

**Pimpri Chinchwad Education Trust’s**

**Pimpri Chinchwad College of Engineering**

**Department of Computer Engineering**

# Mini Project 1 Report

## On

PARKING MANAGEMENT SYSTEM

## Group Members

Bhavansh Gupta BECOA134

Jitendra Joshi BECOA139

Ganesh Shinde BECOA164

Vaishali Avhale BECOA104

**Guide**

Prof. Manjiri Ranjanikar

INDEX

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Content** | **Page No.** |
| 1. | Introduction | 3 |
| 2. | Functional Requirements | 5 |
| 3. | Non - Functional Requirements | 5 |
| 4. | Design (Block Diagram) | 6 |
| 5. | Source Code / Functions | 8 |
| 6. | Output Screenshots | 9 |
| 7. | Test Plan Details | 15 |
| 8. | Test Scenarios/Test Cases | 18 |
| 9. | Junit Test Cases | 20 |
| 10. | Output Screenshots | 24 |

1. **INTRODUCTION**

**1.1 Motivation**

Parking management system for managing the records of the incoming and outgoing vehicles in an parking. It’s an easy for Admin to retrieve the data if the vehicle has been visited through number he can get that data .

Now days in many public places such as malls, multiplex system, hospitals, offices, market areas there is a crucial problem of vehicle parking. The vehicle parking area has many lanes/slots for car parking. So to park a vehicle one has to look for all the lanes. Moreover this involves a lot of manual labor and investment. Instead of vehicle caught in towing the vehicle can park on safe and security with low cost.

Parking control system has been generated in such a way that it is filled with many secure devices such as time and attendance machine, car counting system etc. These features are hereby very necessary nowadays to secure your car and also to evaluate the fee structure for every vehicle’s entry and exit.

The objective of this project is to build a Vehicle Parking management system that enables the time management and no parking disputes. The system that will track the entry and exit of cars, maintain a listing of cars within the parking lot, and determine if the parking lot is full or not. It will determine the cost of per vehicle according to their time consumption.

**1.2 Objectives**

We can park our vehicle in our own slot by paying.

* Because of that there are no towing problems.
* And our vehicle has been parked as a secure condition.
* There is no risk for vehicle owner for parking the car.
* In case of any damages and problem of vehicle that will claim by parking management.
* Maintain records in short time of period.
* Determines the parking area is full or not.
* Enhances the visitor’s experience.
* Generate Total Revenue Report

**1.3 Scope:**

* In the modern age. Many people have vehicles. Vehicle is now a basic need. Every place is under the process of urbanization. There are many corporate offices and shopping centers etc. There are many recreational places where people used to go for refreshment. So, all these places need a parking space where people can park their vehicles safely and easily. Every parking area needs a system that records the detail of vehicles to give the facility. These systems might be computerized or non-computerized. With the help of computerized system we can deliver a good service to customer who wants to park their vehicle into the any organization’s premises.
* Vehicle parking management system is an automatic system which delivers data processing in very high speed in systematic manner. Parking is a growing need of the time. Development of this system is very useful in this area of field. We can sell this system to any organization. By using our system they can maintain records very easily. Our system covers the every area of parking management. In coming future there will be excessive need of Vehicle parking management system.

1. **FUNCTIONAL REQUIREMENTS**

This is how the system shall look like and do when successfully completed. The system shall meet the following functional requirements:

* Admin need to enter all details for registration.
* Admin need to insert all details about customer and vehicle.
* Admin need to save all the details of customer and vehicle.
* Admin can retrieve the details of customer.
* Admin must generate a report of revenue collected.
* Admin can check for all parking spots
* Admin can delete remove vehicles after time expires from the record in a single click
* Admin can check if a specific parking spot is free or not.

1. **NON-FUNCTIONAL REQUIREMENTS**

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Some of the non-functional requirements include:

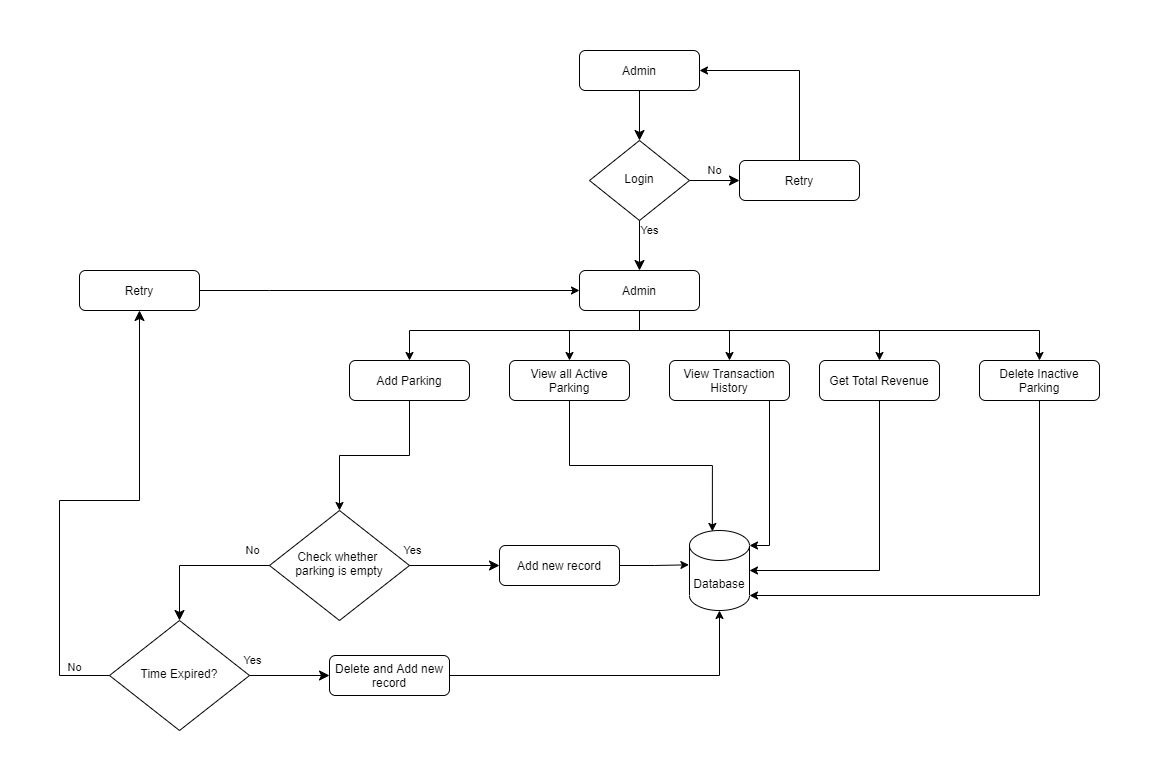
* Performance should be fast.
* Software should be easy to use.
* Databases should be consistent and atomic.
* Errors should be handled properly.
* Software should pass all the tests during testing.
* These Software is capable to secure the data and easily retrieve the data
* This software provide user and authentication so that only the legitimate user are allowed to use the software

1. **DESIGN**

**4.1 BLOCK DIAGRAM:**

The system consists of three modules one is the administrator module to add vehicles to parking, calculate total revenue, and check all parking’s and transactions.

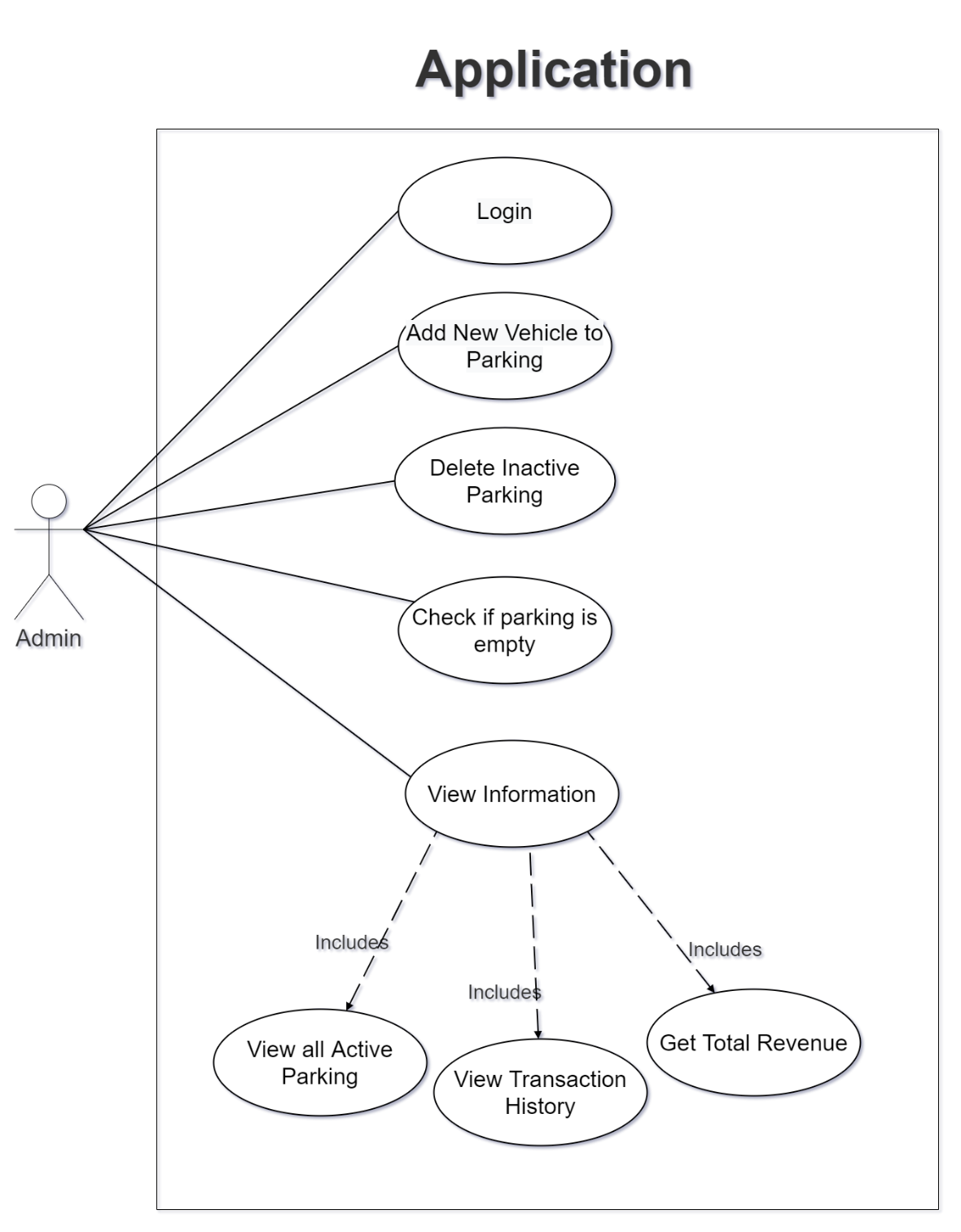
Following is the system architecture for the Online Blood Bank System.

****

**Admin Module:**

* Admin can insert all details about customer and vehicle.
* Admin can generate report of revenue collected.
* Admin can check for all parking spots.
* Admin can delete remove vehicles after time expires from the record in a single click
* Admin can check if a specific parking spot is free or not.

**4.2 USE CASE DIAGRAM:**



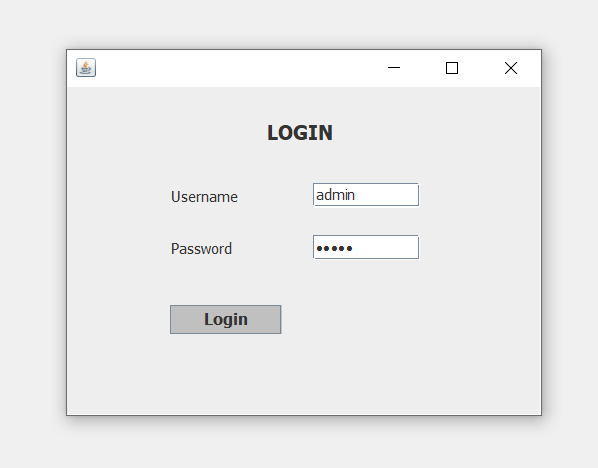
1. **SOURCE CODE / FUNCTIONS**

The source code can be accessed on GitHub at this link -

[bhavansh/Parking-Management-System (github.com)](https://github.com/bhavansh/Parking-Management-System)

(<https://github.com/bhavansh/Parking-Management-System>)

1. **OUTPUT SCREENSHOTS**

****

**Graphical user interface, application

Description automatically generated**

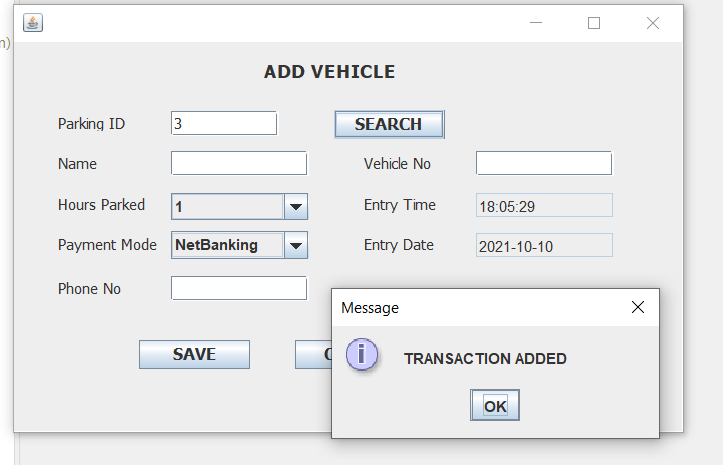
**Graphical user interface, application, Word

Description automatically generated**

**Graphical user interface, application, Word

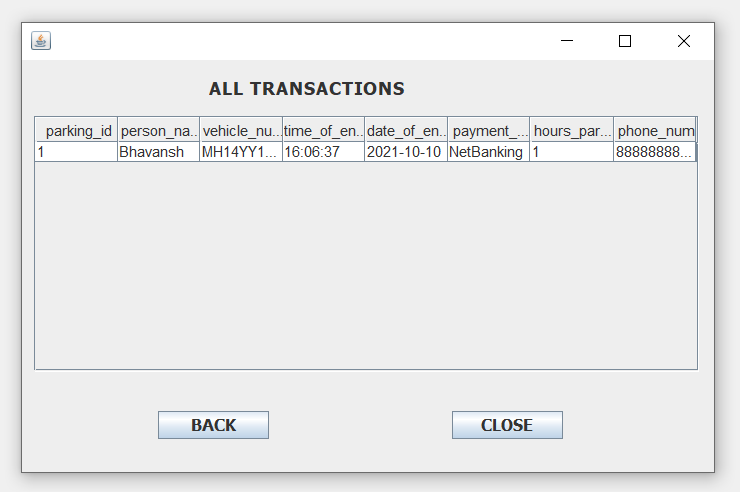
Description automatically generated**

**Graphical user interface

Description automatically generated**

**Graphical user interface, table

Description automatically generated**

****

**Graphical user interface

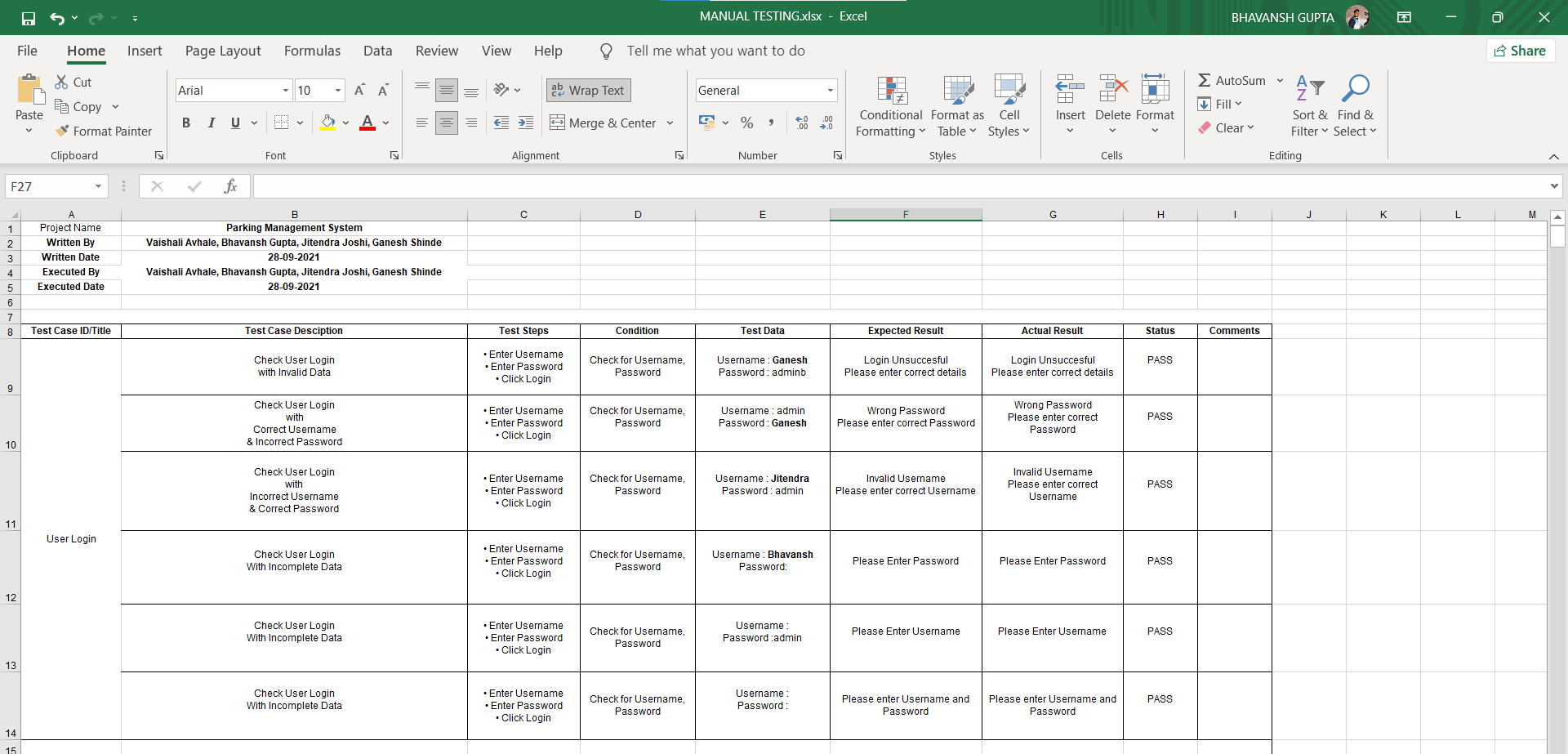
Description automatically generatedGraphical user interface

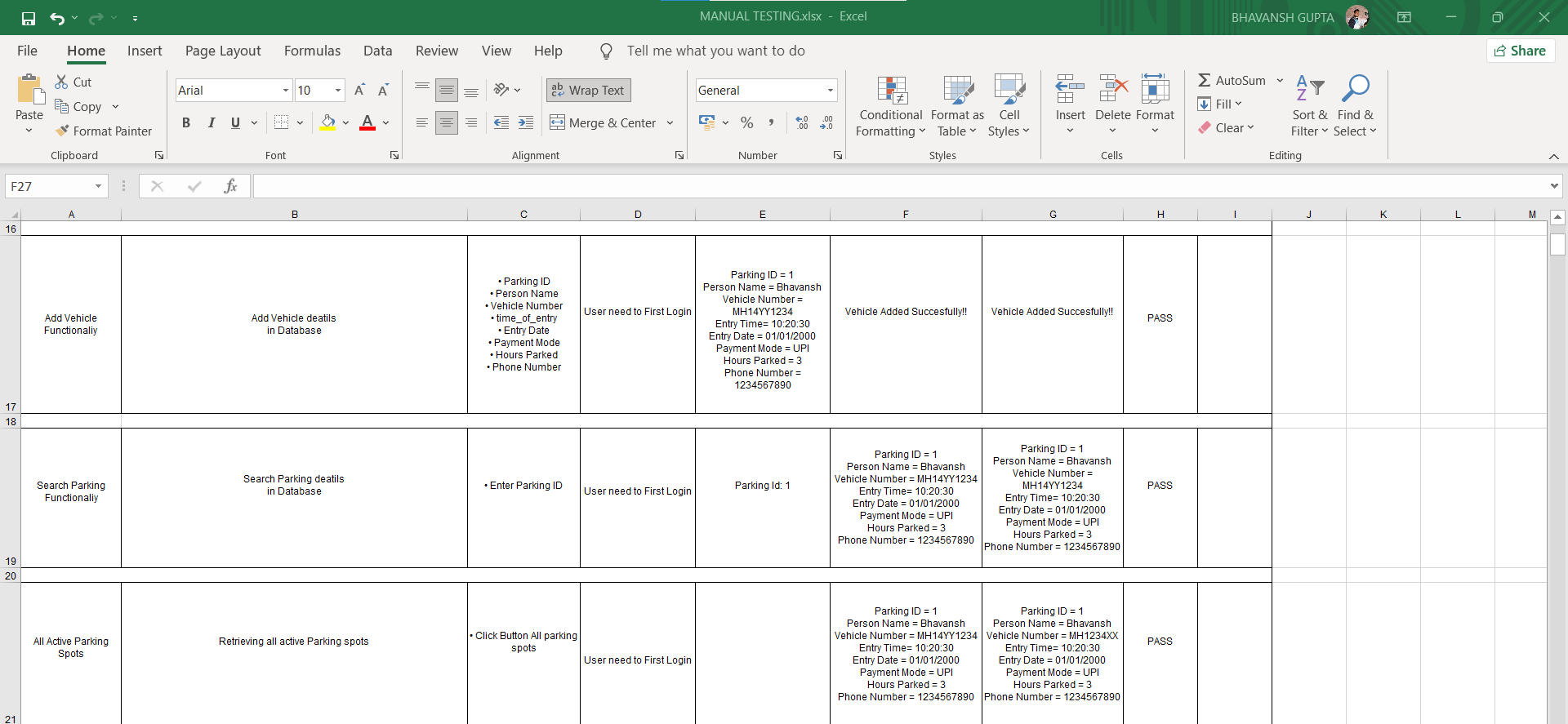
Description automatically generated**

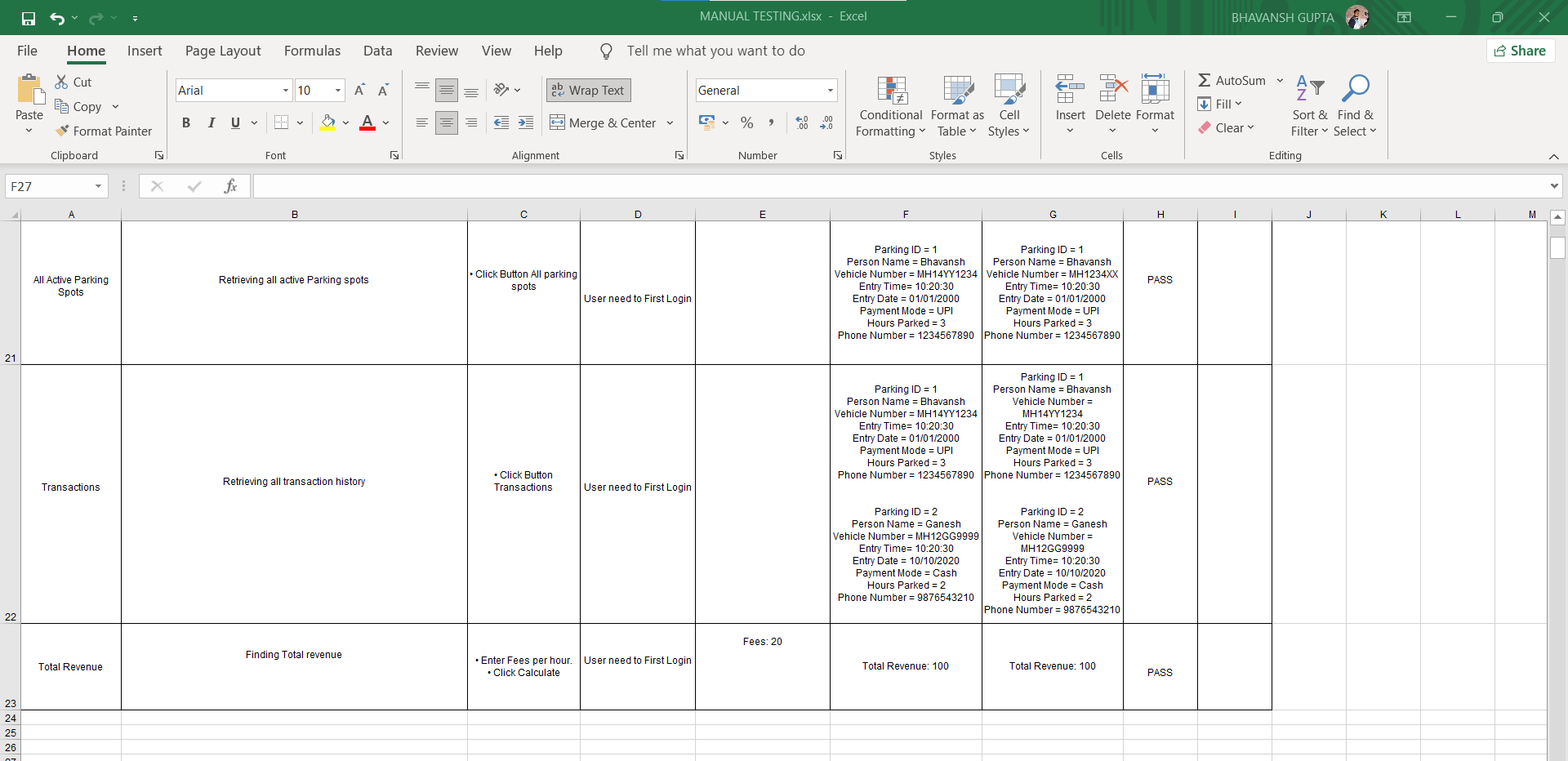
**Graphical user interface, application

Description automatically generated**

1. **TEST PLAN DETAILS**

****

****

****

Check Out the Excel sheet for test plan - [EXCEL SHEET](https://docs.google.com/spreadsheets/d/1qhcVGY10TX2uBj3XTi7c0wexfk6j12Z5/edit?usp=sharing&ouid=110755924810613437209&rtpof=true&sd=true)

(<https://docs.google.com/spreadsheets/d/1qhcVGY10TX2uBj3XTi7c0wexfk6j12Z5/edit?usp=sharing&ouid=110755924810613437209&rtpof=true&sd=true>)

1. **TEST SCENARIOS/ TEST CASES**

**Login:**

1. To check whether the inserted username and password fields are not blank?
2. To check if both the fields are filled or not?
3. to check if any username and password is correct or not?
4. To check if the system notifies or not on wrong input details?
5. To check if any field remains empty or not corrected?

**Add Vehicle:**

1. To check if the data is in correct format or not?
2. To check if any of the fields are empty or not?
3. To check if the data is found or not?
4. To check if the data already exists?
5. To check if a validation message pops up on the completion of addition or not?

**Search and Delete Vehicle:**

1. To check if the parking spot is occupied ?
2. To check if hours allotted are finished or not ?
3. To check if the data is deleted or not?
4. To check if a validation message pops up on the completion of deletion or not?

**Active Parking Spots :**

1. To check if the parking spot is occupied ?
2. To check if all occupied parking spots are listed or not ?

**Transaction History:**

1. To check if the Transactions are added or not?
2. To check if post time expiry are the parking spots freed and data is added to transactions or not?

**Total Revenue:**

1. To check if all the total hour count of active parking spots and transactions included or not?
2. To check if the input for fees per hour is working ?
3. To check if a message pops up with the total revenue calculated?

**Delete all inactive parking spots:**

1. To check if all parking spots whose time is expired are freed or not?
2. **JUNIT TEST CASES**
3. **AddVehicleTest.java**

package test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import parking.AddVehicle;

public class AddVehicleTest {

    @Test

    public void testCorrect() {

        AddVehicle obj9 = new AddVehicle();

        assertEquals(false, obj9.AddVehicles("","","","","","","",""));

    }

    @Test

    public void testCorrect2() {

        AddVehicle obj9 = new AddVehicle();

        assertEquals(true, obj9.AddVehicles("4","Bhavansh","MH14CC1234"," 12:34:50","2021-10-10","UPI","2","8888888888"));

    }

    @Test

    public void testCorrect3() {

        AddVehicle obj9 = new AddVehicle();

        assertEquals(true, obj9.AddVehicles("5","Jitu","MH14CC1234"," 12:34:50","2021-10-10","UPI","2","8888888888"));

    }

    @Test

    public void testCorrect4() {

        AddVehicle obj9 = new AddVehicle();

        assertEquals(true, obj9.AddVehicles("6","Ganesh","MH14CC1234"," 12:34:50","2021-10-10","UPI","2","8888888888"));

    }

}

1. **DeleteInactiveParkingTest.java**

package test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import parking.HomePage;

public class DeleteInactiveParkingTest {

    @Test

    public void test() {

        HomePage obj9 = new HomePage();

        assertEquals(true, obj9.DeleteInactiveParking());

    }

}

1. **LoginPageTest.java**

package test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import parking.LoginPage;

public class LoginPageTest {

    @Before

    public void setUp() throws Exception {

    }

    @After

    public void tearDown() throws Exception {

    }

    @Test

    public void LoginTest()

    {

        LoginPage obj = new LoginPage();

        assertTrue(obj.loginactivity("admin", "admin"));

    }

    @Test

    public void LoginFail()

    {

        LoginPage obj = new LoginPage();

        assertFalse(obj.loginactivity("", "admin"));

    }

    @Test

    public void LoginFail\_2()

    {

        LoginPage  obj = new LoginPage();

        assertFalse(obj.loginactivity("admin", ""));

    }

    @Test

    public void LoginFail\_3()

    {

        LoginPage obj = new LoginPage();

        assertFalse(obj.loginactivity("", ""));

    }

    @Test

    public void LoginFail\_4()

    {

        LoginPage  obj = new LoginPage();

        assertFalse(obj.loginactivity("Jitendra", "Joshi"));

    }

}

1. **TotalRevenueTest:**

package test;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import parking.HomePage;

import parking.TotalRevenue;

public class TotalRevenueTest {

    @Before

    public void setUp() throws Exception {

    }

    @After

    public void tearDown() throws Exception {

    }

    @Test

    public void testCorrect() {

        TotalRevenue obj9 = new TotalRevenue();

        assertEquals(140, obj9.getTotalRevenue(20));

    }

    @Test

    public void testCorrect1() {

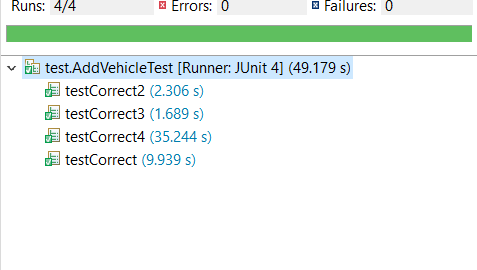
        TotalRevenue obj9 = new TotalRevenue();

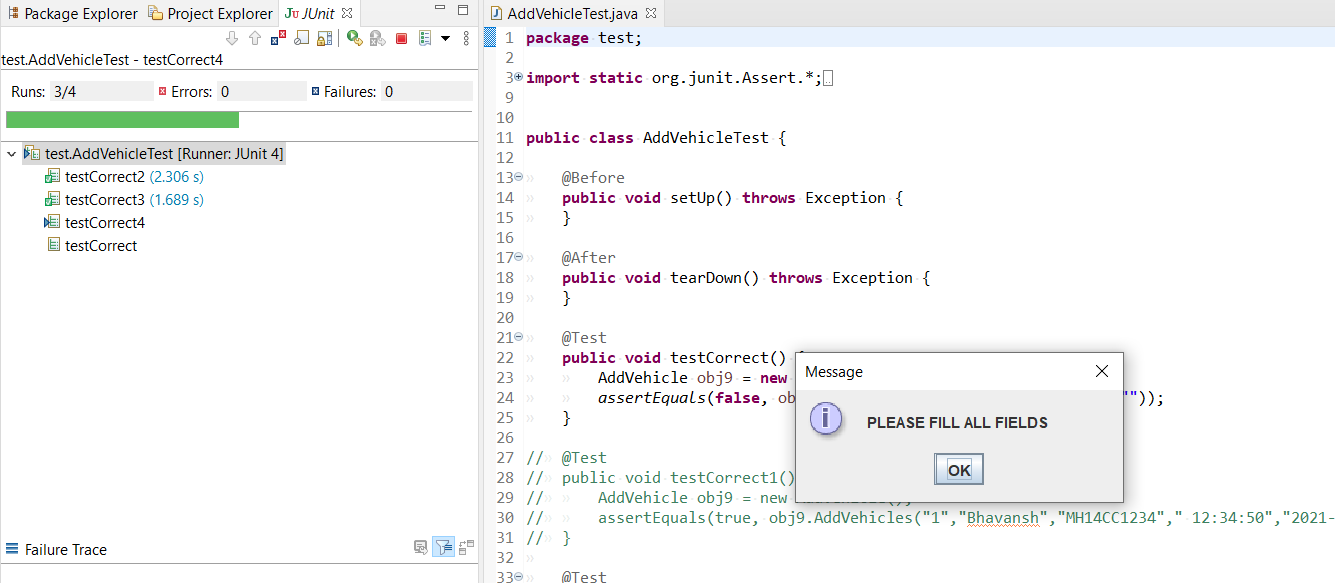
        assertEquals(133, obj9.getTotalRevenue(19));

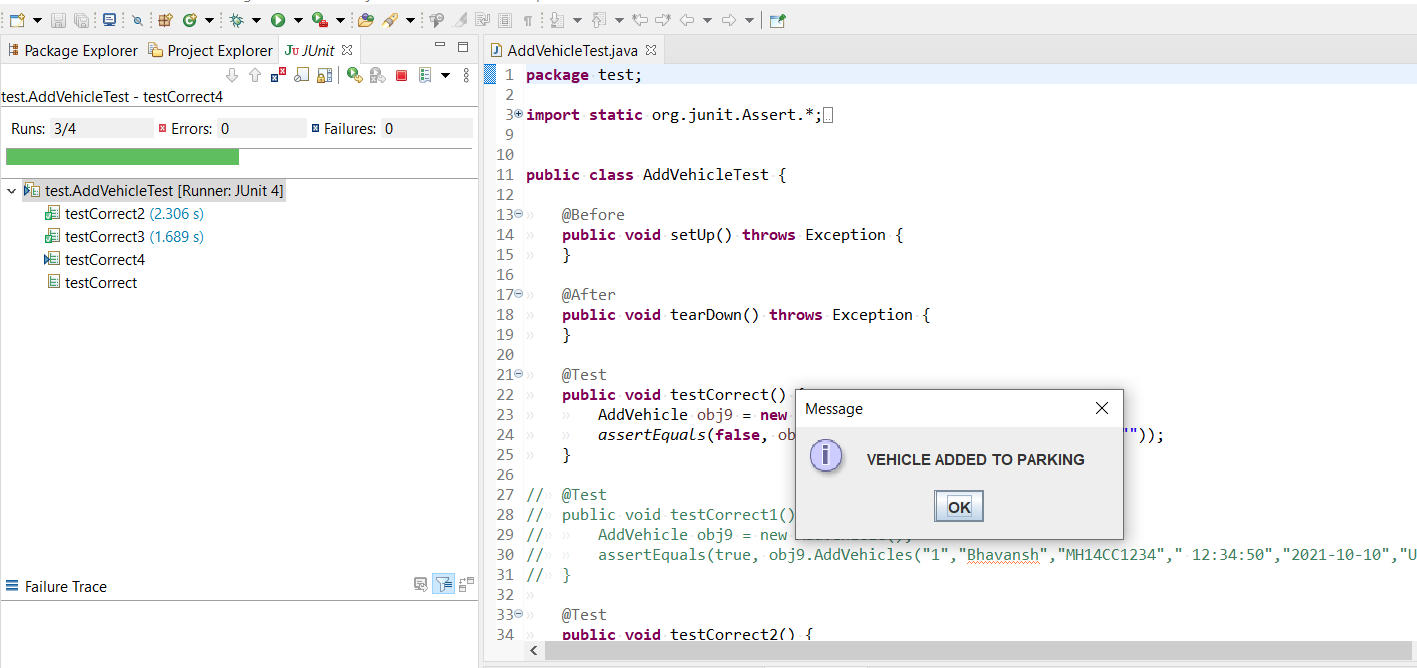
    }

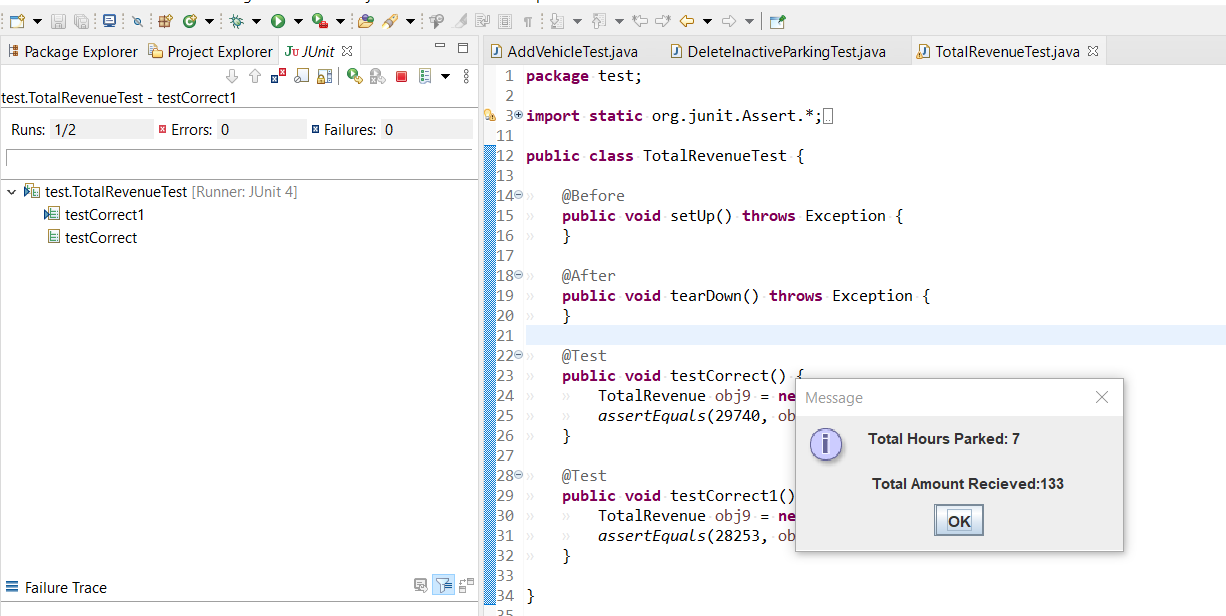
}

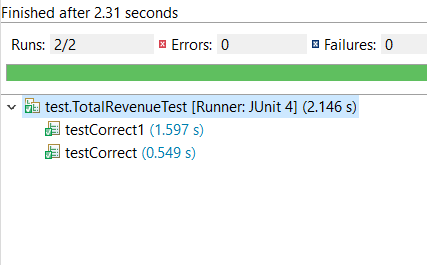
1. **OUPUT SCREENSHOTS**

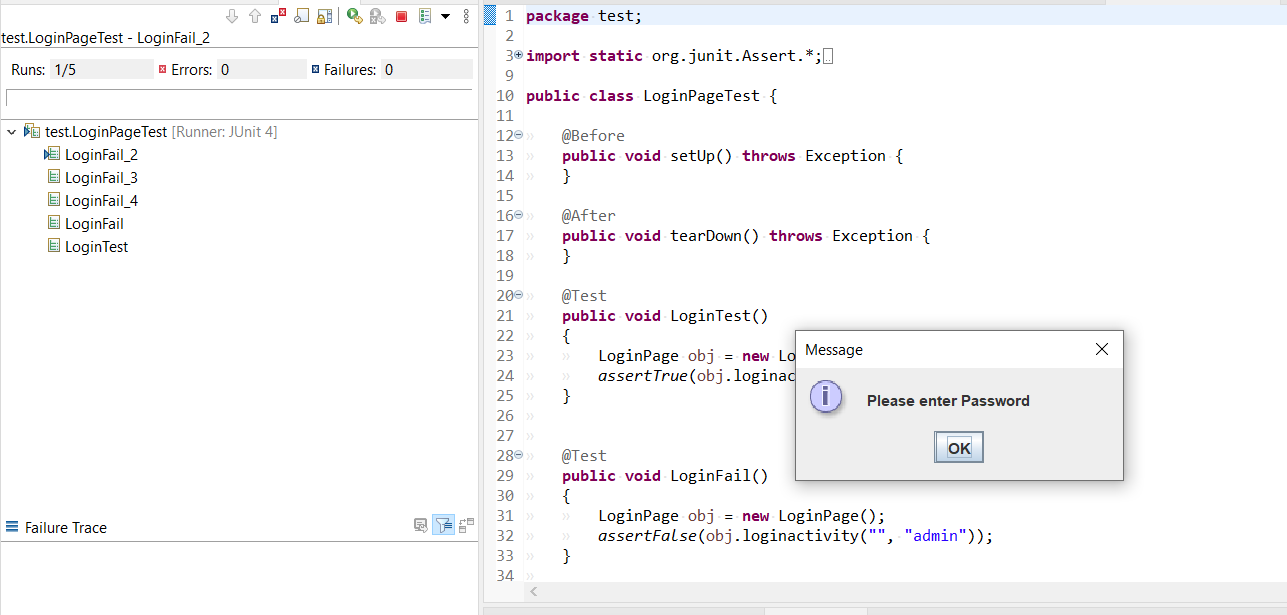
****

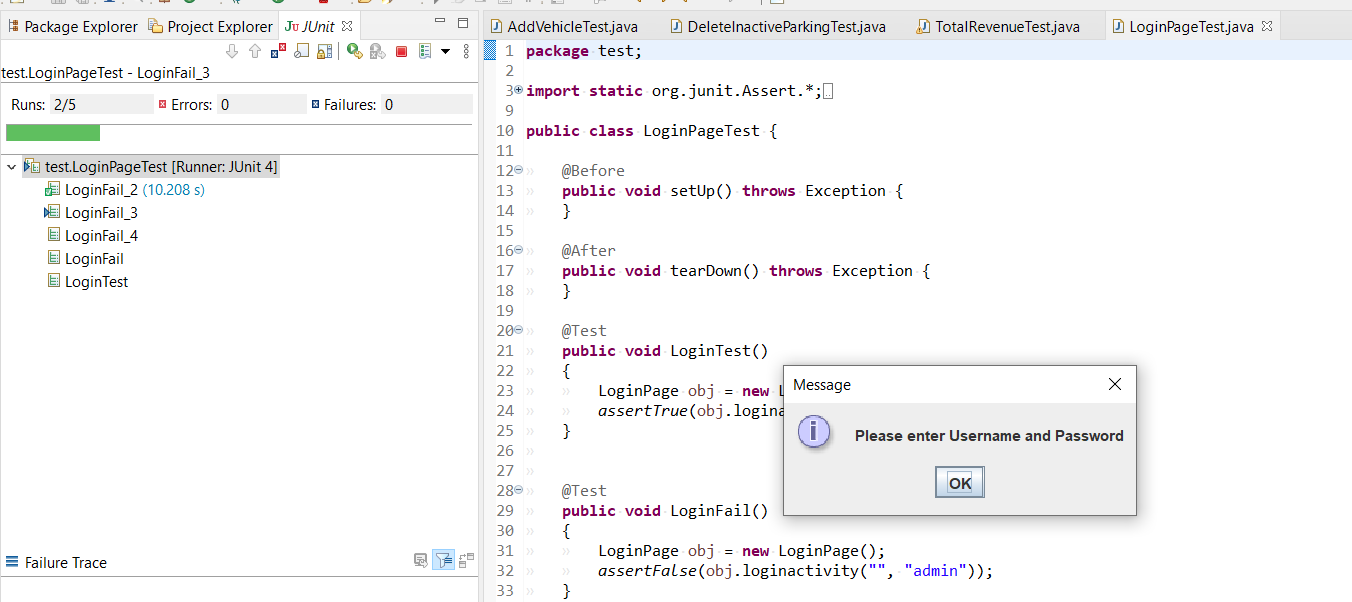
****

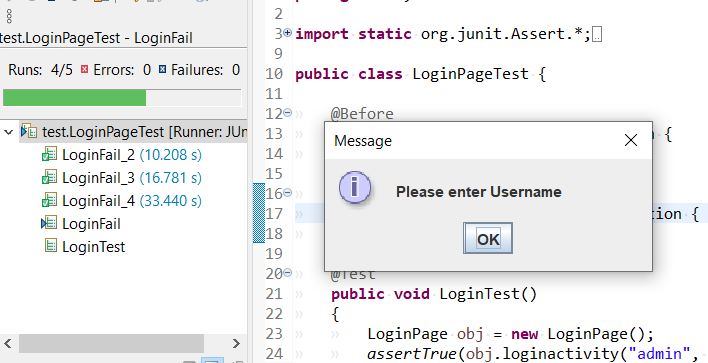
****

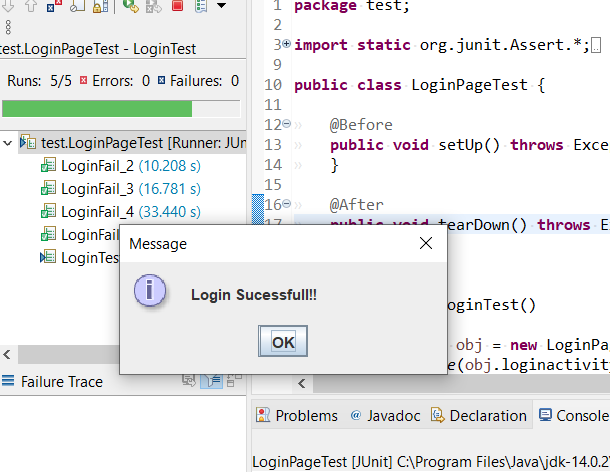
****

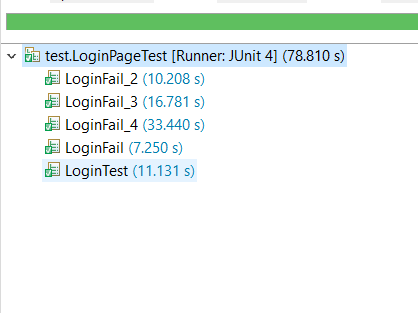
****

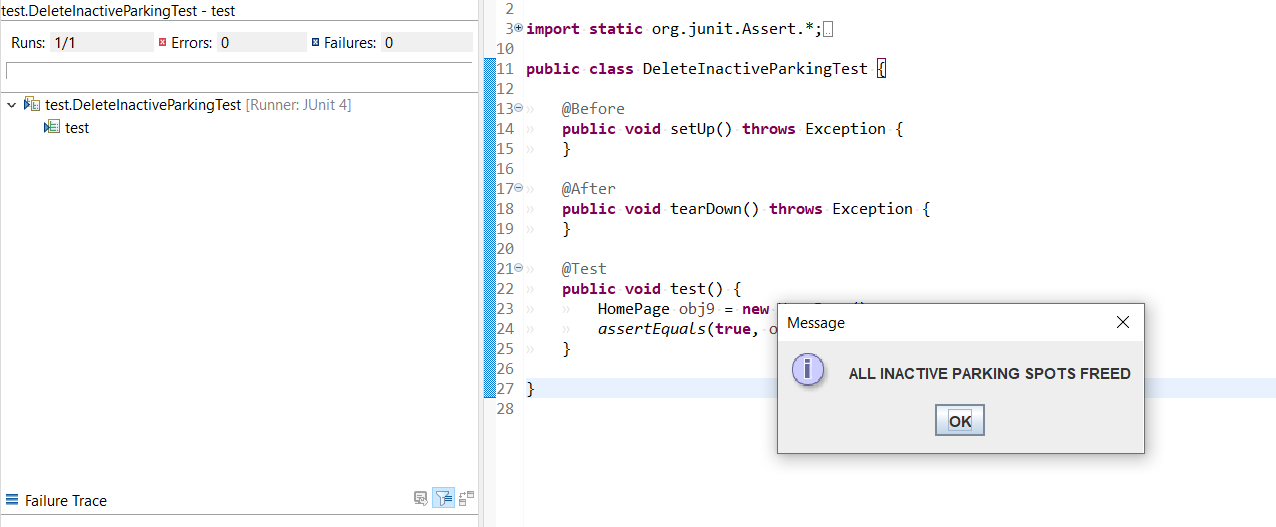
****

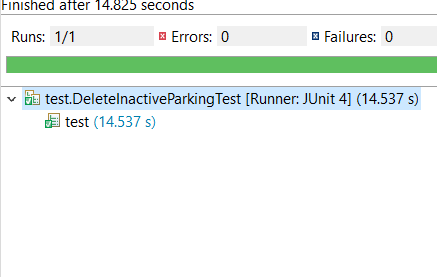
****

****

****

****

****

****