Specifications Document For A Web-Based Car Rental Application

Submitted by : Michael Okwiri Edalia

Table of Contents

Table of Figures

Introduction

1.1. Purpose

The purpose of this document is to:

i) Specify the system requirements of the CarRentals web application using formal and informal methods.

ii) Illustrate design diagrams used to develop the aforementioned application

iii) List testing techniques used to analyse the final application

iv) Verify the specifications, design and development of the system using formal methods.

1.2. Scope

This report describes the design, development and testing of the CarRentals web application. It therefore describes the functional requirements of the system, outlines the design of the system, documents test cases and uses formal methods to mathematically prove its functionality.

1.3. System overview

The developed system is a web application used by CarRental's clients to hire vehicles at a fee. Customers interested in renting out a car would be required to register an account with this system. They would then sign in, select a vehicle, choose dates for picking up and returning a vehicle, and pay to complete their booking. While logged in, clients would be able to view previous bookings made with their account. An admin can sign in on their respective page to view the customer's booking.

1.4. System Specification

This section covers the functional requirements of CarRental's web application.

1.4.1. User registration

Description	Customers are required to register a user account within the system before they can begin the car rental process
Inputs	First name, surname, email address, password, password confirmation
Pre-condition	The email address must have a valid format, e.g. example@domain.com The password must have the right format, e.g. one uppercase, one lowercase, at least 8 characters. The password field must match the password confirmation field. All fields must have a value.
Result	The user's details are stored in the database and the user is redirected to the login page
Error Handling	If any field is empty, the system displays an error message highlighting there is an empty field. If any field has the wrong format, the system displays an error message showing the input is invalid.

1.4.2. *User authentication*

Description	Customers must verify their identity before starting the car rental process
Inputs	Email address, password
Pre-condition	The email address must have a valid format, e.g. example@domain.com All fields must have a value.
Result	The user's details are queried from the database, a session is started and the user is directed to the home page.
Error Handling	If any field is empty, the system displays an error message highlighting there is an empty field. If the email or password provided do not match any existing record, the system displays an error message stating the information provided is invalid.

1.4.3. *Selecting Pickup and return information*

Description	Customers must specify pickup and return dates for the hire
Inputs	Pickup date, Return date
Pre-condition	The earliest dates which can be used for picking up or returning a vehicle is the current date The return date must be later than the pickup date All fields must have a value.
Result	The system stores both the pickup and return dates and redirects the user to the booking page to view available cars for hire.
Error Handling	If any field is empty, the system displays an error message highlighting there is an empty field. If the return date is earlier than a pickup date, the system displays an error message highlighting it is an invalid option.

1.4.4. *Reserving a vehicle*

Description	Customers must be able to reserve an available vehicle.
Inputs	User clicks the element on the page of a car they are interested in
Pre-condition	The user must have an active sign in session There must be an existing pickup and return date to calculate their fee.
Result	The system accepts the user's selection and shows a confirmation page showing the car selected and total renting fee.
Error Handling	If the user is not signed in, the system informs them to login to proceed with the process. If the pickup and return dates are empty, the system gives the user a link to return to the home page.

1.4.5. *Searching for a vehicle*

Description	Customers must be able to search for an available vehicle.
Inputs	Car search string
Pre-condition	The user must have an active sign in session Their search must be a valid string There must be an existing pickup and return date to calculate their fee.
Result	The system accepts the user's string, queries the input against the database and presents a result.

Error Handling	If the user is not signed in, the system redirects the user to the login page. If the pickup and return dates are empty, the system redirects the user to the home page. If the user's search is not a valid string, the system informs them that their search request is invalid. If the user's search lacks a result, the system will inform them that their search was unsuccessful.

1.4.6.Accessing customer reservations

Description	Customers must be able to view reservations made with their account.
Inputs	User clicks the 'show reservations' link
Pre-condition	The user must have an active sign in session
Result	The system redirects the user to the bookings page and loads all bookings made using the customer's account.
Error Handling	If the user is not signed in, the system redirects the user to the login page.

1.4.7.Admin authentication

Description	Admins must verify their identity before accessing the admin page.
Inputs	Username, password
Pre-condition	All fields must have a value.
Result	The admin's user details are queried from the database, a session is started and they are redirected to the admin index page.
Error Handling	If any field is empty, the system displays an error message highlighting there is an empty field. If the username or password provided do not match any existing record, the system displays an error message stating the information provided is invalid.

1.4.8.Admin viewing customer reservations

Description	Admin users must be able to view all the customer's current reservations
Inputs	Admin clicks 'show reservations' button
Pre-condition	The admin must have an active sign in session.
Result	The system loads all bookings made by customers.
Error Handling	If the admin is not signed in, the system redirects them to the admin login page.

2. System design

This section outlines various design wireframes and principles considered in developing the system.

2.1. System Overview

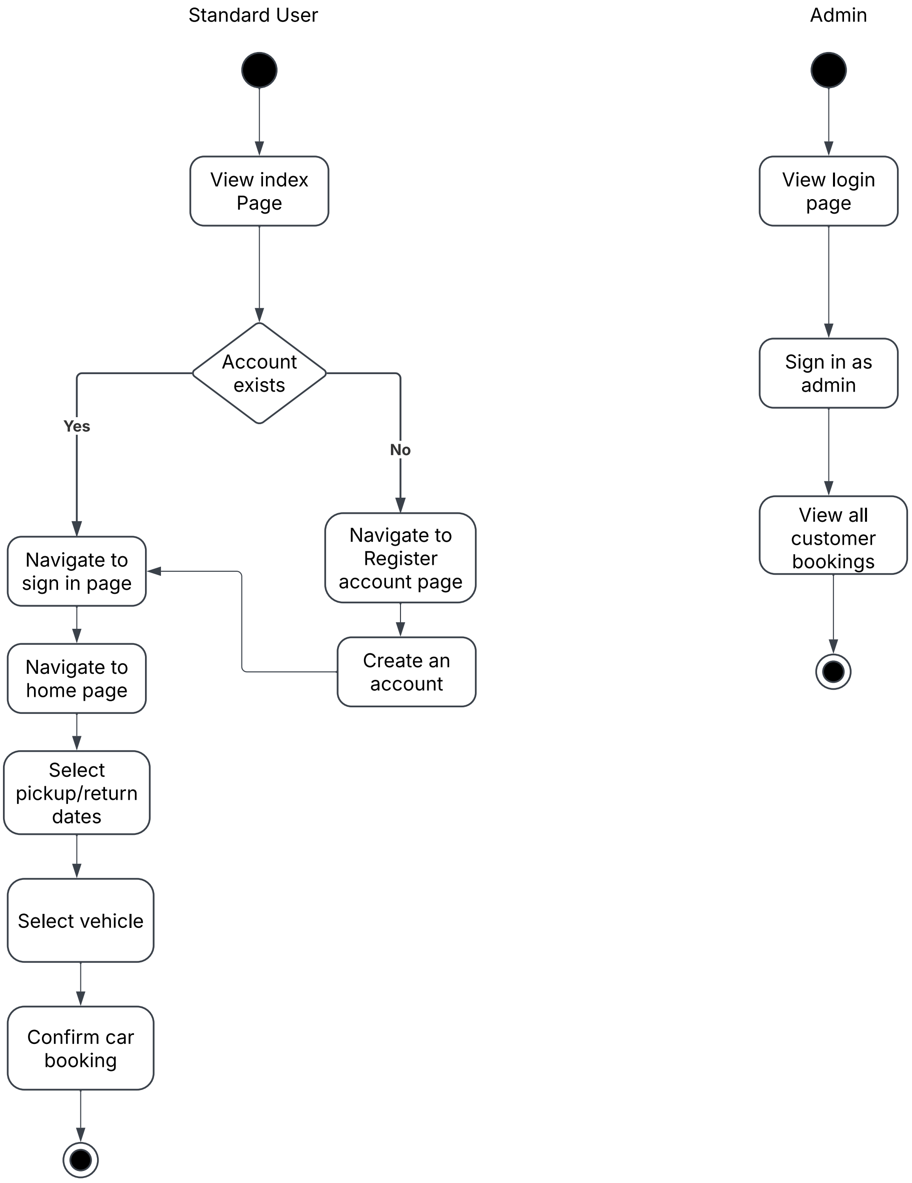


Figure 1: User and admin flow chart

2.2. Use case diagram

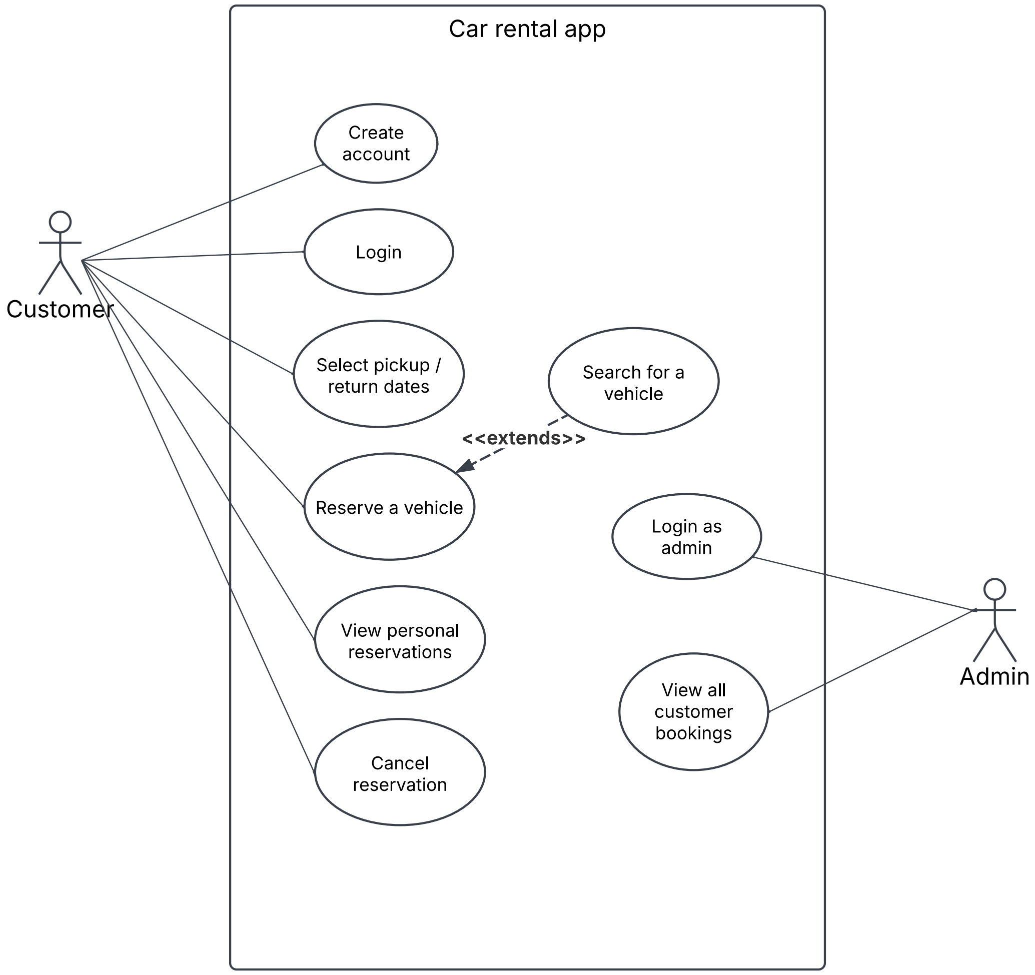
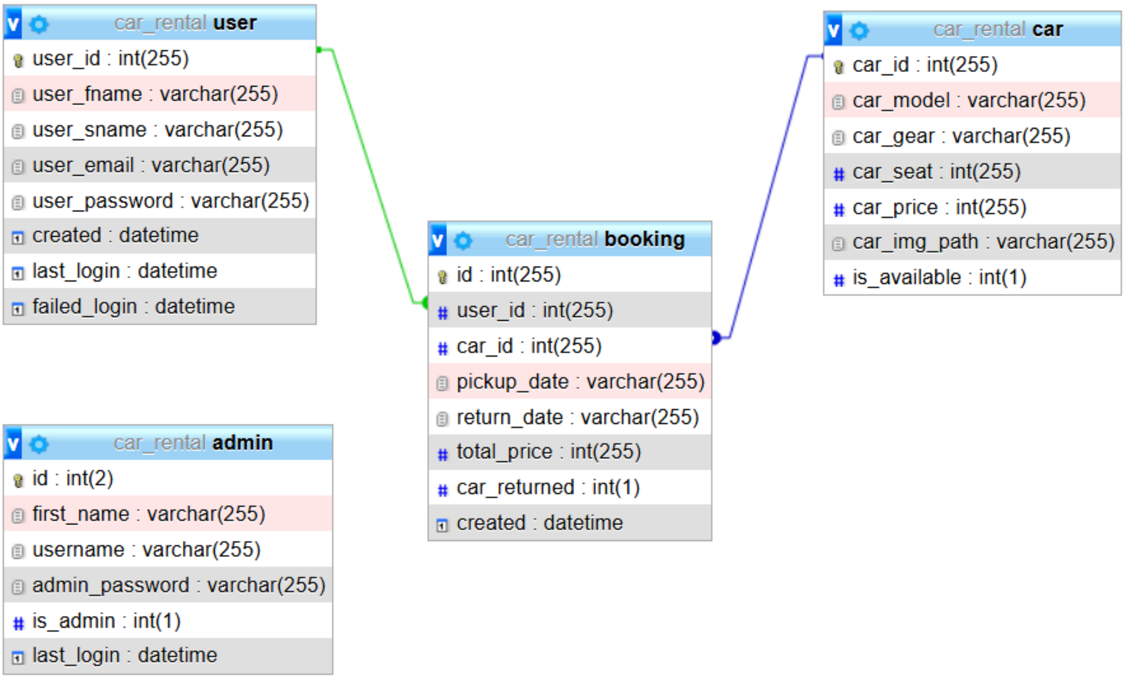


Figure 2: System Use Case Diagram

2.3. Database schema



car_rental user
user_id : int(255)
user_fname : varchar(255)
user_sname : varchar(255)
user_email : varchar(255)
user_password : varchar(255)
created : datetime
last_login : datetime
failed_login : datetime

car_rental booking
id : int(255)
user_id : int(255)
car_id : int(255)
pickup_date : varchar(255)
return_date : varchar(255)
total_price : int(255)
car_returned : int(1)
created : datetime

car_rental car
car_id : int(255)
car_model : varchar(255)
car_gear : varchar(255)
car_seat : int(255)
car_price : int(255)
car_img_path : varchar(255)
is_available : int(1)

car_rental admin
id : int(2)
first_name : varchar(255)
username : varchar(255)
admin_password : varchar(255)
is_admin : int(1)
last_login : datetime

Figure 3: Database schema outlining relationships

2.4. Secure Design Principles

This section outlines various secure principles used to design the web application. We referred to OWASP’s secure product design cheat sheet as a guide to ensure the developed web application meets appropriate security requirements:

2.4.1.Least Privilege

Outside customers should have the least amount of functionality to rent out a vehicle. Clients are only expected to view their specific bookings and additional information can only be viewed by the system administrator. This enhances the system’s security by limiting unauthorised access to private data such as the clients and the cars they have booked, since users will only access the information and functionality they need. Only administrators should be able to have an overview of booking information contained in the system.

2.4.2.Defence in Depth

The web application should apply multiple security measures to protect the data contained within the application. The system should implement input validation to guard against SQL injection or cross-site (XSS) scripting which can enable an attacker to gain unauthorised access into a user’s information. Enforcing strong password policies would also guard a user’s account against brute force or dictionary attacks. Applying appropriate role-based access also ensures users have access to their specific roles as a customer and not access other information intended for the admin. Layered security is beneficial for the system because it creates a contingency for a security mechanism in case an attacker bypasses it.

2.4.3.Fail securely

The system should manage an error or failure without compromising its security. We considered this principle to ensure the system does not reveal vulnerable functionality which can be exploited by a malicious actor. An example would be when an invalid entry discloses a null query, which exposes the system is using an SQL database. This enhances the system's security because without this technical information, an attacker cannot gather insight about the system and limits the approaches they could take to exploit the web application.

2.4.4.Secure the weakest link

The web application should apply a security mechanism to guard the weakest security point. New users might create their user accounts with easy to remember passwords which lack necessary complexity and can be easy to crack. Hanamsagar, et al. (2016) found that most online users use weak passwords which could be easily brute forced. Additionally, they found that weak password-length policies also contribute to these ineffective passwords. The system should therefore enforce a strong password policy which asks for more complex password combinations. The system's security would be enhanced by compelling prospective users to use complex passwords, which reduces its ability to be cracked by an attacker.

2.4.5.Logging mechanisms

The system should have a mechanism of logging security events. OWASP's cheat sheet authors highlight that system logging is necessary for identifying security incidents or monitoring policy violations. These mechanisms increase the security of the system by creating a means of tracing a security event such as a failed sign in attempt, which might signal a potential attack.

3. Development

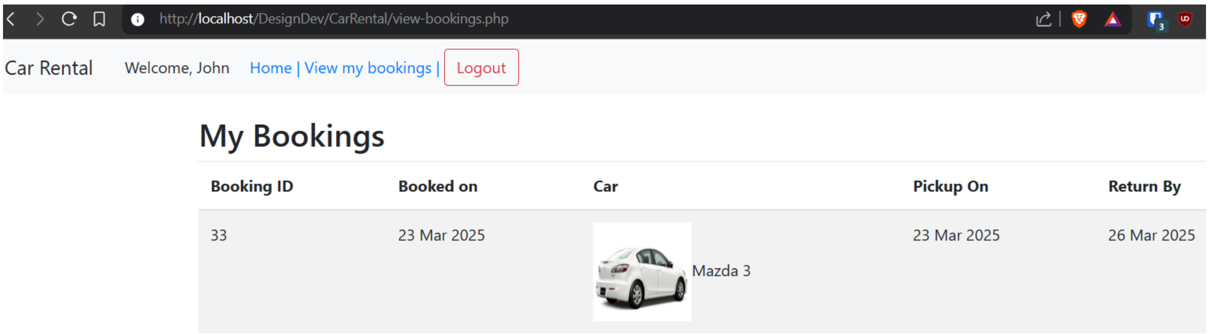
This section reviews the security features and principles used to develop the web application.

3.1. Implemented security features

The following outlines various security features implemented within the web application:

3.1.1.Access Control

A customer can only view bookings made using their specific account and cannot access a complete list of bookings made by other customers.


http://localhost/DesignDev/CarRental/view-bookings.php
Car Rental
Welcome, John
Home | View my bookings |
Logout

My Bookings

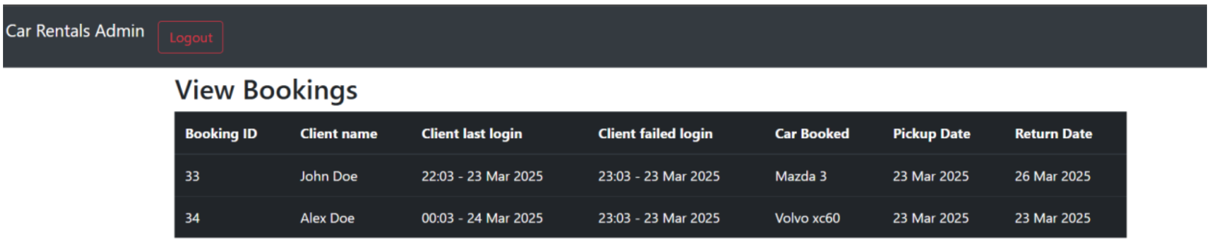
Booking ID	Booked on	Car	Pickup On	Return By
33	23 Mar 2025	Mazda 3	23 Mar 2025	26 Mar 2025

Figure 4: User 'John' viewing their respective bookings

Only admins are able to view the list of bookings made by customers because they are expected to review them. Admins are also restricted to their respective view to monitor a customer's client activity on the system.

3.1.2.Logging of security events

The system implements this principle by logging a timestamp when a user provides an invalid password for their account. This information is accessible to the admin who can monitor an account for suspicious behaviour.



View Bookings

Booking ID	Client name	Client last login	Client failed login	Car Booked	Pickup Date	Return Date
33	John Doe	22:03 - 23 Mar 2025	23:03 - 23 Mar 2025	Mazda 3	23 Mar 2025	26 Mar 2025
34	Alex Doe	00:03 - 24 Mar 2025	23:03 - 23 Mar 2025	Volvo xc60	23 Mar 2025	23 Mar 2025

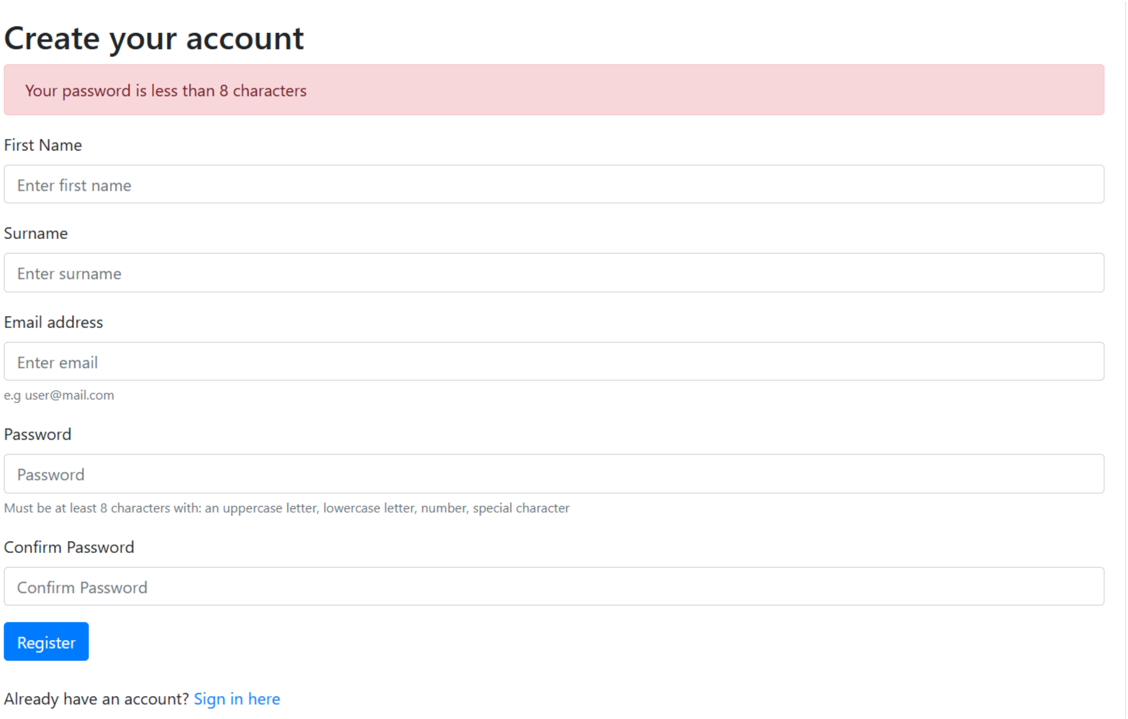
Figure 5: Admin page showing customer bookings information with their login attempts

3.1.3.Minimise Attack Surface Area

The system minimises the attack surface area by only loading the user’s credentials during the authentication process. When a customer is signing in, the system only queries their email and password from the database instead of returning the user’s complete record. This reduces the likelihood of their information leaking in the event of a compromise.

3.1.4.Strong Password Policies

The system implements a password complexity check when creating an account. It specifies a user requires at least 8 characters that can either be a number, special character, upper case or lowercase letter. Password complexity is useful in preventing brute force and dictionary attacks against a user’s account as more complex passwords. Using less than 8 characters rejects the input and throws an error message.



Create your account

Your password is less than 8 characters

First Name

Enter first name

Surname

Enter surname

Email address

Enter email

e.g user@mail.com

Password

Password

Must be at least 8 characters with: an uppercase letter, lowercase letter, number, special character

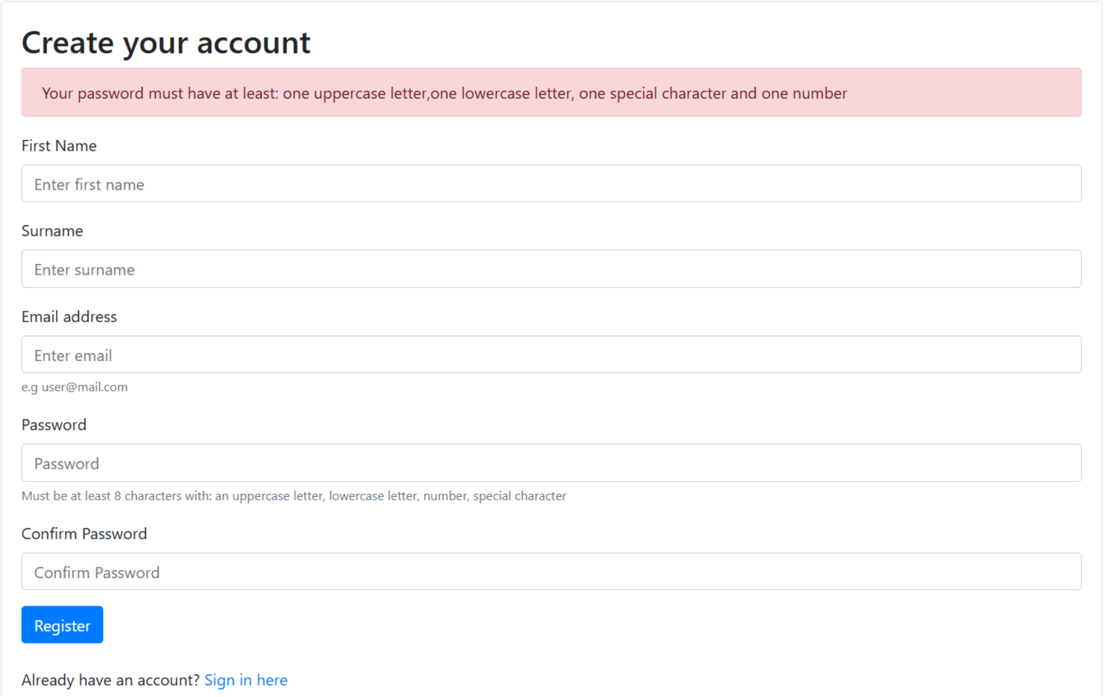
Confirm Password

Confirm Password

Already have an account? Sign in here

Figure 6: System throwing error message about invalid password

Using a password without necessary complexity throws an error message specifying the user needs to use multiple character types.



Create your account

Your password must have at least: one uppercase letter,one lowercase letter, one special character and one number

First Name

Enter first name

Surname

Enter surname

Email address

Enter email

e.g user@mail.com

Password

Password

Must be at least 8 characters with: an uppercase letter, lowercase letter, number, special character

Confirm Password

Confirm Password

Already have an account? Sign in here

Figure 7: System informing user they need to use a complex password

Additionally, when a user attempts to sign in, invalid attempts will issue an “incorrect email or password” message. This keeps the invalid part of the request ambiguous, protecting the user’s credentials from a potential brute force attack. These controls are enforced at a server level.

3.2. Security Standards

The developed system follows the OWASP top 10 standard for secure applications by guarding against:

3.2.1.Broken Access Control

The system’s server uses a custom session identifier to differentiate a user and an admin to prevent a violation of least privilege. Users cannot view or perform operations in other user’s accounts. They also cannot access admin pages as a normal user.

3.2.2.Injection Attacks

The system applies server-side input validation to ensure the user submitted input does not cause SQL injections.

3.2.3.Logging and monitoring

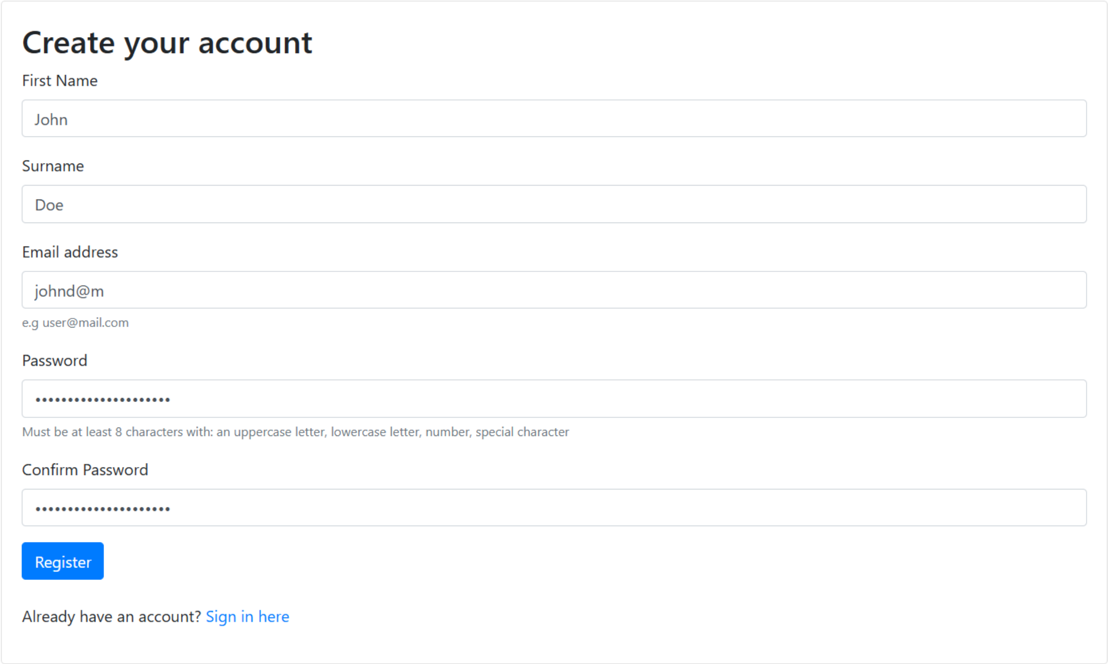
In order to provide an admin with information leading up to a breach, the system implements logging of a customer’s failed logins. This would help in identifying accounts linked with suspicious activities.

3.3. Secure programming principles

We developed this web application referencing OWASPs Secure Coding Practices guide and the following outline the coding practices we implemented:

3.3.1.Input validation

User input is validated on the server side before being stored or queried from the database. When creating an account, the application checks whether the email address provided has a valid format and rejects invalid responses.



Create your account

First Name

John

Surname

Doe

Email address

johnd@m

e.g user@mail.com

Password

••••••••••••••••••••

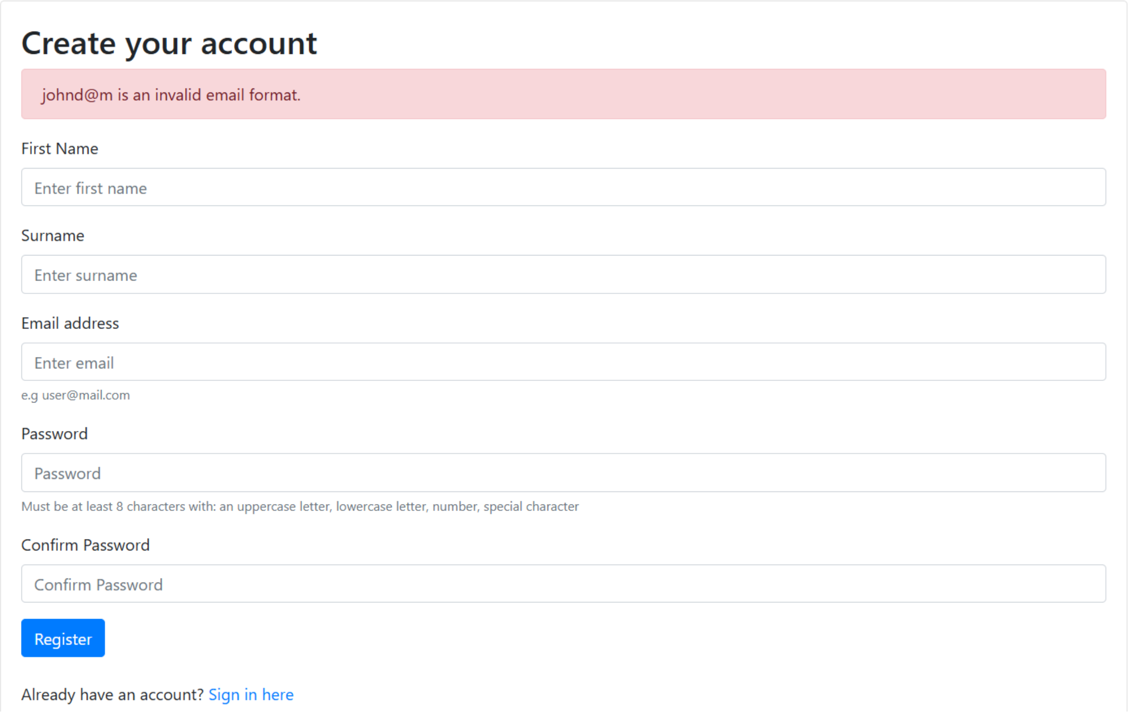
Must be at least 8 characters with: an uppercase letter, lowercase letter, number, special character

Confirm Password

••••••••••••••••••••

Already have an account? Sign in here

Figure 8: User trying to create account with the email 'john@m'



Create your account

johnd@m is an invalid email format.

First Name

Enter first name

Surname

Enter surname

Email address

Enter email

e.g user@mail.com

Password

Password

Must be at least 8 characters with: an uppercase letter, lowercase letter, number, special character

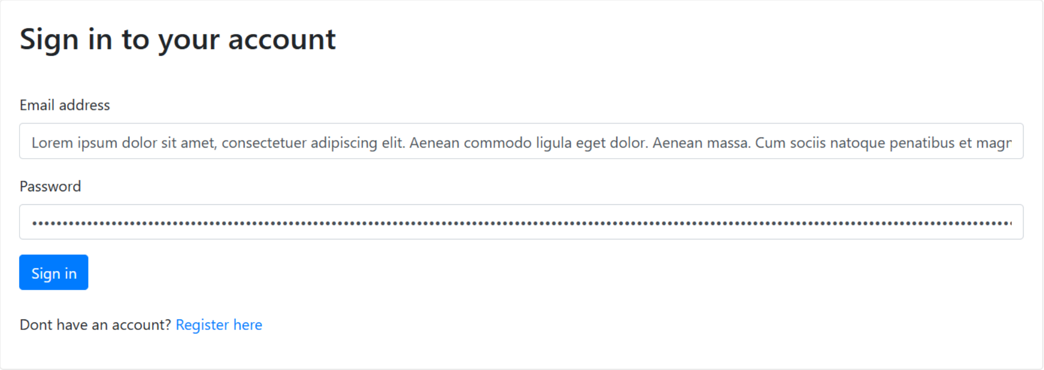
Confirm Password

Confirm Password

Already have an account? Sign in here

Figure 9: System sends error message that the email provided is invalid

This ensures the data passed is a valid string to guard against SQL or XSS injection attacks. The system also validates the length of user inputs to ensure they are not more than 255 characters. This aids in guarding the system against buffer overflows where an attacker can attempt to input a value with more characters than the limits specified in the database.



Sign in to your account

Email address

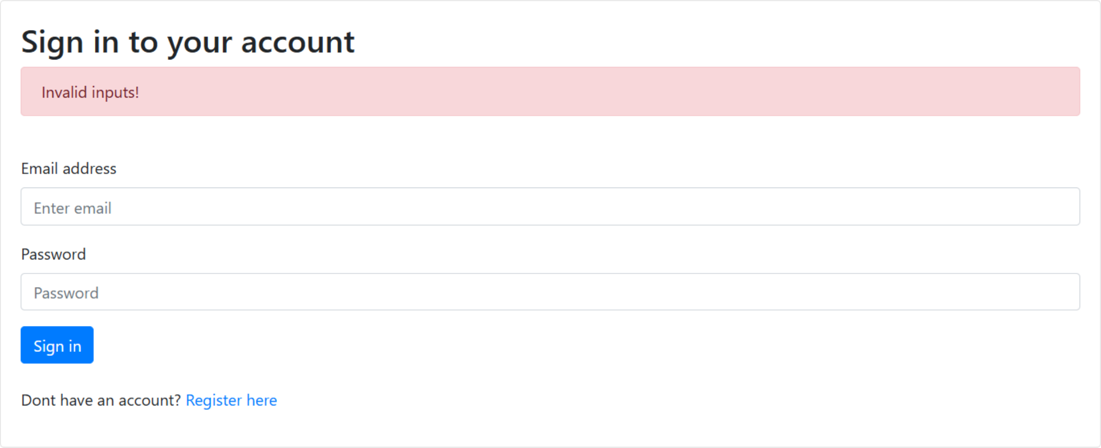
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magr

Password

••

Dont have an account? Register here

Figure 10: Simulating adding a large input to attempt an overflow



Sign in to your account

Invalid inputs!

Email address

Enter email

Password

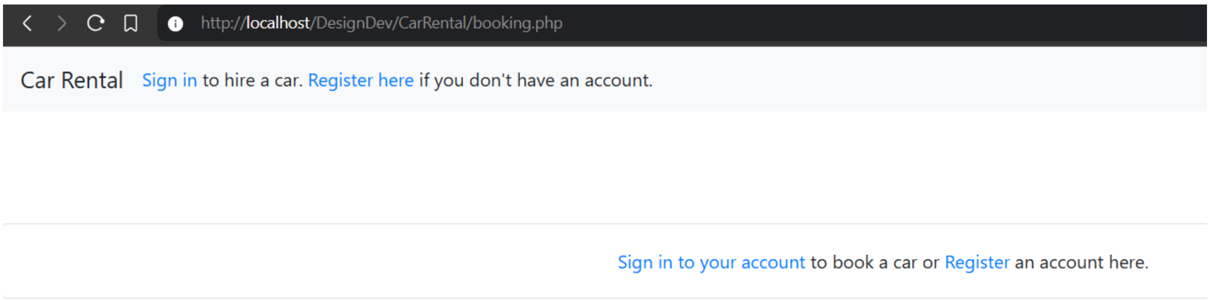
Password

Dont have an account? Register here

Figure 11: System rejecting input more than 255 characters

3.3.2.Authentication and Password Management

The system requires clients to sign in to access private pages. A user requires an active session to be able to initiate the booking process under their account. Without a valid session, the system notifies the user to sign to access these pages.



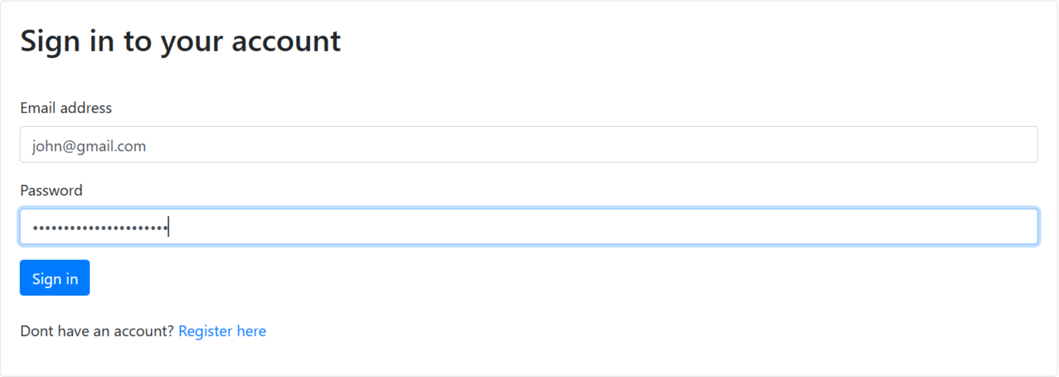
http://localhost/DesignDev/CarRental/booking.php

Car Rental Sign in to hire a car. Register here if you don't have an account.

Sign in to your account to book a car or Register an account here.

Figure 12: System notifying user they need to log in before they can access booking page

Password fields are obscured when a user is interacting with them on the screen. This enhances the systems security by guarding the user against shoulder surfing where an attacker views the victim's credentials.



Sign in to your account

Email address

john@gmail.com

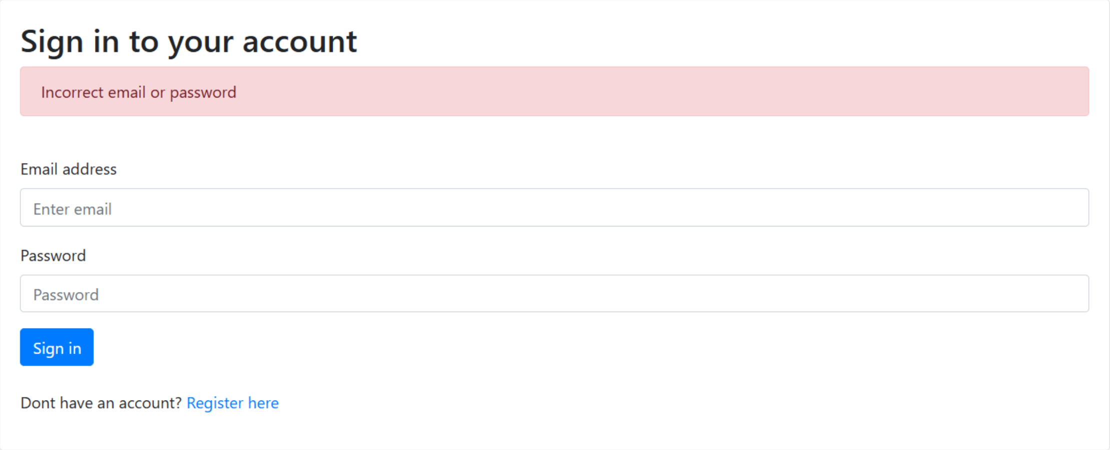
Password

••••••••••••••••••••••

Dont have an account? Register here

Figure 13: Passwords are submitted using password input

The system also enforces password attempt limits. After multiple unsuccessful password attempts, the system informs the user that they are unable to log in the system. This protects the customer's credentials by increasing the time and effort an attacker requires to gain access to the account.


Sign in to your account

Incorrect email or password

Email address

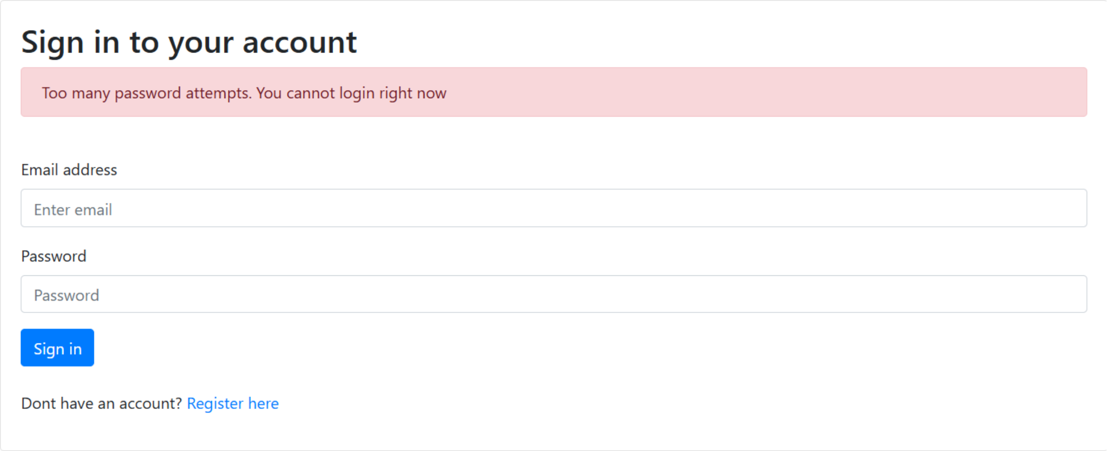
Enter email

Password

Password

Dont have an account? Register here

Figure 14: First invalid login attempt


Sign in to your account

Too many password attempts. You cannot login right now

Email address

Enter email

Password

Password

Dont have an account? Register here

Figure 15: Error message after multiple wrong password submissions

3.3.3.Session Management

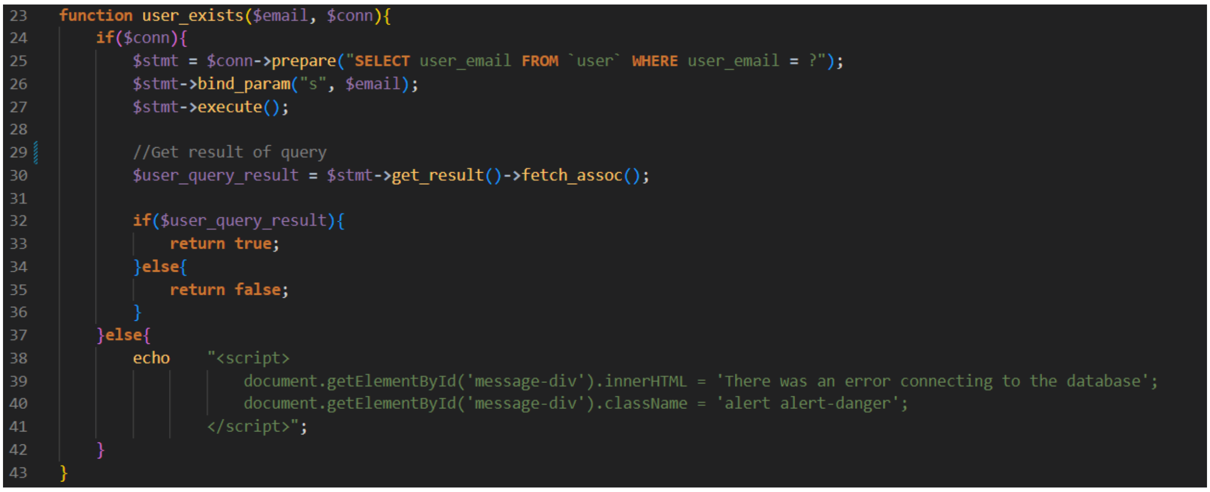
The server side creates a user’s session identifier after the user has successfully authenticated themselves on the system. After the session identifier is created the user cannot access the login page, which helps prevents the same user creating concurrent sessions. Once logged in, there is a logout button which terminates the current user’s active session. This button is available to the active user on all pages throughout their session.

4. Testing and Analysis

The following captures the testing and analysis of various functions found within the web application.

4.1. User Exists() function

This section analyses the function **user_exists**, with two arguments: **email** and **conn** which checks whether a user with the given email exists in the database. The code referenced is in the figure below:



```
function user_exists($email, $conn){
    if($conn){
        $stmt = $conn->prepare("SELECT user_email FROM `user` WHERE user_email = ?");
        $stmt->bind_param("s", $email);
        $stmt->execute();

        //Get result of query
        $user_query_result = $stmt->get_result()->fetch_assoc();

        if($user_query_result){
            return true;
        }else{
            return false;
        }
    }else{
        echo    "<script>
                    document.getElementById('message-div').innerHTML = 'There was an error connecting to the database';
                    document.getElementById('message-div').className = 'alert alert-danger';
                </script>";
    }
}
```

Figure 16: user_exists() on code editor

4.1.1.Data Flow Analysis

The function relies on two parameters, **email**, the user's email and **conn,** a mysqli object which opens a connection to the system's MySQL server.

No	Reference	Definition
1		email, conn
2	conn	
3	conn	stmt
4	stmt, email	
5	stmt	
6	stmt	user_query_result
7	user_query_result	
8		true
10		false
11		
12		echo "There was an error"
13		

The corresponding control flow graph is:

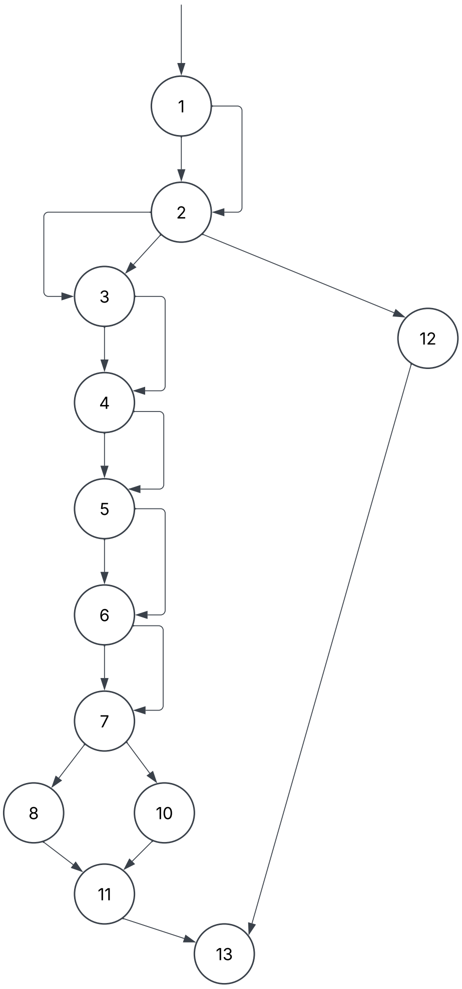


Figure 17: CFG for user_exists()

There are 3 potential paths to be followed:

Path A: 1,2,3,4,5,6,7,8,11,13

Path B: 1,2,3,4,5,6,7,10,11,13

Path C: 1,2,12,13

4.1.2.Test Case 1

email	**conn**
“john@gmail.com” (existing email)	true

4.1.2.1. Statement coverage

Using the test case, statements 1, 2, 3, 4, 5, 6, 7, 8 will be executed because both **email** and **conn** parameters return true:

- Statement 1: Reads both the **email** and **conn** parameter.
- Statement 2: Checks whether **conn** is true.
- Statement 3: Defines a new variable **stmt** and refers **conn** to prepare an SQL statement
- Statement 4: Uses the **email** parameter as a value of the SQL prepared statement
- Statement 5: Uses the **stmt** variable to execute the SQL query
- Statement 6: Defines new variable **user_query_result** and assigns the SQL query’s response to it
- Statement 7: Checks whether the value of **user_query_result** is true
- Statement 8: Returns **true** because **user_query_result** is true, as we using an existing email address.

4.1.2.2. Edge coverage

The following edges were covered by the test case: 1-2, 2-3, 3-4, 4-5, 6-7, 7-8. Not all edges have been covered using this test cases.

4.1.2.3. Path coverage

The test case covered path A with nodes: 1,2,3,4,5,6,7,8,11,13.

4.1.3.Test Case 2

email	**conn**
“john@gmail.com” (existing email)	False / null

4.1.3.1. Statement coverage

The test case covers the following statements: 1, 11, 12

- Statement 1: Reads both the **email** and **conn** parameter
- Statement 11: Executes the else block because **conn** returns false
- Statement 12: Prints the message showing there was an error connecting to the database

4.1.3.2. Edge coverage

The following edges were covered 1-2, 2-12. Not all edges were covered using the test case.

4.1.3.3. Path Coverage

The test case covered path C with nodes 1,2,12,13.

4.1.4.Test Case 3

email	**conn**
“new@mail.com” (non-existing email)	true

4.1.4.1. Statement coverage

The test case coverage the following statements: 1, 2, 3, 4, 5, 6, 9, 10

- Statement 1: Reads both the **email** and **conn** parameter.
- Statement 2: Checks whether **conn** is true.
- Statement 3: Defines a new variable **stmt** and refers **conn** to prepare an SQL statement
- Statement 4: Uses the **email** parameter as a value of the SQL prepared statement
- Statement 5: Uses the **stmt** variable to execute the SQL query
- Statement 6: Defines new variable **user_query_result** and assigns the SQL query’s response to it
- Statement 7: Checks whether the value of **user_query_result** is true
- Statement 9: Runs the else block because **user_query_result** is false.
- Statement 10: Returns **false** because the email we provided did not exist in the database.

4.1.4.2. Edge coverage

The test case coverage the edges: 1-2, 2-3, 3-4. 4-5, 5-6, 6-7, 7-10, 10-11, 11-13. The case did not cover all the edges.

4.1.4.3. Path coverage

The test case covered path B with nodes: 1,2,3,4,5,6,7,10,11,13

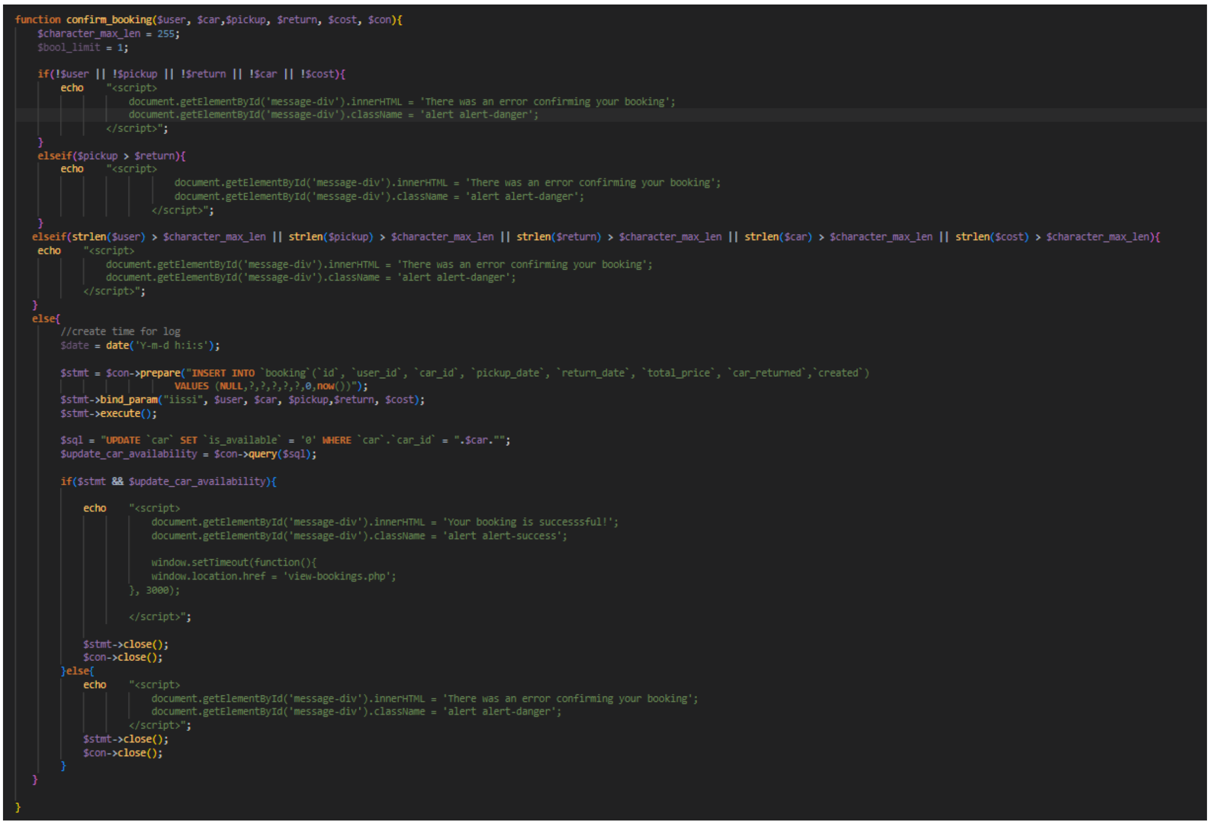
4.1.5.Recommendations

The user_exists function does not validate the email address independently. While other functions that invoke it have their input validation mechanisms, adding email input validation in this function helps guard against injection attacks. If user_exists() is called and the caller lacks email validation, this might present an injection vulnerability when submitting queries to the database.

4.2. Confirm_booking() function

This function is used to confirm the user’s booking information. It has 6 arguments:

- **user** : reads the user’s id
- **car**: reads the car’s id
- **pickup**: reads the picking up date
- **return**: reads the returning date
- **cost**: reads the cost for renting the car between the pickup and returning date
- **conn**: reads the Boolean value of attempting to create a DB connection



```php
function confirm_booking($user, $car,$pickup, $return, $cost, $con){
    $character_max_len = 255;
    $bool_limit = 1;

    if(!$user || !$pickup || !$return || !$car || !$cost){
        echo    "<script>
                    document.getElementById('message-div').innerHTML = 'There was an error confirming your booking';
                    document.getElementById('message-div').className = 'alert alert-danger';
                </script>";
    }
    elseif($pickup > $return){
        echo    "<script>
                            document.getElementById('message-div').innerHTML = 'There was an error confirming your booking';
                            document.getElementById('message-div').className = 'alert alert-danger';
                        </script>";
    }
   elseif(strlen($user) > $character_max_len || strlen($pickup) > $character_max_len || strlen($return) > $character_max_len || strlen($car) > $character_max_len || strlen($cost) > $character_max_len){
    echo    "<script>
                document.getElementById('message-div').innerHTML = 'There was an error confirming your booking';
                document.getElementById('message-div').className = 'alert alert-danger';
            </script>";
   }
   else{
        //create time for log
        $date = date('Y-m-d h:i:s');

        $stmt = $con->prepare("INSERT INTO `booking`(`id`, `user_id`, `car_id`, `pickup_date`, `return_date`, `total_price`, `car_returned`,`created`)
                            VALUES (NULL,?,?,?,?,?,0,now())");
        $stmt->bind_param("iissi", $user, $car, $pickup,$return, $cost);
        $stmt->execute();

        $sql = "UPDATE `car` SET `is_available` = '0' WHERE `car`.`car_id` = ".$car."";
        $update_car_availability = $con->query($sql);

        if($stmt && $update_car_availability){

            echo    "<script>
                        document.getElementById('message-div').innerHTML = 'Your booking is successsful!';
                        document.getElementById('message-div').className = 'alert alert-success';

                        window.setTimeout(function(){
                        window.location.href = 'view-bookings.php';
                    }, 3000);

                    </script>";

            $stmt->close();
            $con->close();
        }else{
            echo    "<script>
                        document.getElementById('message-div').innerHTML = 'There was an error confirming your booking';
                        document.getElementById('message-div').className = 'alert alert-danger';
                    </script>";
            $stmt->close();
            $con->close();
        }
   }

}
```

Figure 18: Complete confirm_booking() function

4.2.1.Data flow analysis

The following covers the data flow of the various arguments within the function.

No	Reference	Definition
1		user, car, pickup, return, cost, con
2		character_max_length
3		bool_limit
4	user, car, pickup, return, cost	
5		Echo “error confirming your booking”
6	pickup, return	
7		Echo “error confirming your booking”
8	user, car, pickup, return, cost	
9		Echo “error confirming your booking”
10		

11		date
12	con	stmt
13	stmt	
14	stmt	
15	car	sql
16	con, sql	Update_car_availability
17	sql, update_car_availability	
18		Echo “Booking successful”
19	stmt	
20	con	
21		
22		Echo “Error confirming booking”
23	stmt	
24	con	
25		
26		

The control flow graph for the function is:

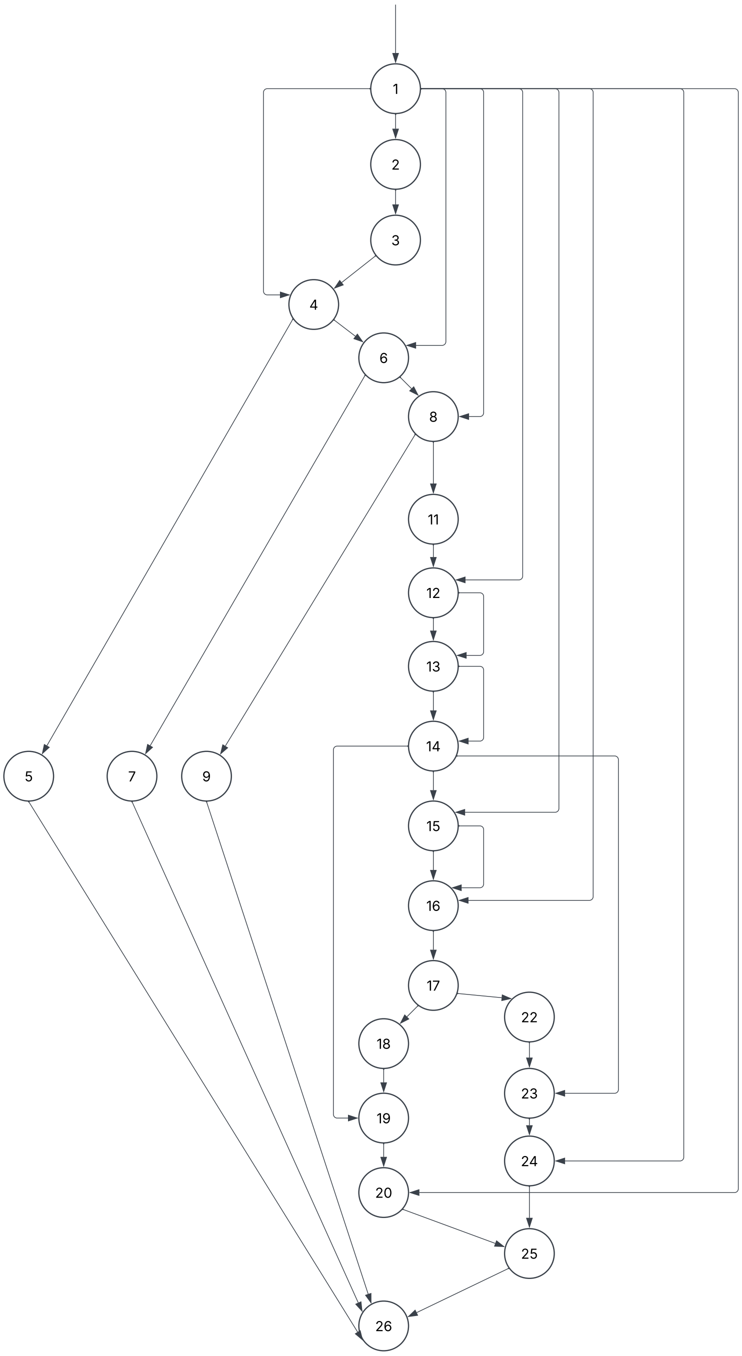


Figure 19: CFG for confirm_booking()

The paths for the control path graph are:

Path A: 1, 2, 3, 4, 5, 26

Path B: 1, 2, 3, 4, 6, 7, 26

Path C: 1, 2, 3, 4, 6, 8, 9, 26

Path D: 1, 2, 3, 4, 6, 8, 11,12, 13,14, 15, 16, 17, 18, 19, 20, 25, 26

Path E: 1, 2, 3, 4, 6, 8, 11,12, 13,14, 15, 16, 17, 22, 23, 24, 25, 26

4.2.2.Test Case 1

user	car	pickup	return	cost	con
80	2	2025-3-23	2025-3-26	30	true

4.2.2.1. Statement coverage

The following statements will be executed by the function:

- Statement 1: Reads the parameters of **user**, **car**, **pickup**, **return**, **cost** and **con**
- Statement 2: Declares the variable **character_max_limit** and a value **255** is assigned to it
- Statement 3: Declares the variable **bool_limit** and the value 1 is assigned to it
- Statement 10: Opens the else block
- Statement 11: Declares the variable **date** and assigns a date value
- Statement 12: Declares the variable **stmt** and references **con** to prepare an SQL statement
- Statement 13: References **stmt**, **user**, **car**, **pickup** and **return** to apply the booking parameters to the prepared SQL statement
- Statement 14: Executes the SQL statement
  Statement 15: Declares a **sql** variable and a query is assigned to it
- Statement 16: Declares variable **update_car_availability** and assigns the executed query. The query references the **sql** variable
- Statement 17: Checks whether the **update_car_availability** and **stmt** queries return a true (execute successfully)
- Statement 18: Prints a successful booking message.
- Statement 19: Closes the **stmt** variable for running queries
- Statement 20:Closes the **con** connection variable.
- Statement 25:Closes the else block
- Statement 26: Closes the function block

4.2.2.2. Edge Coverage

The test case covers the edges: 1-2, 2-3, 3-4, 4-6, 6-8, 8-11, 11-12, 12-13, 13-14, 14-15, 15-16, 16-17, 17-18, 18-19, 19-20, 20-25, 25-26.

4.2.2.3. Path coverage

The test case covers path D with nodes: 1, 2, 3, 4, 6, 8, 11,12, 13,14, 15, 16, 17, 18, 19, 20, 25, 26

4.2.3.Test Case 2

user	car	pickup	return	cost	con
80	2	2025-3-23	2025-3-20	30	true

4.2.3.1. Statement coverage

The following statements will be executed in this test case:

- Statement 1: Reads the parameters of **user**, **car**, **pickup**, **return**, **cost** and **con**
- Statement 2: Declares the variable **character_max_limit** and a value **255** is assigned to it
- Statement 3: Declares the variable **bool_limit** and a value **1** is assigned to it
- Statement 6: Checks whether the date in the **pickup** variable is more recent than that of the **return** variable. The result is true.
- Statement 7: Prints a statement that there was an error confirming the booking
- Statement 26: Closes the function.

4.2.3.2. Edge coverage

The test case covers the edges: 1-2, 2-3, 3-4, 4-6, 6-7, 7-26. Not all edges were covered.

4.2.3.3. Path coverage

The test case covered path B with nodes: 1, 2, 3, 4, 6, 7, 26.

4.2.4.Test Case 3

user	car	pickup	return	cost	con
NULL	2	2025-3-23	2025-3-24	30	true

4.2.4.1. Statement coverage

The following statements will be executed in the test case:

- Statement 1: Reads the parameters of **user**, **car**, **pickup**, **return**, **cost** and **con**
- Statement 2: Declares the variable **character_max_limit** and a value **255** is assigned to it
- Statement 3: Declares the variable **bool_limit** and a value **1** is assigned to it
- Statement 4: Checks whether any of the parameters are false. Returns false because the user parameter is null
- Statement 5: Prints a statement that there was an error confirming the booking
- Statement 26: Closes the function.

4.2.4.2. Edge coverage

The test case covers the edges: 1-2, 2-3, 3-4, 4-5, 5-26. Not all edges were covered.

4.2.4.3. Path coverage

The test case follows the path A with nodes: 1, 2, 3, 4, 5, 26.

4.2.5.Recommendations

The function should be split into smaller functions to reduce its complexity. Increased complexity increases the risk of errors which might compromise the security of the overall application. There was a variable, **bool_limit** on line 3 and **date** on line 11 which was declared but not used in the function.

5. Formal Methods

This section demonstrates the use of formal methods to specify the behaviour of the system as well as verify its correctness. Since this rental application is a complex system, we used formal methods to analyse the system. Having a model that simplifies implementation details allows us to focus on providing a structure of how the system meets its functional requirements. (Woodcock, et al. 2009)

Developing a reliable system that constantly rises in complexity is a challenge. (Bowen and Hinchey 1995) The authors therefore conclude that applying formal methods helps address this challenge by using mathematical tools to design and produce trustworthy applications. They emphasize that formal languages make specifications clear enough to reduce ambiguities and gain necessary understanding about the system design.

We used petri nets to model the system's behaviour. We derived one big model to get a wholistic overview of the system to determine concurrent states within the system as well as identify any interdependent states. This model was derived using a high level of abstraction.

5.1. Petri Net model

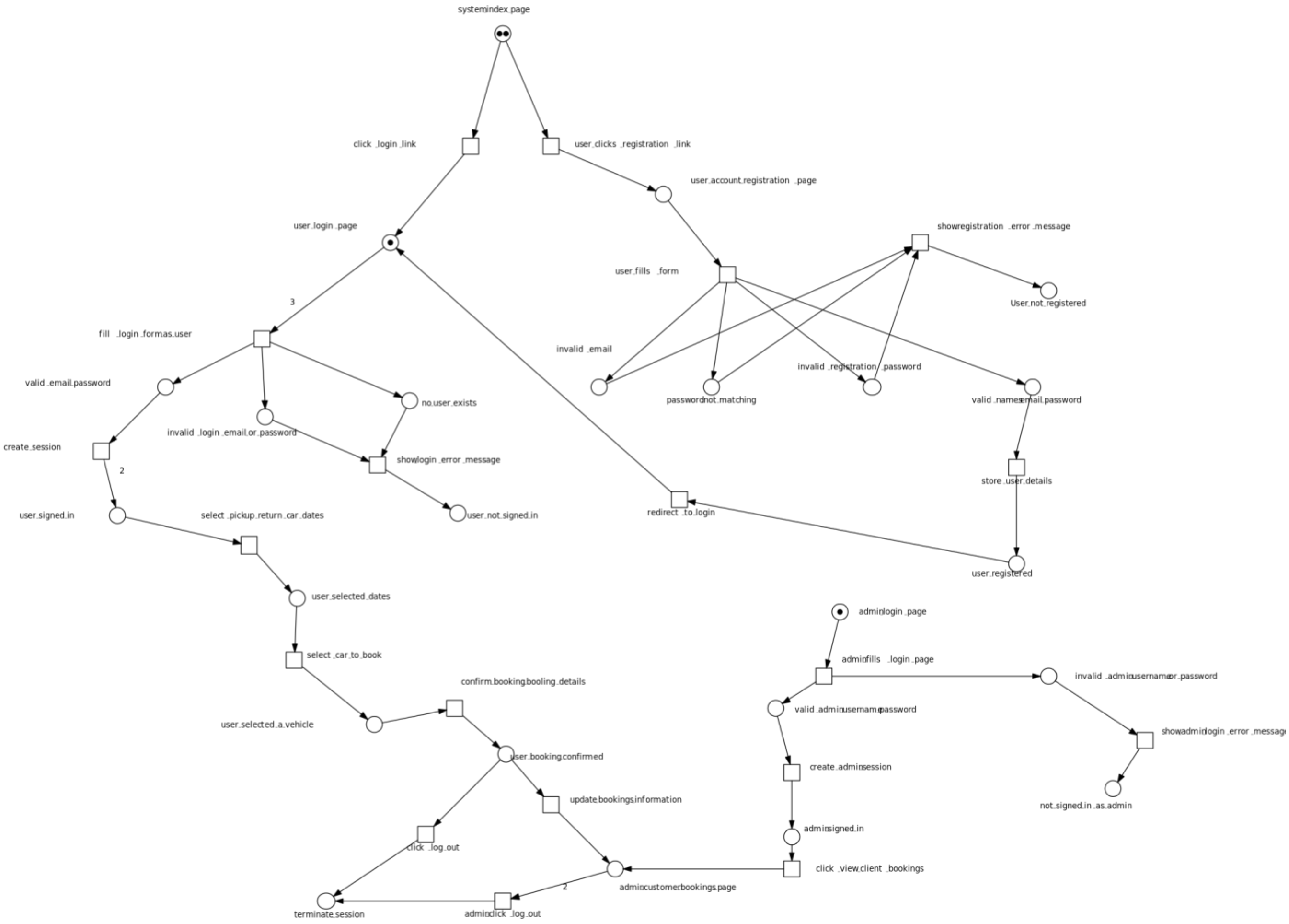


Figure 20: Entire system model

5.2. Formal Verification

This section outlines the analysis of the developed model referencing the system specifications. Each tested property contains its description and corresponding CTL equation.

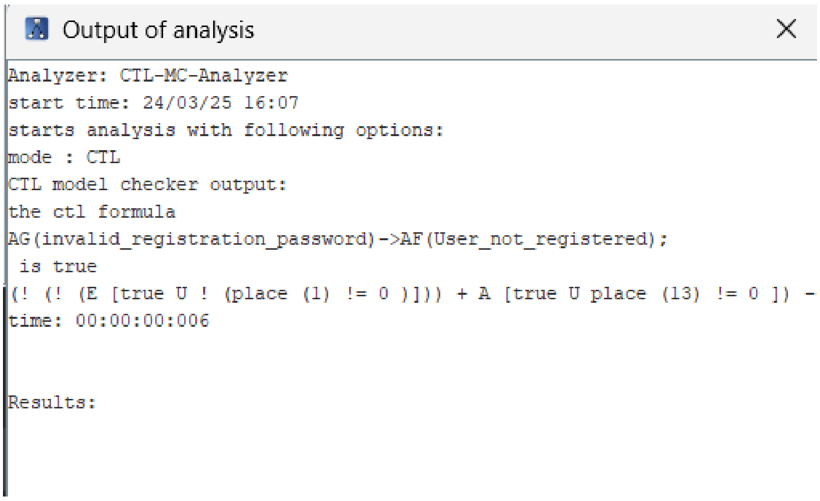
5.2.1.<u>Checking invalid password during registration</u>

The system checks whether the system contains an invalid password when a user attempts to create an account. A user using an invalid password (less than 8 characters long, without a mix of uppercase, lowercase, special characters and integers) during registration should be prevented from creating an account. Therefore for all states within the system, when the user submits an invalid password, their account should not be registered. The corresponding CTL statement would be:

$$\mathbf{AG(invalid\_registration\_password) \Rightarrow AF(User\_not\_registered)}$$

- **AG(invalid_registration_password:** In all instances of the system where user submits an invalid password during registration…
- **AF(User_not_registered):** Eventually the user will not be registered within the system. The expected result of the equation is true

Running the analyser returns true, which shows the model captures the intended behaviour of the system of not registering the user if there is an invalid password.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:07
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(invalid_registration_password)->AF(User_not_registered);
 is true
(! (! (E [true U ! (place (1) != 0 )])) + A [true U place (13) != 0 ]) -
time: 00:00:00:006


Results:
```

Figure 21: AG(invalid_registration_password) ⇒ AF(User_not_registered) result

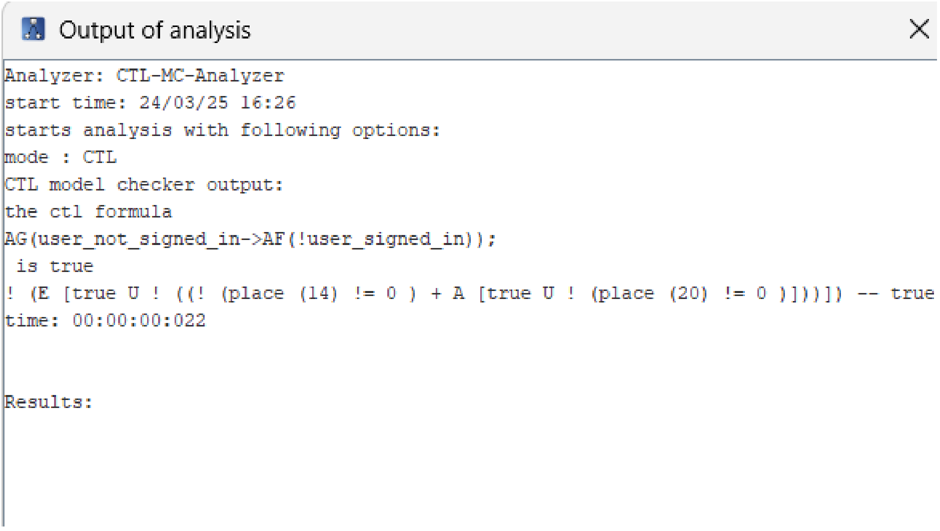
5.2.2.<u>Checking if user can access home page without being logged in</u>

The system is expected to redirect the user to their respective home page only after they have signed in. This means that if a user is not signed in, they system should not be able to navigate to the specific user’s home page. The corresponding CTL property would be:

AG(user_not_signed_in ⇒ AF(!user_signed_in))

- **AG(user_not_signed_in:** When a customer is not logged in…
- **AF(!user_signed_in):** the system will not sign in the user. The expected result is true.

Running the analyser returns true, which means the model returns the intended behaviour of the system of not accessing the user’s home page when the user is not signed in.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:26
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(user_not_signed_in->AF(!user_signed_in));
 is true
! (E [true U ! ((! (place (14) != 0 ) + A [true U ! (place (20) != 0 )]))]) -- true
time: 00:00:00:022


Results:
```

Figure 22: AG(user_not_signed_in ⇒ AF(!user_signed_in)) result

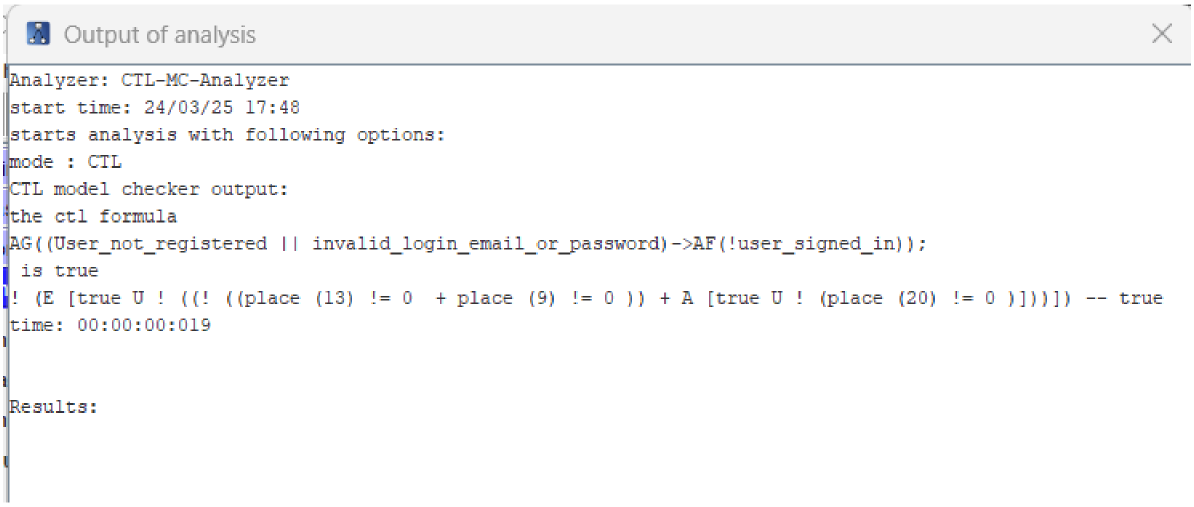
5.2.3.Check if either not having an account or inserting invalid credentials will log in the user

The system should not permit a customer without a registered account to log into the system. Additionally, it should also not permit users who attempt to sign in using the wrong email or password stored within the system. This CTL property would be:

AG((User_not_registered || invalid_login_email_or_password) ⇒ AF(!user_signed_in))

- **AG((User_not_registered || invalid_login_email_or_password)**: When the customer does not have an account OR inputs wrong credentials…
- **AF(!user_signed_in)):**The system will reject signing them in. The expected result is true.

Running the analyser returns true, meaning the model returns the intended behaviour of the system of not signing in the user without a valid account or valid credentials.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 17:48
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG((User_not_registered || invalid_login_email_or_password)->AF(!user_signed_in));
 is true
! (E [true U ! ((! ((place (13) != 0  + place (9) != 0 )) + A [true U ! (place (20) != 0 )]))]) -- true
time: 00:00:00:019


Results:
```

Figure 23: AG((User_not_registered || invalid_login_email_or_password) ⇒ AF(!user_signed_in)) result

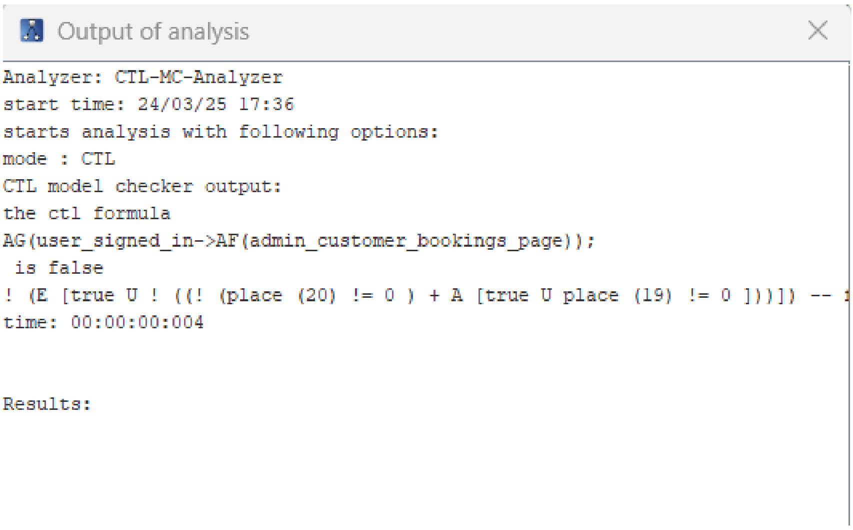
5.2.4. *Check if a user can view other user's bookings on the admin page*

The system should only enable the admin to view other customer's bookings. Customers are only restricted to their respective view and bookings. The CTL statement to verify this is:

AG(user_signed_in ⇒ AF(admin_customer_bookings_page))

- **AG(user_signed_in:** When the customer is signed in as a user…
- **AF(admin_customer_bookings_page)):** The system will redirect the customer to the admin page showing all the customers' bookings. We expect the result to be false.

The resulting analysis returns false, which verifies the model is accurately capturing how the system is supposed to prevent a user without admin privileges from accessing the admin page.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 17:36
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(user_signed_in->AF(admin_customer_bookings_page));
 is false
! (E [true U ! ((! (place (20) != 0 ) + A [true U place (19) != 0 ]))]) --
time: 00:00:00:004


Results:
```

Figure 24: AG(user_signed_in ⇒ AF(admin_customer_bookings_page)) result

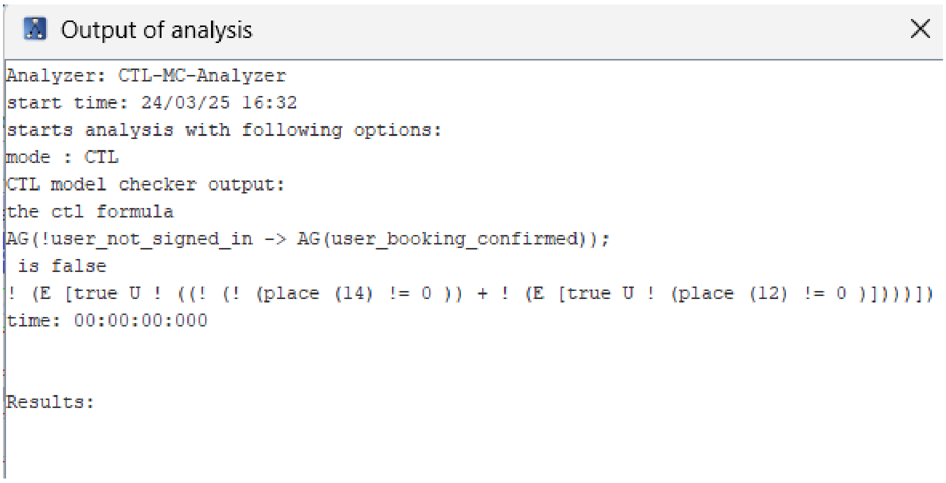
5.2.5.Check if user can book a car if they are not logged in

Once logged in, the user will be redirected to a home page where they can begin the booking process. The system should not allow the user to perform a car booking without having a valid session in the home page. The CTL statement to verify this is:

AG(!user_not_signed_in ⇒ AG(user_booking_confirmed))

- **AG(!user_not_signed_in:** When the user is not signed in…
- **AG(user_booking_confirmed)):** the user can perform a car booking. This should be false.

The resulting analysis returns false, showing the model captures the system’s expected functionality of not allowing a customer to book a car until they are signed in.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:32
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(!user_not_signed_in -> AG(user_booking_confirmed));
 is false
! (E [true U ! ((! (! (place (14) != 0 )) + ! (E [true U ! (place (12) != 0 )]))))])
time: 00:00:00:000


Results:
```

Figure 25:AG(!user_not_signed_in ⇒ AG(user_booking_confirmed)) result on analyser

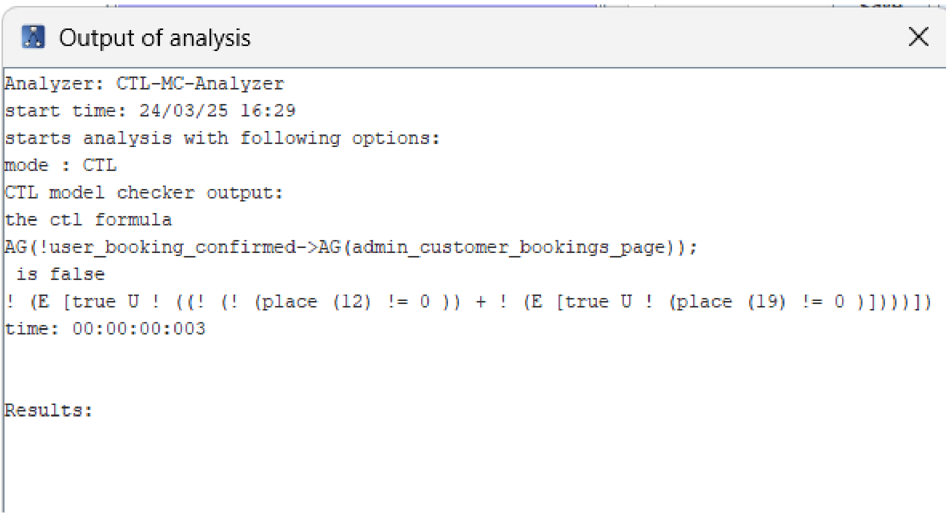
5.2.6. *Check if admin can view a user's booking information if the user has not booked a car*

Customers provide the information which will be reflected on the admin's page. Without a user booking, admins cannot review an updated page of booked vehicles. We derived the following CTL statetement:

AG(!user_booking_confirmed ⇒ AG(admin_customer_bookings_page))

- **AG(!user_booking_confirmed:** When a user has not confirmed a booking in the system…
- **AG(admin_customer_bookings_page)):** the admin should be able to view an updated page of bookings. We expected a false result.

The false result produced confirms the model captures the expected behaviour of not being able to view booking information that has not been submitted by the user.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:29
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(!user_booking_confirmed->AG(admin_customer_bookings_page));
 is false
! (E [true U ! ((! (! (place (12) != 0 )) + ! (E [true U ! (place (19) != 0 )]))))])
time: 00:00:00:003


Results:
```

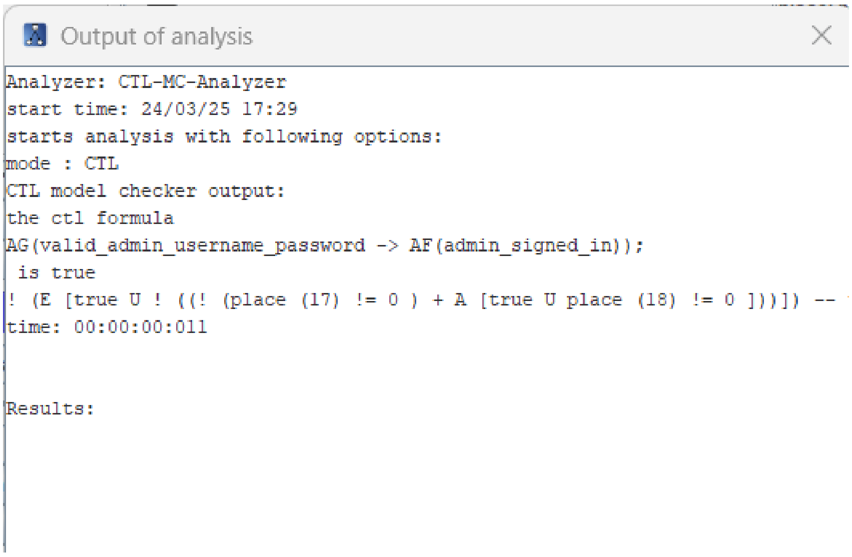
Figure 26: AG(!user_booking_confirmed ⇒ AG(admin_customer_bookings_page)) output

5.2.7. *Check if admin can access admin page*

The admin within the system has their own custom login page which redirects them to an admin home page after successful authentication. Using the correct credentials should redirect them to their respective view. The CTL statement used was:

AG(valid_admin_username_password ⇒ AF(admin_signed_in))

- **AG(valid_admin_username_password:** When the system captures a valid admin username and password…
- **AF(admin_signed_in)):** The system should sign them in as an admin. We expected the result to be true.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 17:29
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(valid_admin_username_password -> AF(admin_signed_in));
 is true
! (E [true U ! ((! (place (17) != 0 ) + A [true U place (18) != 0 ]))]) --
time: 00:00:00:011


Results:
```

Figure 27: Analysing AG(valid_admin_username_password ⇒ AF(admin_signed_in))

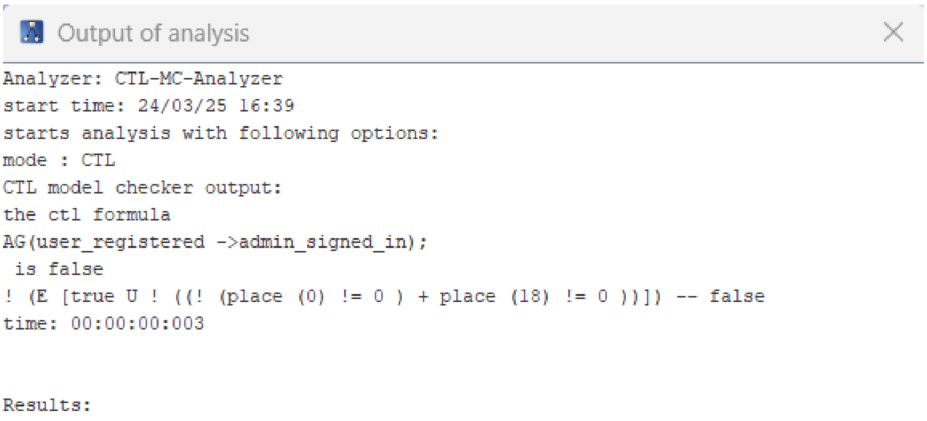
5.2.8. <u>Checking if registered user can sign in as admin</u>

The customer should not be able to sign in as an admin after registering within the system. This would count as a critical security flaw within the system since customers should not have admin access to the application’s information. We used the following CTL statement:

AG(user_registered ⇒ admin_signed_in)

- **AG(user_registered:** When the customer has successfully registered as a user…
- **⇒ admin_signed_in):** they can sign in the system as an admin. We expected the result to be false.

The resulting analysis shows false which verifies the model represents a system that is not vulnerable to a faulty access control.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:39
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(user_registered ->admin_signed_in);
 is false
! (E [true U ! ((! (place (0) != 0 ) + place (18) != 0 ))]) -- false
time: 00:00:00:003


Results:
```

Figure 28: Verifying AG(user_registered ⇒ admin_signed_in)

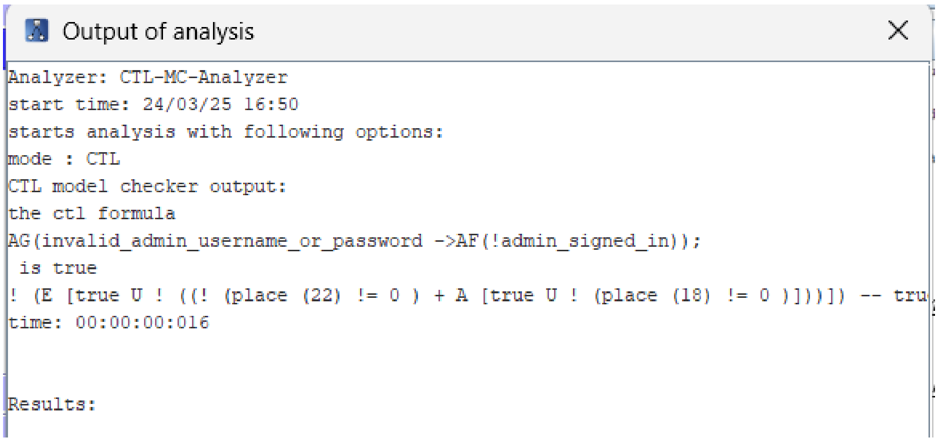
5.2.9. *Checking if someone can sign in as the admin with invalid credentials*

The system should only sign in the admin after issuing a valid username or password. This helps verify the identity of the admin and prevent unauthorised users from accessing the admin pages. We used the following statement:

AG(invalid_admin_username_or_password ⇒ AF(!admin_signed_in))

- **AG(invalid_admin_username_or_password:** When the admin submits an invalid username or password…
- **AF(!admin_signed_in)):** the system should not sign them in as an admin. We expected this to be true.

The statement shows the model appropriately represents the authentication functionality expected in the web application.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 16:50
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG(invalid_admin_username_or_password ->AF(!admin_signed_in));
 is true
! (E [true U ! ((! (place (22) != 0 ) + A [true U ! (place (18) != 0 )]))]) -- tru
time: 00:00:00:016


Results:
```

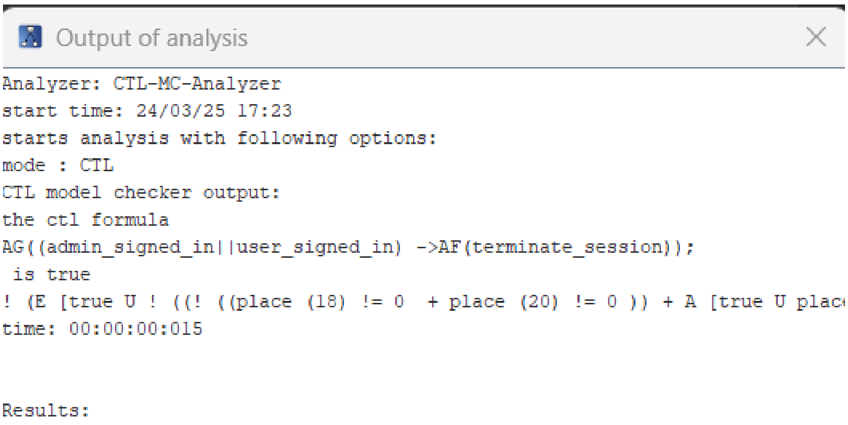
Figure 29: Verifying AG(invalid_admin_username_or_password ⇒ AF(!admin_signed_in))

5.2.10. *Check if user or admin clicking logout terminates their session*

Both the customer and admin should be able to sign out of their respective sessions in the system. This would ensure they no longer have access to their respective information. The statement used to verify this was:

AG((admin_signed_in||user_signed_in) ⇒ AF(terminate_session))

- **AG((admin_signed_in||user_signed_in):** When the system has an active session for either the admin or user…
- **AF(terminate_session)):** the system should allow them to eventually terminate their sessions. We expected the result to be true.



```
Analyzer: CTL-MC-Analyzer
start time: 24/03/25 17:23
starts analysis with following options:
mode : CTL
CTL model checker output:
the ctl formula
AG((admin_signed_in||user_signed_in) ->AF(terminate_session));
 is true
! (E [true U ! ((! ((place (18) != 0  + place (20) != 0 )) + A [true U plac
time: 00:00:00:015


Results:
```

Figure 30: Verifying session termination of the system

Performing analysis showed that the model captures the system’s ability or session termination.

References

Bowen, J.P., and Hinchey, M.G., 1995. *Ten Commandments of Formal Methods*

.

Hanamsagar, A., Woo, S.S., Kanich, C. and Mirkovic, J., 2016. *How Users Choose and Reuse Passwords.*

OWASP, 2010. *OWASP Secure Coding Practices Quick Reference Guide.*

OWASP, *Secure Product Design Cheat Sheet* [online].Available at: https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.html.

Woodcock, J., Larsen, P.G., Bicarregui, J. and Fitzgerald, J., 2009. Formal methods. *ACM Computing Surveys*, 41 (4), 1.

Appendix

User and admin credentials

Email / username	**Password**
john@gmail.com	jWcKx7AJ@DDcp9o^!#o$c
Alex@gmail.com	SMyZrdKCbDz8Xb#i#!*R2
admin	ebd^*iiu#MB8M*X