Phase I

GitHub Submission Template:

Group Division

Team Information

1.Andeta Mushi

2.Cristina Cala

3.Ina Zenelaj

4.Anxhela Koka

5.Erika Mejdi

Team Leader:

1.Andeta Mushi

GitHub: andetam

Team Members:

2. Cristina Cala - GitHub: cristinacala

3. Ina Zenelaj - GitHub: zenelajina

4. Anxhela Koka - GitHub: AnxhelaKoka

5. Erika Mejdi - GitHub: erikamejdi

Project Details

Project Title: Car Rental System

Problem Statement:

Every day more and more there is a need to be on the move and the way to make the movement even simpler is to use the car. Regardless of other alternative ways, the use of a personal car offers comfort, flexibility and above all you have the opportunity to move based on your needs,

adapting to your schedules. Also, we notice that recently the number of tourists has increased in our country, and the majority of them are obviously looking for a safe trip, suitable to the schedules theirs and above all at a low cost, all the more so that most of them choose to visit Albania during the summer, when the movement becomes even more difficult by bus or even by car. by taxi, where we often hear cases where most of the companies, due to the influx, do not even have the opportunity to respond and provide the service, therefore the best way would be to rent a car. In recent years, this service has been provided even more popular among Albanians and foreigners, but this service is still lacking in our country, this is because most of the businesses are not known because they do not have a proper marketing. Most of them operate through an Instagram page, where there is a facility they are incomplete, and they need improvement. Most of these pages only show images of cars, and not very clear information about the schedule, prices, and often customers feel in difficulty and simply choose to give up.

Solution Proposed:

To improve the clients way of traveling, we offer an application called Car Rental System which allows the users to make a safe journey, and also allows them to reserve the car that they want. With this innovative application, customers can choose the car they prefer in a quick and simple way.Also, in this application, we offer all the information that users needs about the car, information about the type of car, the rental price, the dates of free to reserve the car etc. We think that in this way the work of the business and also of the client is simplified.

Project Scope:

Aim:

The purpose of Car Rental System is to offer an innovative application in a convenient way to facilitate the process of choosing cars and to offer an excellent service to the users of this application.Through this application we will offer a great variety of cars to suit the preferences and demands of customers.

Main Objectives:

- To simplify the process
- Security
- Technological Innovation
- To provide transparency

Application Description:

Car Rental System is an innovative application that deals with car rental reservations. This application is designed based on the need of customers to have an easier and faster way to reserve a car rental. Car Rental System offers a solution to meet the needs of customers.

Functionality

1. User Friendly Interface - Car Rental System application offers a user-friendly interface for users, so that it is easy to see available cars and make reservations.

2. Wide inventory of cars - This application offers a wide range of cars, taking into account that users choose the car they prefer and that best suits their requirements.

3.Availability- This app provides real-time availability allowing users to check car availability and free schedules.

4. Confirmation - After the users have made the reservation, a notification will come that the reservation has been confirmed.

5.Feedback- Car Rental System application, it will include a rating system in order to ensure improvements and to share with others their experiences regarding our service.

Roles and Tasks Distribution

1. Andeta Mushi- Project Planning, App Development,Design

2. Cristina Cala - Project Planning,App Development,Design

3. Ina Zenelaj - Project Planning,App Development,Design

4. Anxhela Koka - Market research,Testing and Integration,Design

5. Erika Mejdi - Market research,Testing and Integration,Design

Deadline

Submission Deadline: 04.03.2024, 23:59 hours.

Phase II: User Requirements and Application Specifications

Submission Deadline: 18.03.2024, 23:59

1. Chosen Development Model:

Waterfall

Justification:

The systematic and sequential approach of waterfall development, which can be advantageous for projects with clearly specified and solid needs up front, is the reason why the automobile rental application was developed using this methodology. Within the automobile rental sector, where specifications might not alter regularly and a thorough plan can be created beforehand, Waterfall offers an organised approach to moving through several project stages.

Requirements Gathering:

In the Waterfall approach, compiling and recording all of the prerequisites for the vehicle rental application is the first step. Understanding user requirements, desirable features, and corporate goals are all part of this. During this stage, stakeholders including administrators, clients, and end users offer their opinions.

Design:

The process starts as soon as the criteria are decided upon. This entails drafting comprehensive requirements for the database design, user interface, and system architecture. The application's functionality and layout are visualised through the creation of wireframes and mockups.

Implementation:

After receiving the design specifications, the development team begins writing the programme in accordance with the specified specifications. The entire vehicle rental application is made up of all the system's built-in components.

Testing:

Extensive testing is carried out following the implementation phase to guarantee that the application satisfies the requirements and operates as intended. To find and address any faults or problems, this comprises unit, integration, and system testing. *Deployment:*

The application is put into production environments after testing is finished and it is judged ready for public release. This include configuring databases, setting up servers, and granting end users access to the application.

Maintenance:

When an application is deployed, it goes into a maintenance phase where any updates, patches, or repairs that are required are performed. This guarantees that the application will function efficiently and without interruption over time.

2. User Requirements:

a. Stakeholders:

End users: These are people or businesses who will use the programme to rent cars. Their top priorities include a user-friendly interface for simple reservations, open pricing, and consistent car availability.

Clients: The clients could be companies that own a fleet of cars or rent-a-car agencies. A strong platform that effectively handles reservations, car inventory, payments, and customer service is what they are looking for.

Developers: They are in charge of the application's conception, execution, and upkeep. Their responsibility is to meet the functional requirements specified by stakeholders while ensuring the application's technological stability, scalability, and security.

Administrators: In charge of maintaining user accounts, vehicle listings, price configurations, and handling any operational problems, administrators are in charge of the application's backend. Their main concern is being able to have complete control and monitoring capabilities.

b. User Stories:

User Type: End-user

User Story: As an end-user, I want to be able to quickly look for automobiles that are available to rent based on my location and rental preferences.

Requirement: Users should be able to input their location, preferred rental dates, car type (e.g., sedan, SUV), and any extra amenities (e.g., GPS, child seat) into the application's search tool.

Benefit: By enabling consumers to locate appropriate rental options fast, this feature improves their entire experience and cuts down on the amount of time spent exploring.

User Type: Administrator

User Story: As an administrator, I want to be notified when a rental is past due so I may take the necessary action.

Requirement: When a rental goes past the agreed-upon return time, the programme must to automatically notify administrators.

Benefit: This guarantees prompt handling of overdue returns, enabling administrators to contact users, impose penalties if necessary, and keep the fleet available for future clients.

User Type: Client (Car Rental Company)

User Story: As a car rental company, I want to customize pricingaccording on variables like the kind of vehicle, length of the rental, and demand during certain seasons.

Requirement:The application must enable dynamic pricing algorithms, which modify rental prices in response to current market trends, car availability, and demand.

Benefit: By enabling dynamic pricing adjustments, this feature helps automobile rental

companies maximise revenue and maintain market competitiveness by maximising profitability.

3. Functional Requirements:

a. Brief Description:

● User Registration: To access the application, users must first create accounts. ● Car Search: Allow consumers to look up available cars by location, date, and preferences ● Booking: Permit users to reserve a particular car for a given date or time. ● Payment Processing: Enable safe financial transactions for rental car reservations. ● Admin Dashboard:Give managers a single interface to control people, vehicles, reservations, and money.

b. Acceptance Criteria:

● User Registration:

Registration of Users: Users may complete a registration form.
User gets an email asking for verification.
If the user has the given credentials, they can log in.

● Car Search:

The search bar allows users to enter their location, date, and preferences.
The available cars that meet the requirements are listed by the system.
Each car's details, including availability and price, are viewable by the user.

● Booking:

Along with choosing an automobile, the user can set the booking dates and times. The user receives an email confirmation from the system after the booking is confirmed. The reserved vehicle is not available on the chosen dates.

● Payment Processing:

The user can input payment information safely.
The payment is processed by the system, and the booking status is updated.
The user gets a confirmation of their payment.

● Admin Dashboard:

Admin credentials allow for a unique login.
A dashboard shows an overview of data on users, vehicles, reservations, and payments. The administrator has the ability to modify vehicle listings, add and remove users, and access payment information.

4. Non-Functional Requirements:

a. Brief Description:

Performance: The system should be able to manage several requests with ease and react to user actions fast.

Usability: Users should need little training to operate the interface because it should be simple to understand and navigate.

Security: To prevent unwanted access and cyberattacks, the system must protect user data and payment details.
Reliability: To provide a dependable user experience, the system should function consistently and without frequent faults or outages.

Scalability: As the application expands, the system should be able to manage a rise in the volume of data and user traffic.

b. Acceptance Criteria:

-Performance: Search requests should respond in less than two seconds.

At least 1000 people should be able to connect concurrently without experiencing any noticeable lag.

-Usability: It should take fewer than five minutes for users to finish the booking procedure.

Logically arranged and labelled navigation menus facilitate effortless access to many functionality.

-Security: Industry-standard hashing techniques should be used to encrypt user credentials.

To ensure encryption during transmission, payment information should be sent via the HTTPS protocol.

-Reliability: To enable for planned maintenance, the system should have at least 99.9% monthly uptime.

Less than 0.1% of total transactions should have an error rate.

-Scalability: Within six months, the system should be able to handle a 50% increase in user traffic without seeing a discernible drop in performance.

Within the following year, the volume of data should double, and database storage should be scalable to accommodate this.

5. Application Specifications:

a. Architecture:

For the rental car application, a microservices architecture was selected.
Overview: Several separate microservices make up the system, and each is in charge of handling a particular set of tasks like booking, user management, processing payments, and inventory management. Through APIs, these microservices can interact with one another.

Components:

- User Service
- Booking Service
- Payment Service
- Inventory Service
- Notification Service

Interactions: RESTful APIs allow microservices to communicate asynchronously with one another. For instance, when a user submits a request for a reservation, the Reservation Service gets in touch with the Inventory Service to see if a car is available and the Payment Service to handle the payment.

b. Database Model:

Tables:

- Users
- Cars
- Bookings
- Payments
- Notifications

Relationships:

Users have a one-to-many relationship with Bookings.

Cars have a many-to-many relationship with Bookings.

Bookings have a one-to-one relationship with Payments.

Notifications have a one-to-many relationship with Users.

Constraints:

Foreign key constraints to ensure referential integrity.

Unique constraints on email addresses for Users.

Constraints on payment amounts for Payments.
c. Technologies Used:

- Backend: Node.js with Express.js: This quick and scalable backend was created to effectively manage several requests at once using Node.js and Express.js. ● Database:

MongoDB: Adaptable NoSQL database that effectively stores dynamic data, perfect for handling user profiles, reservations, and car information. ● Frontend: React.js is a component-based user interface package that provides a declarative method for creating dynamic, interactive user interfaces.

- API Gateway: NGINX: Lightweight, high-performance gateway for routing and load balancing, enhancing application performance and reliability.
- Message Queue: RabbitMQ: Dependable message broker that promotes asynchronous microservices communication while guaranteeing fault tolerance and scalability. ● Containerisation: Docker: Portable containers make microservices easier to deploy and maintain while guaranteeing dependability and consistency in a variety of settings. ● Orchestration: Kubernetes is a scalable orchestration platform that optimises resource usage, guarantees high availability, and automates the deployment and scaling of containerised applications.

d. User Interface Design:

Mockups and wireframes (Offer links to mockups or wireframes, or explain the UI design). The user interface's straightforward and user-friendly design makes it simple for customers to manage their accounts, search for cars, and make reservations.

e. Security Measures:

- Encryption: To prevent unwanted access, user passwords are hashed with bcrypt before being stored in the database.
- Authentication: To provide a safe method for managing sessions, JWTs, or JSON Web Tokens, are utilised for user authentication.
- Authorization: To limit access to specific capabilities depending on user roles (e.g., admin, regular user), role-based access control, or RBAC, is established. ● HTTPS: To secure data transfer over the network, all client-server communication is encrypted using the HTTPS protocol.
- Input validation: To guard against injection attacks and guarantee data integrity, input validation is applied on both the client and server sides.

Phase III: Software Design and Modeling

Deadline: April 1st, 2024, 23:59

Software Design and Modeling

Group Name: Car Rental System

Software Architecture

System Architecture:

User Interface Layer:

- **Web Interface:** Our project will feature an interface that users can interact via a web browser, connecting them to the system.The services offered to clients will include managing their reservation books and checking the features of different kinds of cars.Expect this; the user's account login details will also be provided.In order to make accessing the platform easier and faster, we will first introduce a web browser page and afterwards plan to develop an application.

Application Layer :

Verification and Permission:

- User authentication: Using credentials like a username and password, email verification, or social media login, the system confirms users' identities.
- Access control based on roles (RBAC): The levels of access to system functions and data vary depending on the job of the user (admin, and client). RBAC makes sure that users can only access the features they are allowed to use.
- Security Procedures: Use secure authentication methods to protect user credentials while logging in, such as OpenID .Use encryption methods to secure data transfer over networks, such as SSL/TLS.

Car Management:

- Inventory management: Keeps data for the model, year, mileage, condition, and location of all the rental cars as well as their availability.

- Pricing and Availability: Establishes rental prices according on demand, location, car type, and length of hire. Real-time updates are made to the availability status to account for reservations and refunds.
- Maintenance Scheduling: Keeps track of each vehicle's maintenance plans, which include routine servicing, repairs, and inspections to make sure the fleet is kept in top shape.

Booking Management:

- Reservation Process: Leads customers through the steps of looking for cars that are available, choosing rental options (such as dates and hours of the rental, and pickup and drop-off locations), and finalizing reservations.
- Availability checking: This method reserves a few chosen cars for the duration of the booking process in order to check the availability of cars based on the user's provided criteria and avoid duplicate booking.
- Reservation Lifecycle: Oversees all aspects of managing a reservation, including booking confirmation, modification (such as altering the vehicle type or rental dates), cancellation, and processing of refunds.
- Review :Also in this section after the user close the lifecycle the user can leave a feedback.

Notifications:

- Transactional Notifications: Provides consumers with immediate email or SMS notifications for important occasions, like rental reminders, confirmations of reservations, directions for pickup and drop-off, and payment receipts.
- Proactive Communication: Notifies users of any relevant modifications or disruptions (e.g., vehicle maintenance, location closure) and offers proactive updates on reservation status changes (e.g., availability updates, rental extensions).

Backend Layer:

Administrators can handle six important areas of the rental car system by gaining access to the admin view.

Client Management:

User Profiles: In the client management , administrators have access as well as control over detailed user profiles. This contains data including user contact information, rental

history, preferences, and any special requests or customer notes.

Registration and Authentication: Administrators are able to manage the new client registration process. They have the ability to check user data, and accept or decline requests for new accounts. They can also help clients with account access and resolve issues linked to authentication.

Car Management:

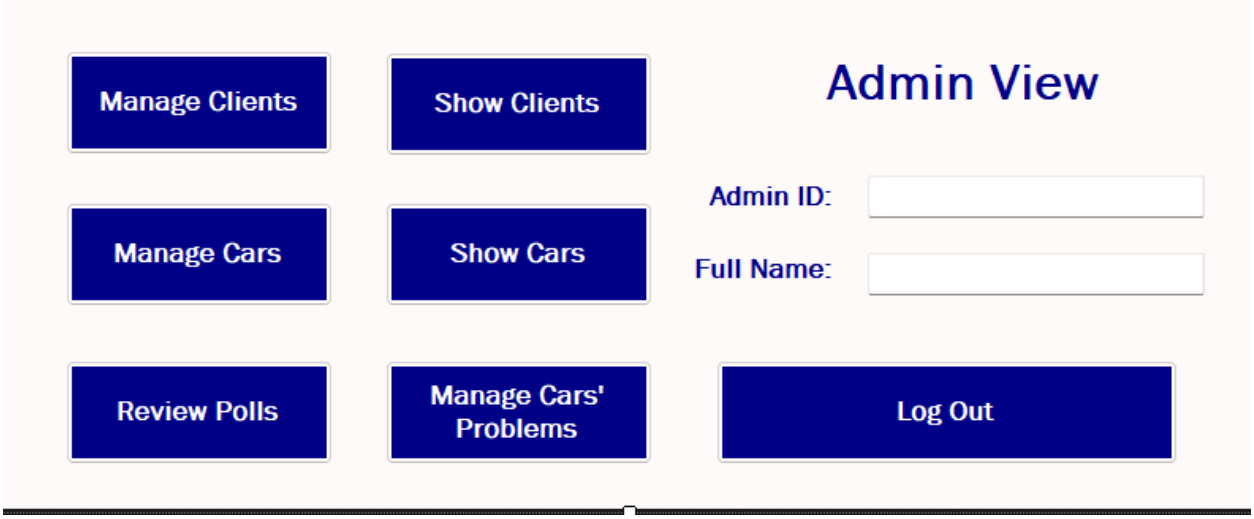
Inventory Overview: Adminstartors can see the details of all cars in the fleet,and also can make addition of new cars by checking the model ,the year etc.They also have the option of editing ,where they can edit details of cars including here pricing, availability, and descriptions.As for the maintenance tracking administrators can have access on schedule and maintenance activities.

Review Polls

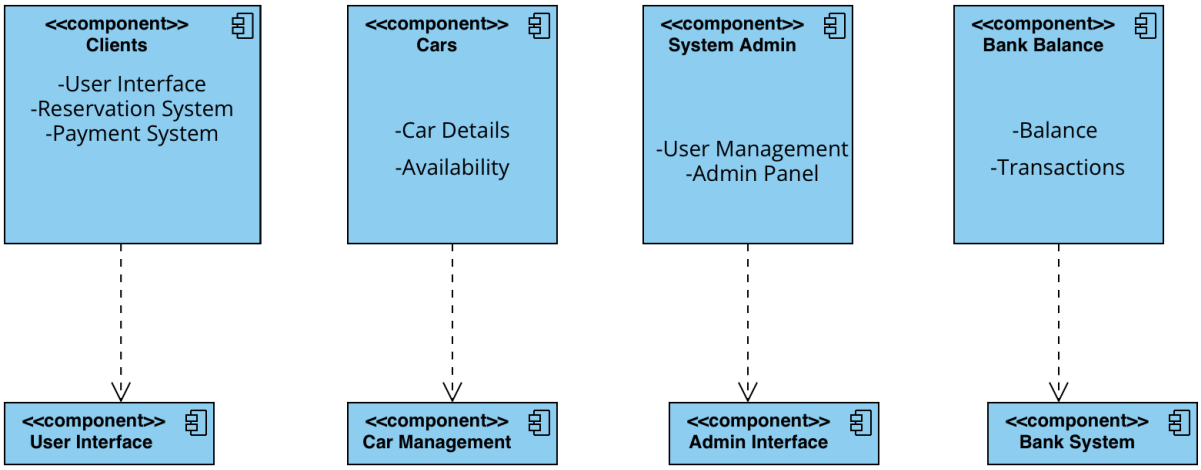
Review polls are made and maintained by administrators to get input on rental experiences. They adapt the questions, analyze the answers, and apply what they know for continued growth. Reports and poll data combine to provide a full picture of client feedback and service improvements.

Manage Car Problems:

Administrators check and handle reported vehicle difficulties, including damage or mechanical issues. They delegate maintenance chores, monitor the status of resolutions, and guarantee that rental cars are kept in top shape.



Component Diagram:



Clients Component - Represents the users of the application. This displays functionalities related to the user interface, booking and making payments.

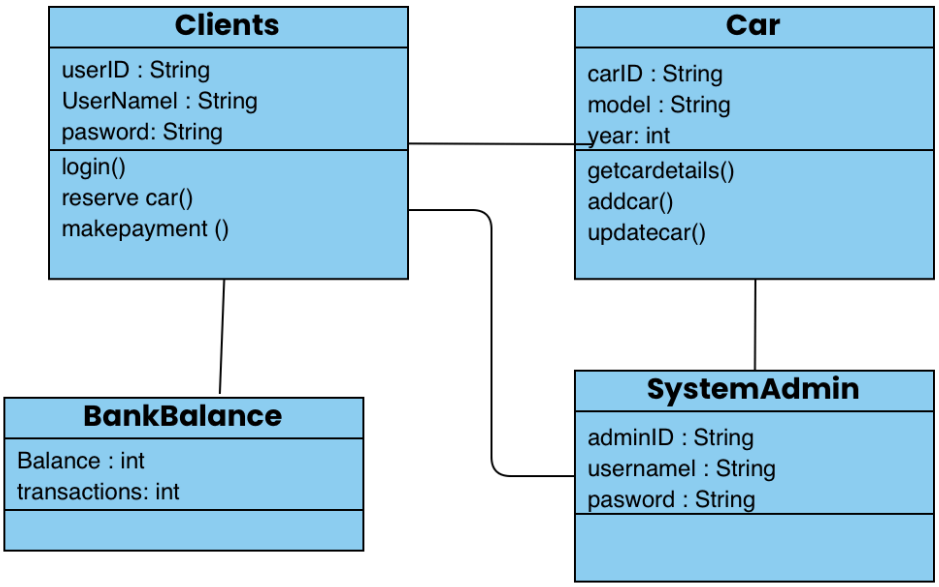
Cars Component - This manages the inventory of cars that are available. This includes data for each car such as, details, availability, etc.

The administrative component of the system - This is responsible for the functionality of managing users, generating reports, etc.

Bank Balance Component - This includes the financial aspects of the application, making payments secure and saving data.

Detailed Design

Class Diagram:



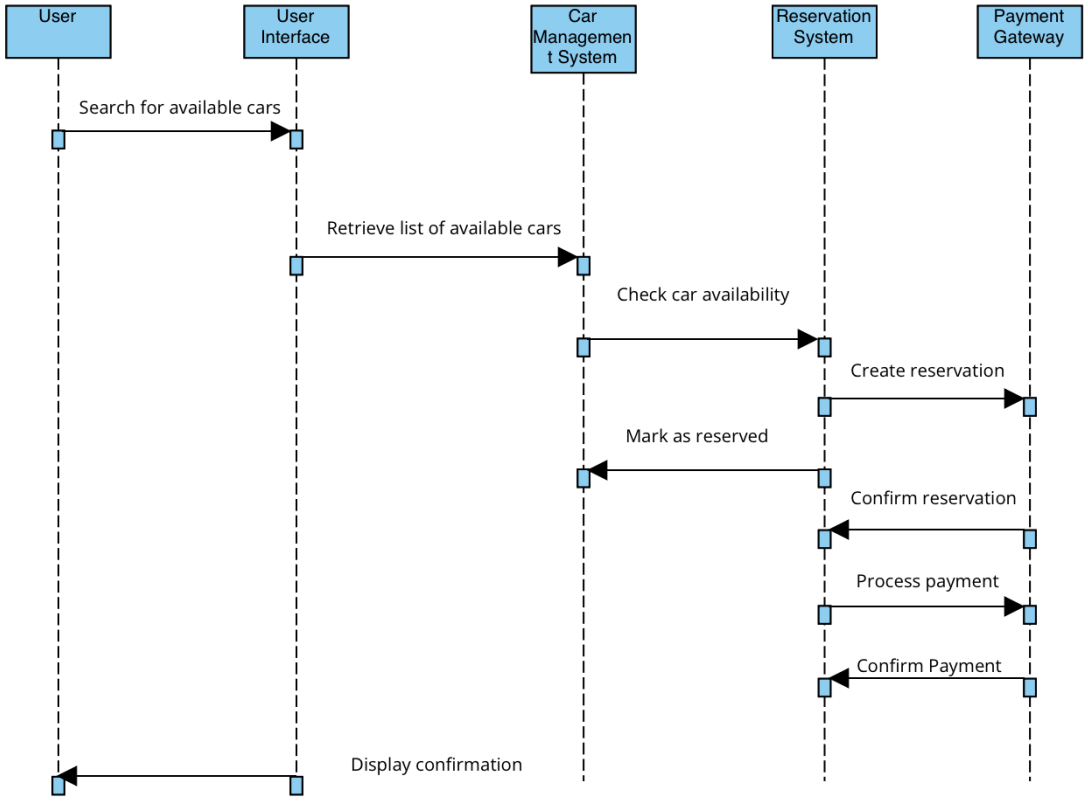
Client: Contains methods such as login(), logout(), , reserveCar(), makePayment(), for user interactions like logging in/out, , reserving/canceling reservations, making payments, and updating profile details.

Car: Contains methods displayDetails() and checkAvailability() to retrieve and display car information, as well as checking availability.

SystemAdmin: Contains methods authenticate() to verify admin credentials and generateReport() to create system reports, as well as methods for managing the car inventory: getCarDetails(), addCar(), updateCar(), and deleteCar().

BankBalance: Does not have methods depicted.

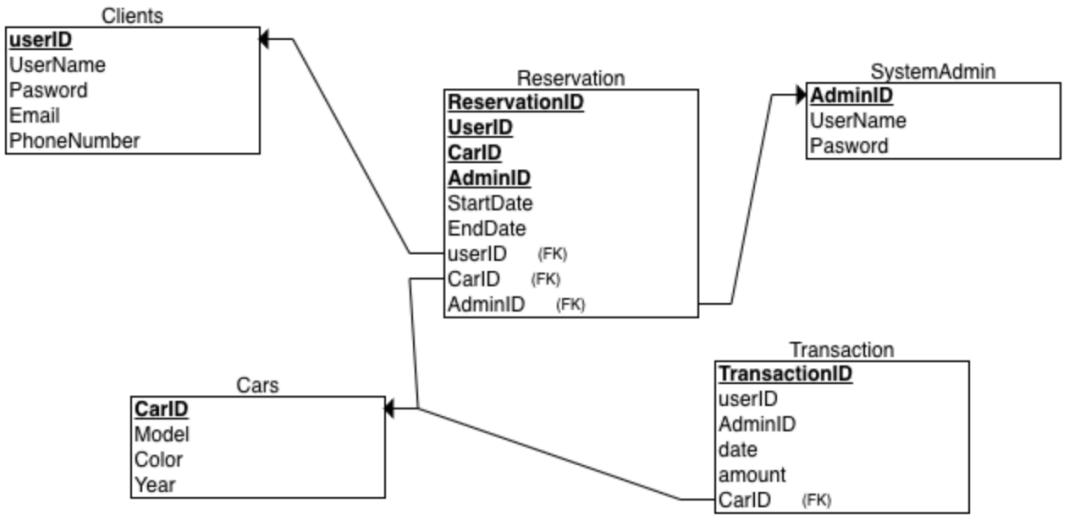
Sequence Diagrams:



The sequence diagram represents the interaction of the different components of the application. This diagram first starts with the user who searches for cars that are available through the user interface and finally we have the confirmation that is shown to the user after he has finished booking and paying for the car.

In this diagram, each step shows the communication between the user interface, the reservation system, the payment system, the car management system, this describes every action that happens during the car rental process.

Database Design:



Clients table- Store information about the clients that are using the application, this includes userID, username,pasword,email and phone number.

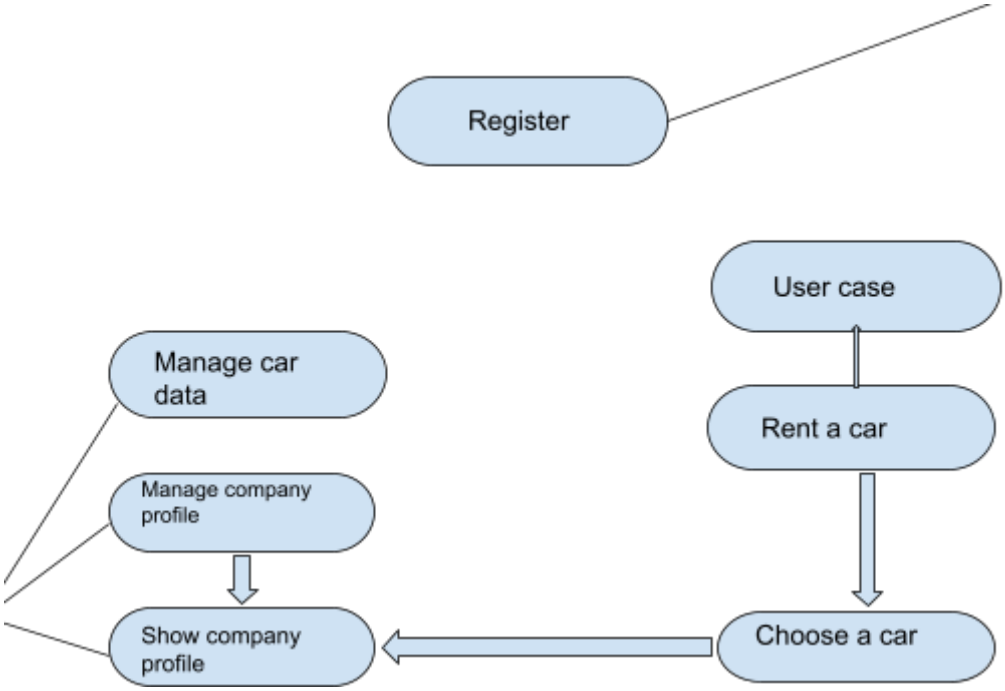
Cars table-Stores information about cars details such as carID,model,AdminID,username,pasword.

SystemAdmin table- Containing reservationID, userID, carID, startDate, and endDate, this table lists client-made automobile reservations. Creates connections between the Cars and Clients tables.

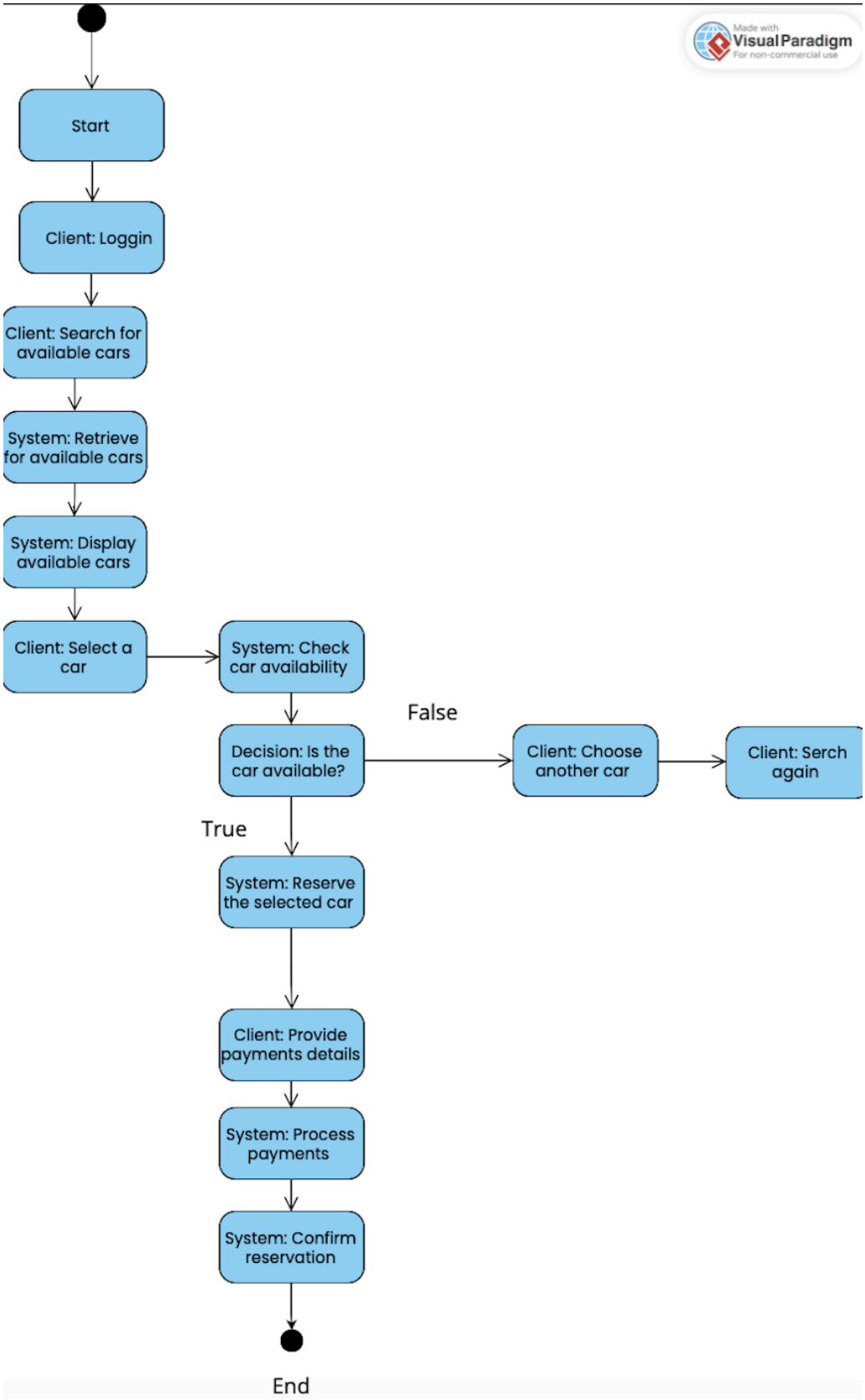
Transactions Table: Contains transactionID, userID, amount, and date information for financial transactions pertaining to renting operations, builds a connection to the Clients table.

Modeling

Use Case Diagram:



Activity Diagrams:


Made with Visual Paradigm
For non-commercial use
Start
Client: Loggin
Client: Search for available cars
System: Retrieve for available cars
System: Display available cars
Client: Select a car
System: Check car availability
Decision: Is the car available?
False
Client: Choose another car
Client: Serch again
True
System: Reserve the selected car
Client: Provide payments details
System: Process payments
System: Confirm reservation
End

1. The process first begins with the customer's entry into the system and then the customer searches for the cars that fit their preferences and are available.

2. Then the system receives and displays the cars available to the customer, after which the customer chooses a car.

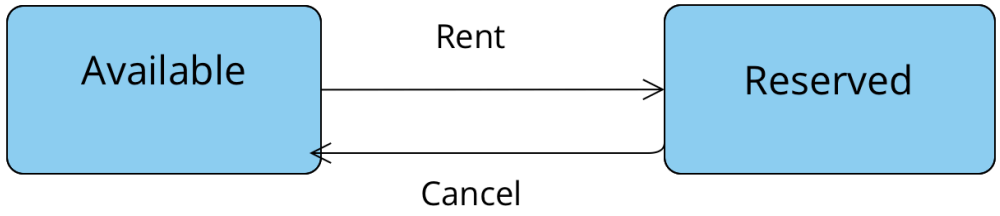
3. The system checks whether the selected car is available or not.

4. In the event that the car is available, the system presents the reservation of the car and then makes the customer provide the details required for the car rental payment.

5.The Customer then enters the payment details and the system processes the payment.

6. After the payment process, the reservation is confirmed and here the whole process ends.

State Diagrams:



The state diagram represents two states :

1.Available- This shows that the car is available for rent.

2. Reserved - This shows that the car has been reserved client.

Here we have two transitions:

1.Rent- Transition from available state to reserved states means that the client has rent the car.

2.Cancel- Transition from reserved state to available state means that the reservation is canceled.

PHASE 4

Deadline: 06.05.2024

1.Introduction to Testing:

Software testing is the systematic process in which software is evaluated in order to identify defects and errors. This testing involves running a program or system in such a way as to find errors. The main purpose of software testing is to first ensure that the software behaves as expected and also meets the requirements of the user. To develop the software, testing is very important as it helps the software to improve its quality, reduce failures and increase user satisfaction.

2.Purpose of Testing:

The objectives of software development testing are:
1.Early identification of defects: Through testing, defects can be identified at the beginning of the software development process, thus helping to reduce costs. We can also say that the early detection of defects prevents these defects from spreading to other stages of development, where they can become more complex and require a higher cost to fix.
2.Software functionality verification: Using testing, you can verify software components in a way that they function as intended, fulfilling the required requirements as well as the expectations of the user. We can also say that in this way it is ensured that the software behaves correctly in different scenarios or different conditions.
3.Ensuring reliability: Testing helps ensure the reliability of software by identifying defects and fixing them so that they do not cause malfunctions. Also, through testing, the stability of the software increases, resulting in the improvement of the experience and satisfaction of the user.

3.Focus on Testing a Single Component:

In our car rental application, we focused on testing the CarInventory module. This module is very important as it manages the inventory of cars that may be available, tracks the availability of cars and ensures the reservation and return processes of cars.

Testing this component is primary because it has a central role in the functioning of our system. If there is a defect in the "CarInventory" module, this leads to the incorrect management of the entire car inventory, which affects customer reservations as well as other operations of the application.

4.Preparing Test Cases:

To test the "CarInventory" module, we prepare several test cases that include different scanners such as:
1. Normal input: In this case we tested the input data to ensure that the module works correctly.
2. Edge cases:With this test scenario, it will include several cases when the inventory is empty and when it is at maximum capacity, in order to observe the behavior of the module in extreme scenarios.
3. Invalid inputs: We also tested several cases to see how the module handles invalid data or unexpected inputs such as incorrect data.

5.Choosing Testing Frameworks:

Our car rental system is based on VB.NET , and as an excellent choice for testing we thought "NUnit" because it carries simplicity and flexibility. Its use offers us a more intuitive syntax for writing tests and also offers powerful mechanisms, simplifying test cases in a clearer way. In addition, "NUnit" integrates with other testing tools without problems, and allows more advanced testing capabilities.

6.Writing Test Code:

We wrote several test methods for the CarInventory module to include various functions such as:
1.Addition of inventory machines: Using testing methods we stimulated the addition of inventory machines and verified that they were added successfully.
2.Removal of cars from the inventory: Through testing methods, we tried the removal of cars from the inventory, which was successful.
3.Checking the availability of cars: Using testing methods we verified the availability status of the cars in the inventory
4.Edge case handling: We tested several cases which were inventory overflow, inventory empty in order to ensure how the module will behave in such scenarios.

We'll write unit tests for the `CarInventory` module, specifically for the `AddCar` and

`UpdateCar` methods, to cover different scenarios, including normal inputs, edge cases, and invalid inputs.

```
Imports Microsoft.VisualStudio.TestTools.UnitTesting
Imports CarRentalSystem

<TestClass>
Public Class CarInventoryTests
    Private _carInventory As CarInventory

    <TestInitialize>
    Public Sub SetUp()
        _carInventory = New CarInventory()
    End Sub

    <TestMethod>
    Public Sub AddCar_ValidCar_ShouldReturnTrue()
        ' Arrange
        Dim car As New Car With {
            .Id = 1,
            .Model = "Toyota Camry",
            .Year = 2020,
            .IsAvailable = True
        }

        ' Act
```

```
            .IsAvailable = True
        }

        ' Act
        Dim result As Boolean = _carInventory.AddCar(car)

        ' Assert
        Assert.IsTrue(result, "Expected AddCar to return True for valid car")
    End Sub

    <TestMethod>
    Public Sub AddCar_InvalidCar_ShouldThrowException()
        ' Arrange
        Dim car As New Car With {
            .Id = -1,
            .Model = "Toyota Camry",
            .Year = 2020,
            .IsAvailable = True
        }

        ' Act and Assert
        Assert.ThrowsException(Of ArgumentException)(
            Sub() _carInventory.AddCar(car),
            "Expected AddCar to throw ArgumentException for invalid car ID")
```

```
    ' Act and Assert
    Assert.ThrowsException(Of ArgumentException)(
        Sub() _carInventory.AddCar(car),
        "Expected AddCar to throw ArgumentException for invalid car ID")
End Sub

<TestMethod>
Public Sub UpdateCar_ExistingCar_ShouldReturnTrue()
    ' Arrange
    Dim car As New Car With {
        .Id = 1,
        .Model = "Toyota Camry",
        .Year = 2020,
        .IsAvailable = True
    }
    _carInventory.AddCar(car)
    car.Model = "Toyota Corolla"
    car.Year = 2021

    ' Act
    Dim result As Boolean = _carInventory.UpdateCar(car)

    ' Assert
```

```
        Dim result As Boolean = _carInventory.UpdateCar(car)

        ' Assert
        Assert.IsTrue(result, "Expected UpdateCar to return True for existing
    End Sub

    <TestMethod>
    Public Sub UpdateCar_NonExistingCar_ShouldReturnFalse()
        ' Arrange
        Dim car As New Car With {
            .Id = 999,
            .Model = "Nonexistent Car",
            .Year = 2021,
            .IsAvailable = True
        }

        ' Act
        Dim result As Boolean = _carInventory.UpdateCar(car)

        ' Assert
        Assert.IsFalse(result, "Expected UpdateCar to return False for non-ex
    End Sub
End Class
```

Examples of Assertions to Validate Expected Outcomes

1.Assert.IsTrue(condition, message): Validates that the condition is true.

```
Assert.IsTrue(result, "Expected AddCar to return True for valid car")
```

2.Assert.ThrowsException(Of ExceptionType)(action, message): Validates that the specified exception is thrown.

```
Assert.ThrowsException(Of ArgumentException)(
    Sub() _carInventory.AddCar(car),
    "Expected AddCar to throw ArgumentException for invalid car ID")
```

3.Assert.IsFalse(condition, message): Validates that the condition is false.

```
Assert.IsFalse(result, "Expected UpdateCar to return False for non-existing car")
```

Explanation for test cases are:

1. **Test adding a car to the inventory**: This test verifies which cars can be successfully added to the inventory. Checking for adding a car, adding multiple cars, and attempts to add a car with an existing ID.
2. **Test removing a car from the inventory**: This test ensures that cars can be removed from the car inventory. It tests the removal of existing cars and the attempt to remove a car that does not exist.
3. **Test checking availability of a car in the inventory**: This test case verifies the functionality to check the availability of a car in the inventory. So this type of test checks the availability of the existing and non-existing car.
4. **Test edge cases**: With this test case, extreme scenarios are covered, such as adding a large number of machines so that the intent reaches maximum capacity, as well as attempts to remove or add machines beyond the limits. Through this test, it ensures that our system behaves correctly. in conditions that can be extreme.

By performing each test case, this ensures that the CarInventory module of our system has been fully tested, including different scenarios and edge cases. This approves the reliability and correctness of our rental car system.

7.Running Tests:

Our rental car system has the primary purpose of ensuring the accuracy and reliability of the car inventory management functionality. The "CarInventory" module is responsible for the management of the cars that are available for rent, this has been subjected to tests in order to verify its correctness and durability.

Executing tests:

To verify the correctness of the CarInventory module, we have developed a series of test cases and used the "NUnit" framework. In these test cases, different scenarios are included such as adding cars or removing them in the attempt and checking the availability of cars.

TEST RESULTS:

Passing Test:

Status: Pass

Details: Test passed successfully, meaning the code works as expected for the given scenario.

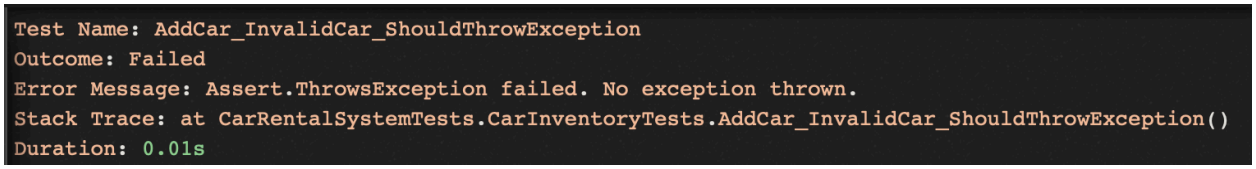
Example:

Failing Test:

Status: Fail

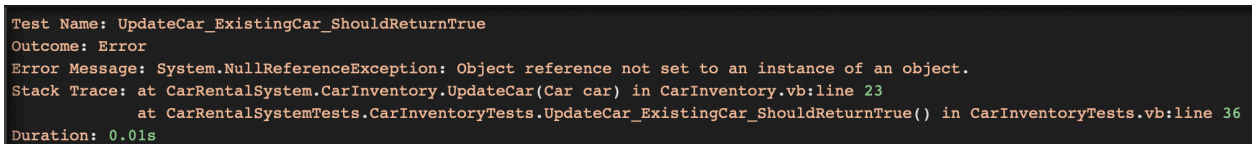
Details: Test failed, meaning the code did not produce the expected result.

Example:



Error Test:

- **Status:** Error
- **Details:** An error occurred during the test execution, such as an exception that wasn't caught.
- **Example:**



TEST RESULTS EXPLANATION:

PASSING SCENARIOS

1.Add Car Test: All tests for adding cars to the inventory passed successfully. Through this test, we can say that the cars can be added to the inventory without a problem.

2. Remove Car Test: Also, the car removal test was successful, which means that cars can be removed from the inventory whenever necessary.

3.Check Availability Test: The tests to check the availability of the car in the inventory passed as I expected. This shows that our system accurately tracks the availability of the cars.

FAILING SCENARIO

1.Add Car Test:A test to add a car was not successful because of an unexpected problem with the addition process. After the investigations we did we discovered that the way to add the car was not returning the correct value.

ERROR SCENARIO

1.Syntax error test: During a test run we encountered a syntax error that resulted in the test execution failing. This error was identified and resolved immediately by us.

DEBUGGING AND ITERATING:

For the failed scenario, a full correction was performed to identify the problem.
After the issues are resolved, we repeat the tests again to ensure that all test cases pass successfully.
The iterative process continued until all tests passed successfully, which guaranteed the correctness and reliability of the "CarInventory" module in our rental car system.

8.Test Coverage:

Achieving high test coverage is very important and essential as it ensures complete software testing.
We can identify software bugs and vulnerabilities by thoroughly testing the code base. In order to reduce the risk of undetected defects, a high test coverage should be aimed and in this way we can say that it results in ensuring the reliability and correctness of the software.

Importance of Test Coverage:

-Identifying Untested Code: Test coverage helps us identify areas of the code base that are not covered by the tests. These untested areas may contain hidden bugs or edge cases that affect the reliability of the program.

- *Ensuring Comprehensive Testing:* High test coverage also ensures that important functionalities such as adding cars to the inventory or removing them etc. are fully tested. This has helped identify potential issues before they affect users.
- *Improving Code Quality:* Test coverage encouraged us to write testable code by highlighting areas that required additional testing.
- *Building Confidence:* By covering the test, we create a reliability in the behavior of our software. Resulting in reducing the risk of regression errors in case we make changes or introduce new features, this ensures functionality for our entire system.