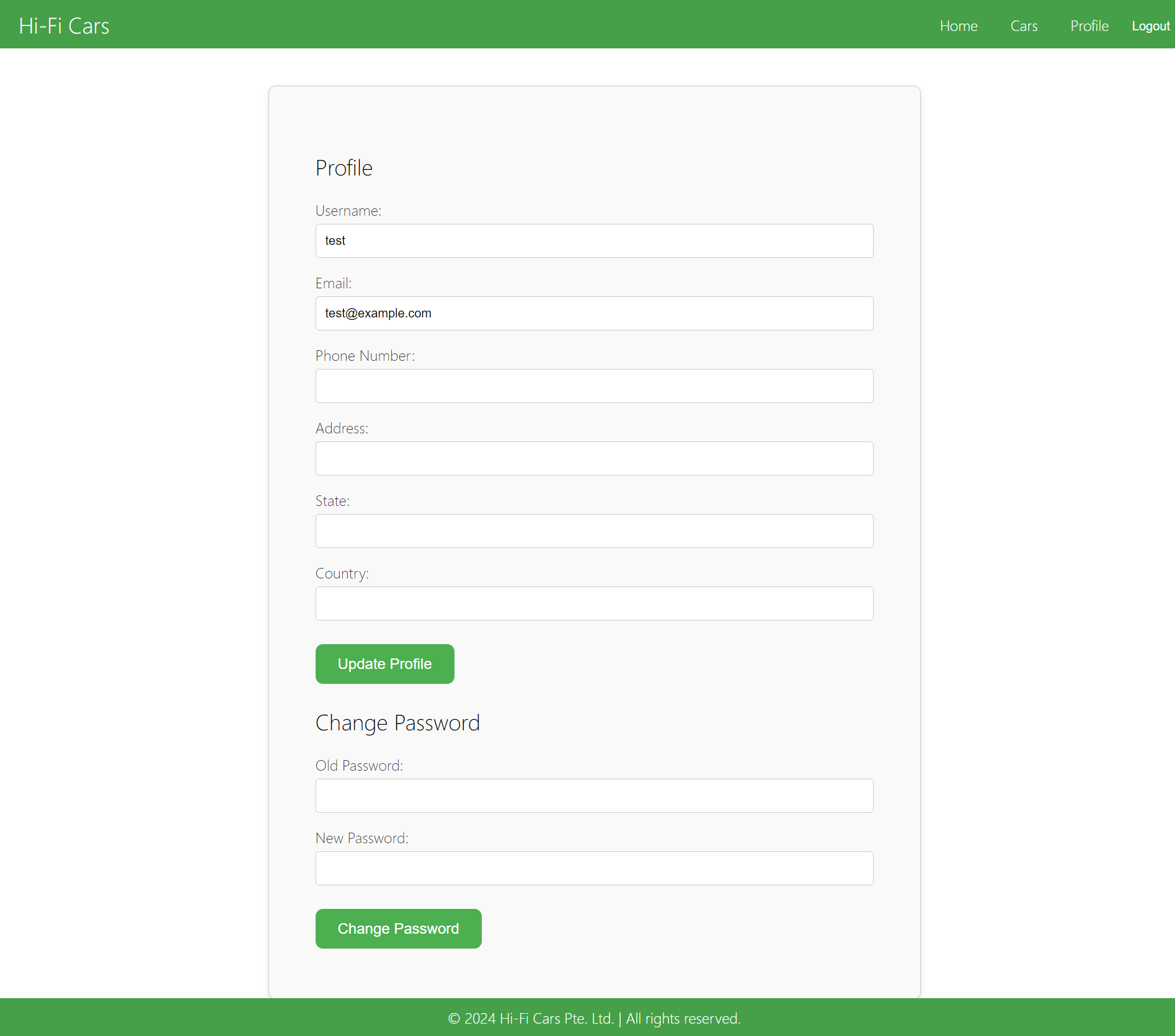
Test Data: User email,User password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | UCP001 | Test Scenario Name | User Change Profile | |
| Test Case ID | | TC\_CHANGE\_EMAIL\_001 | Risk | Medium | |
| Pre-Requisites | | User must be logged in and on the user profile page. | Test Priority | High | |
| Test Case Description | | Verify user can successfully change their email address. | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Open user profile page | Open the user profile page URL | The page loads successfully displaying user info and forms | The page loads successfully and displays the add car form | pass |
| 2 | Enter new email | New Email: new@example.com | The entered email is displayed in the input field | The entered email is displayed in the input field | pass |
| 3 | Click "Change Email" button | Click "Change Email" | The form is submitted, and a success message is displayed | The form is submitted, and a success message is displayed | pass |
| 4 | Verify email change in profile | Check user profile | The new email address is displayed in the user profile | The new email address is displayed in the user profile | pass |
| 5 | Verify email change in database | Check user record in database | The new email address is updated in the database | The new email address is updated in the database | pass |
| 6 | Verify error for invalid email | New Email: invalid-email | An error message "Invalid email format" is displayed | An error message "Invalid email format" is displayed | pass |
| 7 | Verify form validation | Leave email field empty | An error message "Email is required" is displayed | An error message "Email is required" is displayed | pass |

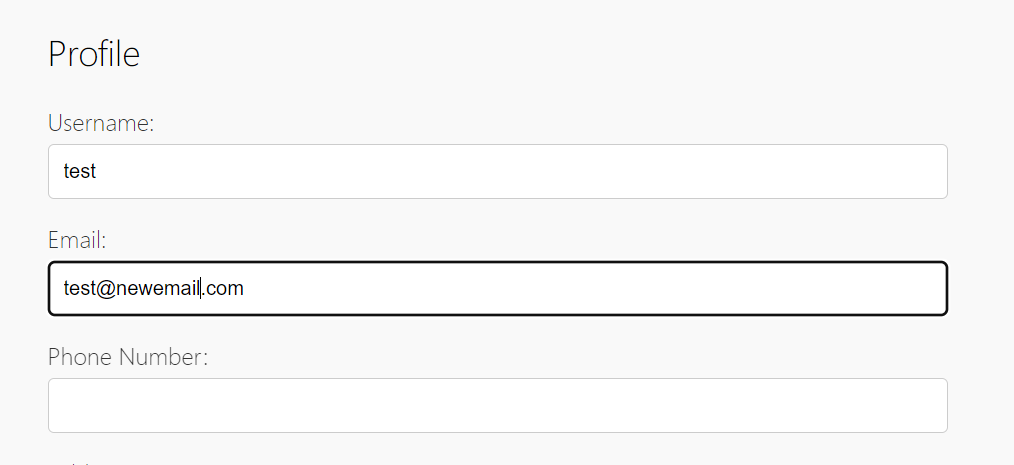
Test Case 2: Change Password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | UCP001 | Test Scenario Name | User Change Profile | |
| Test Case ID | | TC\_CHANGE\_PASSWORD\_001 | Risk | Medium | |
| Pre-Requisites | | User must be logged in and on the user profile page. | Test Priority | High | |
| Test Case Description | | Verify user can successfully change their password. | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Open user profile page | Open the user profile page URL | The page loads successfully displaying user info and forms | The page loads successfully and displays the add car form | pass |
| 2 | Enter new email | New Password: newPassword123 | The entered password is displayed in the input field | The entered password is displayed in the input field | pass |
| 3 | Click "Change Password" button | Click "Change Password" | The form is submitted, and a success message is displayed | The form is submitted, and a success message is displayed | pass |
| 4 | Verify password change in profile | Try to log in with the new password | User is able to log in with the new password | User is able to log in with the new password | pass |
| 5 | Verify password change in database | Check user record in database | The new email address is updated in the database | The new email address is updated in the database | pass |
| 6 | Verify error for weak password | New Password: 123 | An error message "Password is too weak" is displayed | The form is submitted, and a success message is displayed | fail |
| 7 | Verify form validation | leave password field empty | An error message "Password is required" is displayed | An error message "Email is required" is displayed | pass |

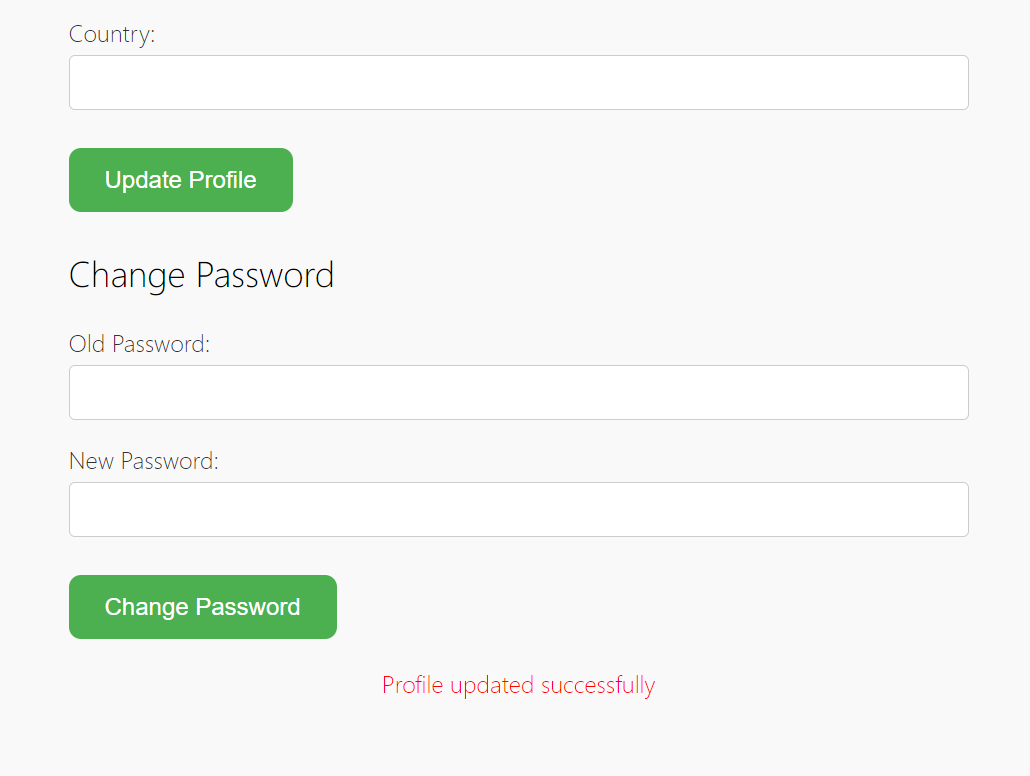
Open user profile page

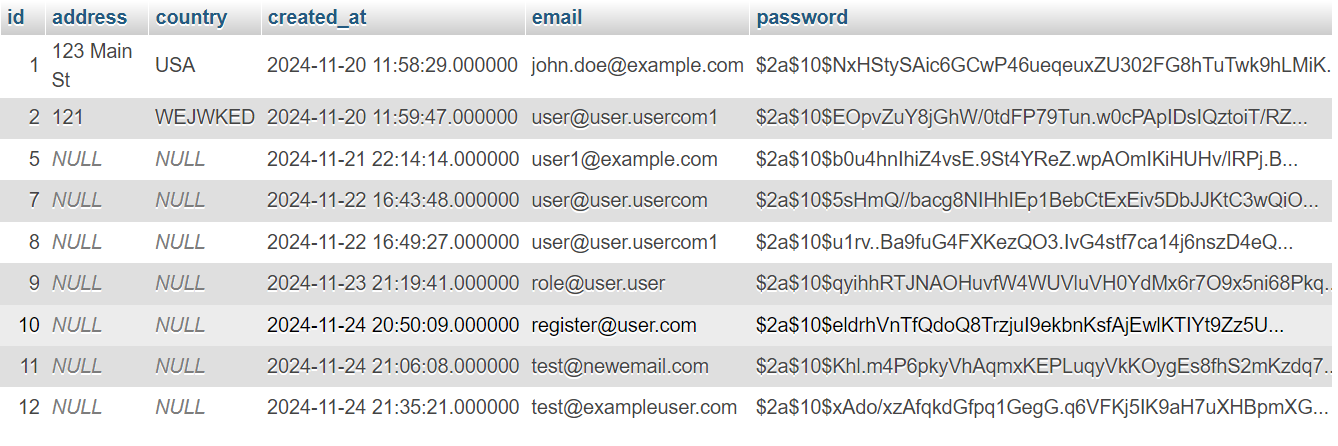


Enter new email



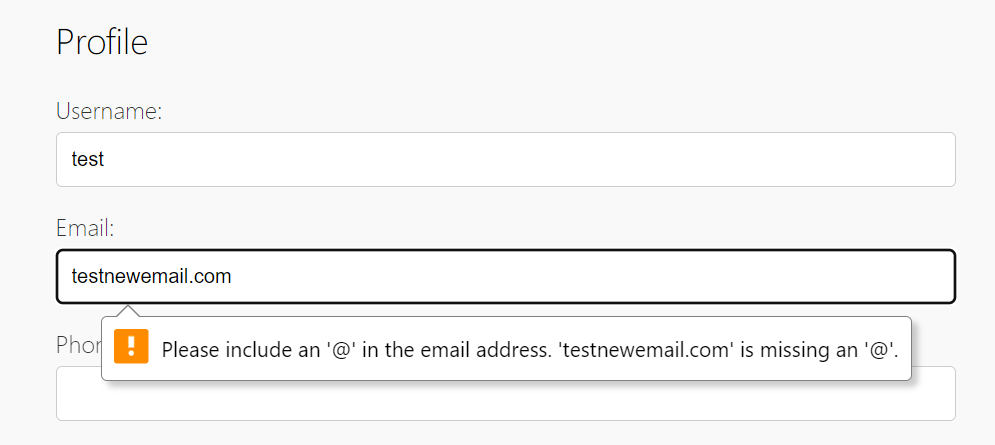
Verify email change in database



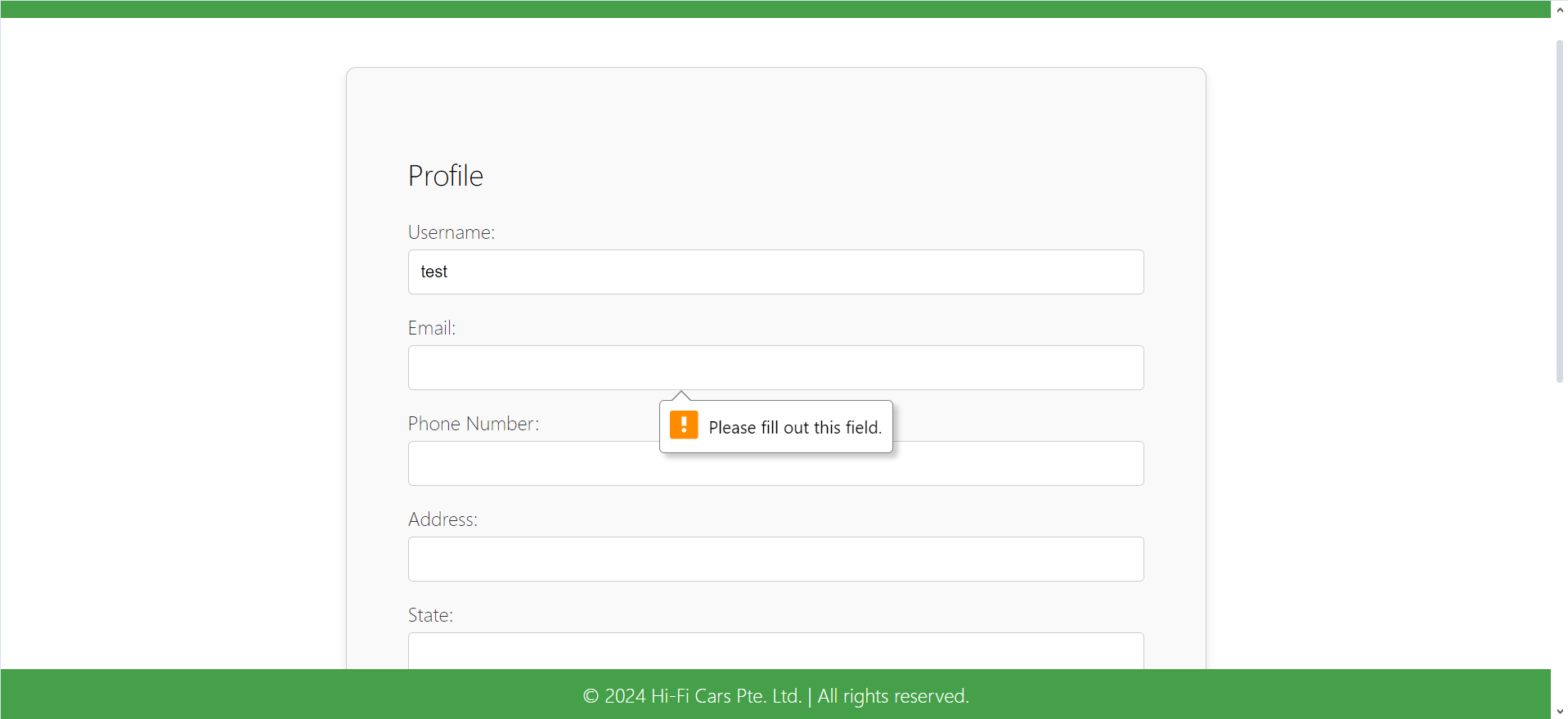




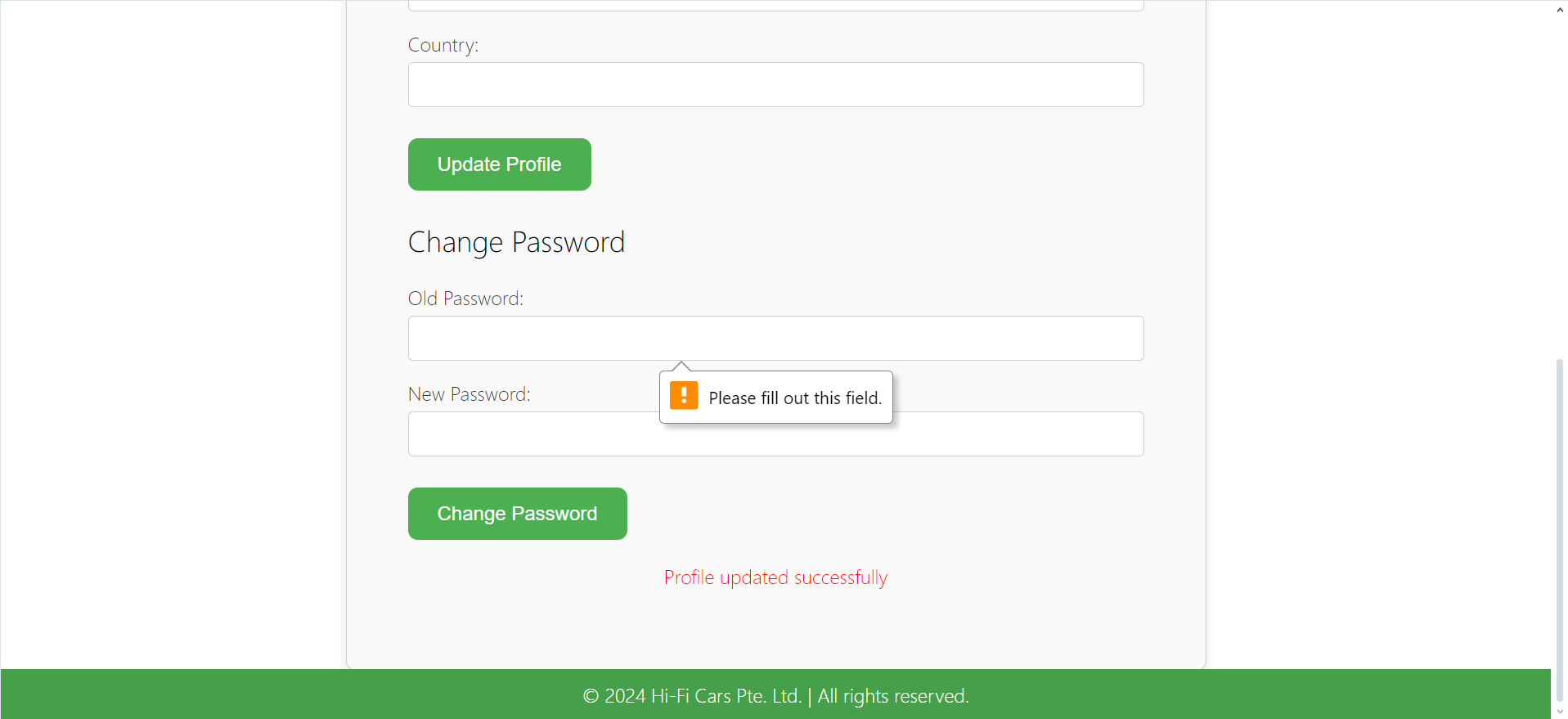
Verify error for invalid email



Verify form validation



Verify form validation

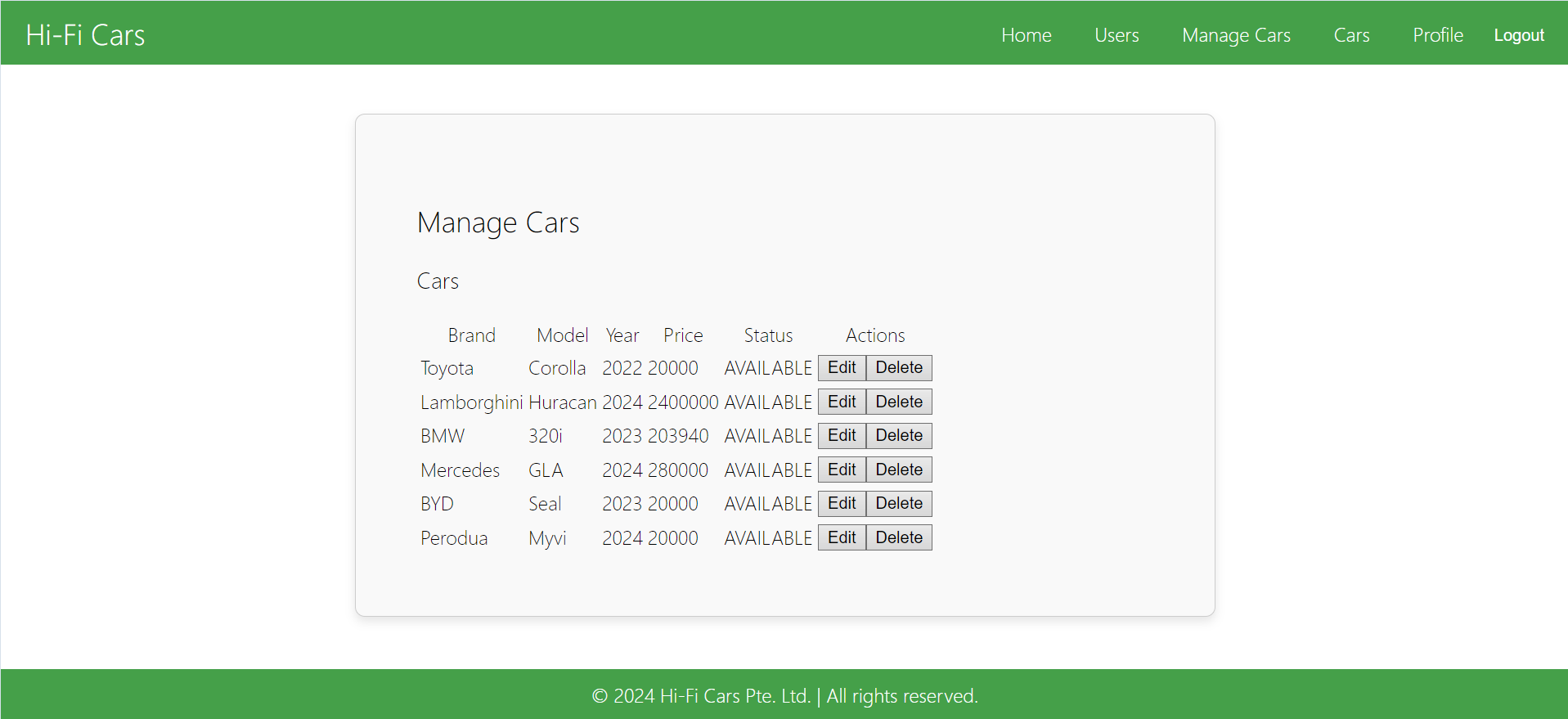


Test Scenario: TA001- Toggle Car Activation Status

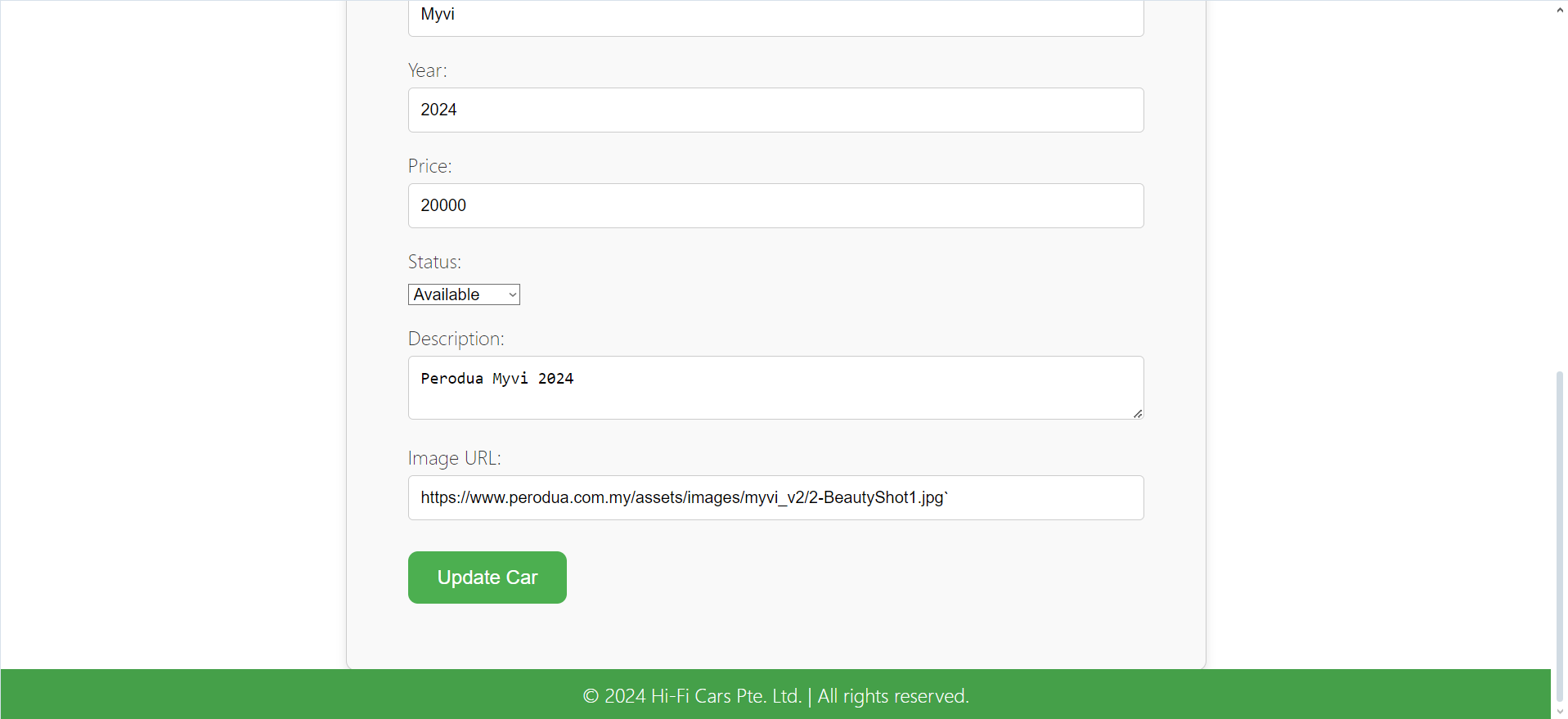
Test Objective: Verify that the "Toggle Activate" feature in the car management system correctly changes a car's activation status.

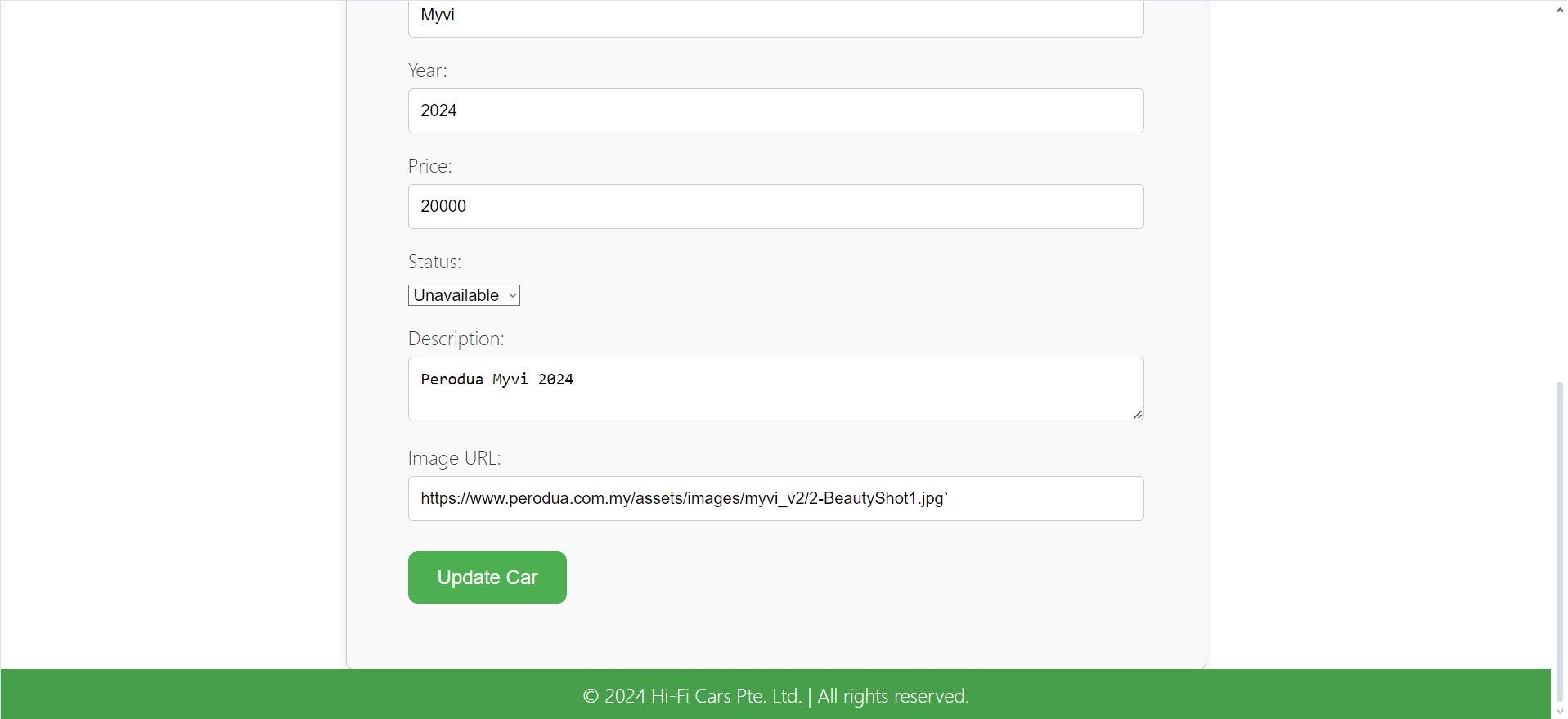
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | TA001 | Test Scenario Name | Toggle Car Activation Status | |
| Test Case ID | | TC\_TOGGLE\_ACTIVATE\_001 | Risk | Medium | |
| Pre-Requisites | | * User is logged in with appropriate permissions * Car entries are available in the system * Security configuration and user roles are set up correctly | Test Priority | High | |
| Test Case Description | | Verify that a user can successfully toggle the activation status of a car. | | | |
| Test Execution Steps: | | | | | |
| No. | Action | Inputs | Expected Output | Actual Output | Test Result |
| 1 | Open User Car List page | Open the User car list page URL | The page loads successfully and displays the car list | The page loads successfully and displays the car list | pass |
| 2 | Identify car to toggle activation status | Locate a car with a known status | The car is located in the list with the current status | he car is located in the list with the current status | pass |
| 3 | Click "Toggle Activate" button | Click "Toggle Activate" button | The car's status is toggled and updated | The car's status is toggled and updated | pass |
| 4 | Verify status change | Refresh the car list page | The car's status reflects the toggled state | The car's status reflects the toggled state | pass |
| 5 | Check database entry | Query the database for car status | The car status in the database is updated accordingly | The car status in the database is updated accordingly | pass |
| 6 | Ensure UI reflects changes immediately | Toggle activation status and observe UI update | The UI updates immediately to reflect the new status | The UI updates immediately to reflect the new status | Pass |

Open User Car List page

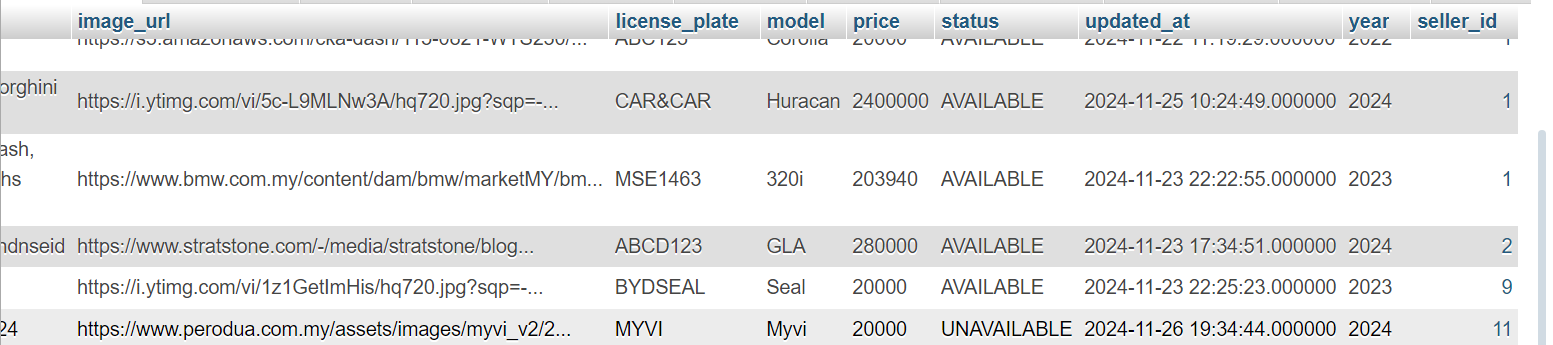


Verify status change





Check database entry



**Task 8: Evaluation of Risk-Based Testing Strategy Effectiveness and Assessment of Security Testing Cycles**

**Introduction:** Risk-Based Testing (RBT) is a strategic approach to testing that prioritizes testing efforts based on the identified risks to the system. This approach ensures that testing resources are allocated efficiently, focusing on areas of the system that pose the highest potential impact or likelihood of failure. This report evaluates the effectiveness of our RBT strategy and assesses the security testing cycles employed in our project.

**Effectiveness of Risk-Based Testing Strategy:**

1. **Risk Coverage:** The RBT strategy effectively prioritizes testing based on identified risks, ensuring that critical functionalities and high-risk areas receive thorough testing. By leveraging risk assessment matrices, we identified and focused on testing scenarios that could significantly impact user experience, system stability, and data integrity.
2. **Priority Alignment:** Test cases were prioritized according to their potential impact and likelihood, ensuring that critical paths and essential functionalities were thoroughly tested. High-priority tests such as registration, login, and core functionality validation were emphasized to mitigate risks early in the development lifecycle.
3. **Testing Outcomes:** Post-test analysis indicated that the RBT strategy successfully identified critical defects and vulnerabilities early in the testing phase. Issues related to user registration failures, login inconsistencies, and functionality errors were detected and addressed promptly, contributing to improved system reliability.
4. **Feedback Integration:** Continuous feedback loops from testing informed development iterations, allowing for timely adjustments and improvements. This iterative process facilitated the resolution of identified issues and enhanced overall system quality.

**Assessment of Security Testing Cycles:**

1. **Scope and Coverage:** Security testing cycles encompassed comprehensive evaluations of authentication mechanisms, access controls, data protection measures, and vulnerability assessments. This ensured that the system's security posture was rigorously tested against potential threats and vulnerabilities.
2. **Testing Techniques:** Employing techniques such as penetration testing, vulnerability scanning, and security code reviews provided robust coverage across different layers of the application stack. These techniques identified and remediated security weaknesses before deployment, minimizing security risks.
3. **Effectiveness in Threat Mitigation:** Security testing cycles effectively mitigated identified threats and vulnerabilities, enhancing the system's resilience against potential cyber-attacks and unauthorized access attempts. Critical security patches and configurations were implemented to strengthen the system's defenses.
4. **Continuous Improvement:** Ongoing evaluations and updates to security testing methodologies and tools ensured that our approach remained adaptive to emerging threats and industry best practices. Regular updates to security protocols and incident response plans were integral to maintaining a proactive security posture.

**Conclusion:** The adoption of a Risk-Based Testing strategy has proven effective in prioritizing testing efforts and mitigating risks associated with system functionalities and security vulnerabilities. By aligning testing priorities with identified risks and employing comprehensive security testing cycles, we have enhanced the overall quality, reliability, and security of our system. Moving forward, continuous refinement of our testing strategies and proactive security measures will be essential to sustaining a robust and resilient application environment.

**Recommendations:** Based on the evaluation, we recommend further enhancing automated testing capabilities to streamline regression testing and increase test coverage across system updates and feature enhancements. Additionally, investing in continuous training and certification programs for testing and security teams will ensure ongoing proficiency in identifying and mitigating evolving threats.

This report underscores our commitment to delivering a secure and reliable system that meets user expectations and industry standards for quality and security.

**Task 9 :Evaluation of Testing Strategy and Comparison of Its Pros and Cons**

**Introduction:** Effective testing strategies are crucial for ensuring the quality and reliability of software systems. This report evaluates our current testing strategy, highlighting its strengths, weaknesses, and recommendations for improvement.

**Testing Strategy Evaluation:**

**1. Strengths:**

* **Comprehensive Test Coverage:** Our testing strategy encompasses a wide range of functional, including unit testing, integration testing, regression testing, and security testing. This ensures thorough validation of both individual components and system-wide interactions.
* **Risk-Based Approach:** Utilizing risk assessment matrices, we prioritize testing efforts based on identified risks to the system. This approach maximizes testing efficiency by focusing on high-impact and high-probability scenarios, thereby mitigating potential risks early in the development lifecycle.
* **Feedback Integration:** Continuous feedback loops from testing inform iterative development cycles, allowing for timely bug fixes and improvements. This iterative process enhances the agility and responsiveness of our development efforts, ensuring that issues are addressed promptly.
* **Automation Capabilities:** Automation is leveraged extensively for repetitive and regression testing tasks, improving test coverage and efficiency while reducing manual effort. Automated tests also contribute to faster feedback cycles and earlier detection of defects in our future work.

**2. Weaknesses:**

* **Limited Non-Functional Testing:** While our strategy emphasizes functional testing, there is a need to strengthen non-functional testing areas such as usability testing, accessibility testing, localization testing and performance testing. Enhancing these aspects would better address user experience and system performance concerns.
* **Resource Intensiveness:** Comprehensive testing, especially for complex systems, can be resource-intensive in terms of time, manpower, and infrastructure. Balancing thorough testing with project timelines and resource constraints remains a challenge.
* **Dependency on Expertise:** Effective implementation of our testing strategy relies heavily on the expertise and experience of our testing team. Ensuring consistent skill development and knowledge transfer is essential for maintaining testing quality and effectiveness.

**3. Recommendations for Improvement:**

* **Enhance Non-Functional Testing:** Allocate additional resources and focus on enhancing non-functional testing processes to address usability, accessibility, performance, and security aspects more comprehensively.
* **Optimize Test Automation:** Continuously expand test automation coverage to include more scenarios and integrate automated tests earlier in the development process. This will improve test repeatability, reduce manual effort, and accelerate time-to-market.
* **Implement Continuous Testing:** Adopt continuous testing practices to integrate testing seamlessly into the CI/CD pipeline. This approach ensures that every code change is automatically validated through a series of tests, promoting early defect detection and faster feedback.
* **Training and Development:** Invest in ongoing training and skill development programs for the testing team to stay abreast of emerging technologies, testing methodologies, and industry best practices. This will enhance the team's capabilities and improve testing efficiency and effectiveness.

**Conclusion:** Our current testing strategy demonstrates strengths in comprehensive test coverage, risk-based prioritization, feedback integration, and automation capabilities. However, there are opportunities for improvement, particularly in enhancing non-functional testing, optimizing automation, and investing in continuous testing practices. By addressing these areas, we aim to further enhance the quality, reliability, and efficiency of our testing processes, ultimately delivering superior software products that meet user expectations and business requirements.

**Task 10: Evaluation of Heuristic Testing Methods: Importance, Environment Suitability, and Review**

**Introduction:** Heuristic testing methods play a crucial role in software testing by providing structured guidelines and strategies for identifying potential defects and validating system behaviour. This report evaluates the importance and suitability of heuristic testing methods in our testing approach, along with a review of identified heuristic techniques.

**Importance of Heuristic Testing Methods:**

1. **Structured Approach:** Heuristic testing methods offer structured guidelines and techniques for uncovering defects and verifying software functionalities. They provide a systematic approach that complements formal test cases, helping testers uncover issues that might be missed with traditional testing approaches.
2. **Comprehensive Coverage:** By leveraging heuristic techniques, testers can explore different aspects of the software, including user interfaces, error handling, boundary conditions, and performance characteristics. This broadens test coverage and ensures that critical areas of the system are thoroughly validated.
3. **Efficiency and Effectiveness:** Heuristic testing promotes efficient testing practices by focusing on high-risk areas and common sources of defects. This targeted approach maximizes testing effectiveness, enabling testers to prioritize their efforts based on potential impact and likelihood of issues.
4. **Adaptability to Agile Practices:** Heuristic methods are well-suited for Agile and iterative development environments where rapid feedback and continuous improvement are essential. They support quick adaptation to changing requirements and facilitate early defect detection and resolution.

**Environment Suitability Assessment:**

1. **Application Complexity:** Heuristic testing methods are particularly suitable for applications with complex workflows, user interactions, and integrations. They help in simulating real-world scenarios and uncovering usability issues that impact user experience.
2. **Resource Constraints:** Heuristic techniques can be adapted to accommodate resource constraints, such as limited time or availability of testing resources. They offer flexibility in test execution, allowing testers to prioritize critical tests and optimize testing efforts accordingly.
3. **Skill Requirements:** Effective implementation of heuristic testing methods requires testers with a strong understanding of the application domain, testing techniques, and heuristic guidelines. Continuous skill development and knowledge sharing are essential to maintain proficiency and ensure effective application of these methods.

**Review of Identified Heuristic Methods:**

1. **User Interface Heuristics:** Evaluates the usability and accessibility of user interfaces, focusing on clarity, consistency, and navigability.
2. **Error Guessing:** Relies on testers' intuition and experience to predict potential errors based on past issues or common pitfalls in similar applications.
3. **Exploratory Testing:** Combines heuristic methods with exploratory approaches to dynamically explore the software, uncovering defects through interactive and ad-hoc testing sessions.

**Conclusion:** Heuristic testing methods are indispensable tools in our testing strategy, offering a structured yet adaptable approach to uncovering defects and validating software functionalities. Their importance lies in providing comprehensive test coverage, promoting efficiency, and supporting agile development practices. When applied judiciously based on application complexity, resource constraints, and skill requirements, heuristic methods enhance our ability to deliver high-quality software that meets user expectations and business objectives.

**Recommendations:** Continued investment in training and development programs for testers to enhance proficiency in heuristic testing methods. Integration of heuristic techniques into automated testing frameworks to augment test coverage and efficiency. Regular review and refinement of heuristic guidelines based on feedback from testing outcomes and evolving application requirements.

Bibliography

ChatGPT, GitHub Copilot – Debugging and explaining code