OOSE LAB PROJECT
BY
BHARAT
BHUSHAN

AIRBUS MANAGEMENT SYSTEM

ANDHRA UNIVERSITY

ఆంధ్ర విశ్వకళా పరిషత్

Accredited by NAAC with 'A' Grade ISO 9001: 2015 Certified

CERTIFICATE

Certified that is a bonafide Record of Practical work Done by Mr.B. BHARAT BHUSHAN With Registration Number 320506402015 of CSE 3/6 B.TECH + M.TECH Class in OBJECT ORIENTED SOFTWARE ENGINEERING Laboratories of ANDHRA UNIVERSITY COLLEGE OF ENGINEERING College during the year 2022 – 2023.

No. of expts. Done and certified.

Lecture in-Charge

Head of the Department

Date:

INDEX

ABSTRACT

Airbus management system is a software automated project that is built to support the management and working of the passengers, staff, airway systems. This system provides UI supported interface for passengers to book tickets, cancel and other features to support passenger’s requirement and ease their experience during whole journey process. This project has a central system to management the employee related work and has the whole functionality for managing their schedule, create tasks, take feedback, control norms and update payment related events tasks.

An airline reservation system is a complex software application that is responsible for managing a wide range of tasks related to the booking and management of flights. The software typically consists of a user interface that allows travellers to search for flights based on their preferences, such as dates, departure and arrival cities, and the number of passengers. Once a flight is selected, the system checks the availability of seats and provides pricing information based on the fare class and other factors. The software also handles the processing of payments and the issuance of tickets, as well as the management of reservations, which may include the ability to change or cancel flights. Behind the scenes, the system communicates with other systems and databases to access information on flight schedules, seat availability, and pricing data. In short, an airline reservation system is a critical component of the travel industry, enabling airlines and travel agencies to efficiently manage bookings and provide a seamless experience for travellers.

Features of airbus management system are built by providing a capability to the user by login/ sign in activity. This helps the central system to keep a unique security measure and identification. Passengers can view their profile, check their any outstanding refunds, make bookings for travel and have any offers corresponding to their economic bank activity. Central system sitting at the server has the API mean support to take/manage payment related tasks. The API’s enables all the banks to securely authorize and initiate payments of users of their choice.

UNIFIED MODELLING LANGUAGE

1.1 UNIFIED MODELLING LANGUAGE

Unified Modelling Language (UML) is a general-purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is **not a programming language**; it is rather a visual language. We use UML diagrams to portray the **behaviour and structure** of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. It's been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

1.1.1 Do we really need UML

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So, UML becomes essential to communicate with non-programmer essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

1.2 Building Blocks of the UML

The UML encompasses three kinds of building blocks:

- 1.Things
- 2.Relationships
- Diagrams

1.2.1 Things

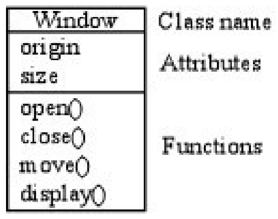
Things are the most important building blocks of UML. There are four kinds of things in the UML.

- Structural things
- Behavioural things
- Grouping things
- Annotation things

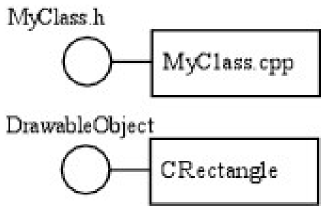
1.2.1.1 Structural things

Structural things are the nouns of the UML models. These are static parts of the model, representing elements that are either conceptual or physical. There are seven kinds of Structural things.

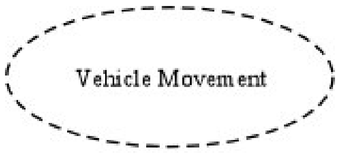
Class: A class defines methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity.



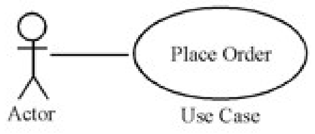
Interface: - A collection of functions that specify a service of a class or component, i.e. externally visible behavior of that class.



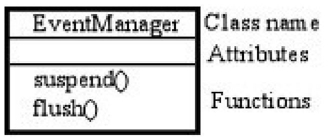
Collaboration: Collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behaviour that's bigger than the sum of all the elements.



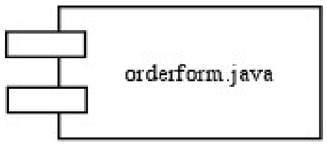
Use case: A sequence of actions that a system performs that yields an observable result. Used to structure behaviour in a model. Is realized by a collaboration.



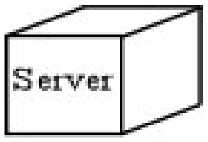
Active class: Like a class but its represents behaviour that runs concurrent with other behaviours, i.e. threading.



Component: A physical and replaceable part of a system that implements a number of interfaces. Example: a set of classes, interfaces, and collaborations.



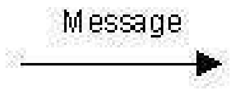
Node: A physical element existing at run time and represents a resource.



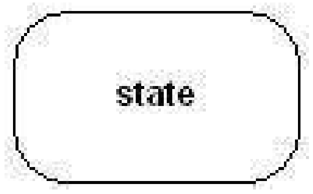
1.2.1.2 Behavioural Things

A behavioural thing consists of the dynamic parts of UML models. Following are the behavioural things −

Interaction − Interaction is defined as a behaviour that consists of a group of messages exchanged among elements to accomplish a specific task.



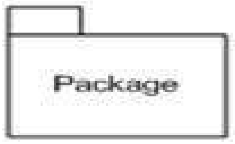
State machine − State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change



1.2.1.3 Grouping Things

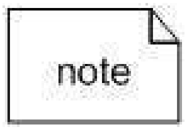
Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

Package − Package is the only one grouping thing available for gathering structural and behavioural things.



1.2.1.4 Annotation Things

Annotation things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



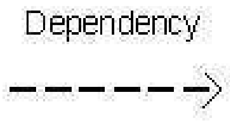
Relationship

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

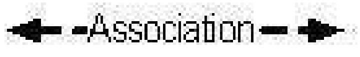
Dependency

Dependency is a relationship between two things in which change in one element also affects the other.



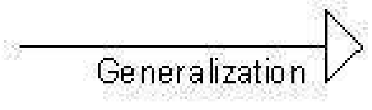
Association

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



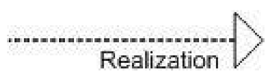
Generalization

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.



Realization

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces



1.3 UML Diagrams

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

1.3.1 Characteristics of UML

The UML has the following features:

- It is a generalized modelling language.
- It is distinct from other programming languages like C++, Python, etc.
- It is interrelated to object-oriented analysis and design.
- It is used to visualize the workflow of the system.
- It is a pictorial language, used to generate powerful modelling artefacts

1.3.2 Goals of UML

- Since it is a general-purpose modelling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the system, such that the system can be software or non-software.
- Thus, it can be concluded that the UML is a simple modelling approach that is used to model all the practical systems.

UML is linked with **object-oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

- **Structural Diagrams –** Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.

- **Behaviour Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

The image below shows the hierarchy of diagrams

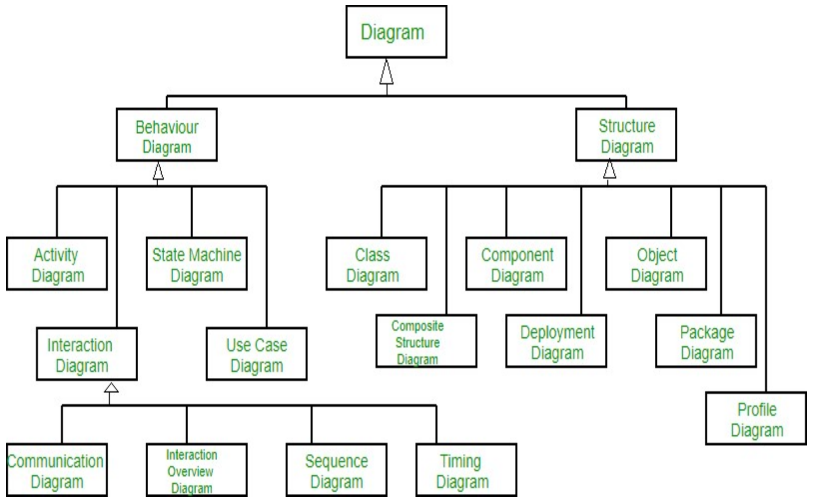


Figure 1: uml flow chart

1.4 Object Oriented Concepts Used in UML

- **Class –** A class defines the blue print i.e structure and functions of an object.
- **Objects** – Objects help us to decompose large systems and help us to modularize our system. Modularity helps to divide our system into understandable components so that we can build our system piece by piece. An object is the fundamental unit (building block) of a system which is used to depict an entity.
- **Inheritance** – Inheritance is a mechanism by which child classes inherit the properties of their parent classes.
- **Abstraction** – Mechanism by which implementation details are hidden from user.
- **Encapsulation** – Binding data together and protecting it from the outer world is referred to as encapsulation.
- **Polymorphism –** Mechanism by which functions or entities are able to exist in different forms.

1.5 Additions in UML 2.0

- Software development methodologies like agile have been incorporated and scope of original UML specification has been broadened.
- Originally UML specified 9 diagrams. UML 2.x has increased the number of diagrams from 9 to 13. The four diagrams that were added are: timing diagram, communication diagram, interaction overview diagram and composite structure diagram. UML 2.x renamed state chart diagrams to state machine diagrams.
- UML 2.x added the ability to decompose software system into components and sub-components.

1.6 Structural UML Diagrams

1.6.1. Class Diagram – The most widely use UML diagram is the class diagram. It is the building block of all object-oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.

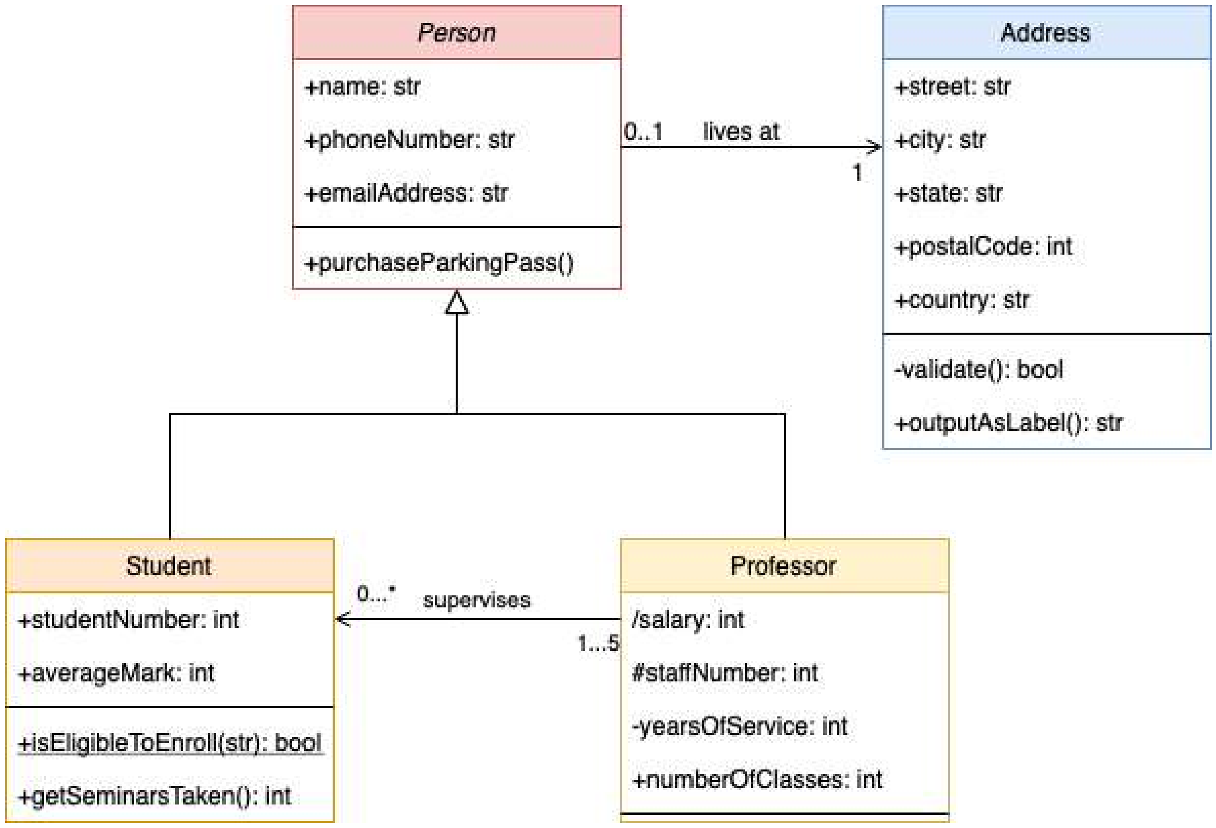


Figure 2: class diagram example

1.6.2 Composite Structure Diagram

We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system. A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves. They represent internal structure of a structured classifier making the use of parts, ports, and connectors. We can also model collaborations using composite structure diagrams. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.

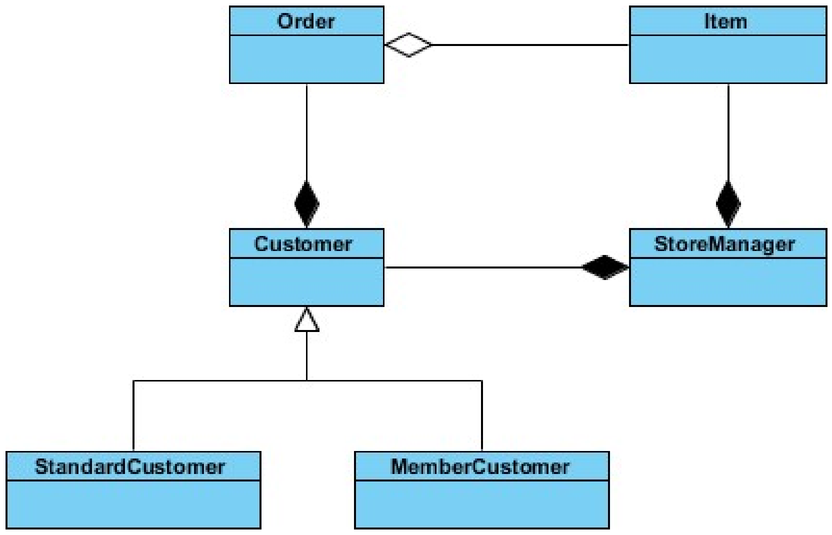


Figure 3: composite structure diagram example

1.6.3 Object Diagram

An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.

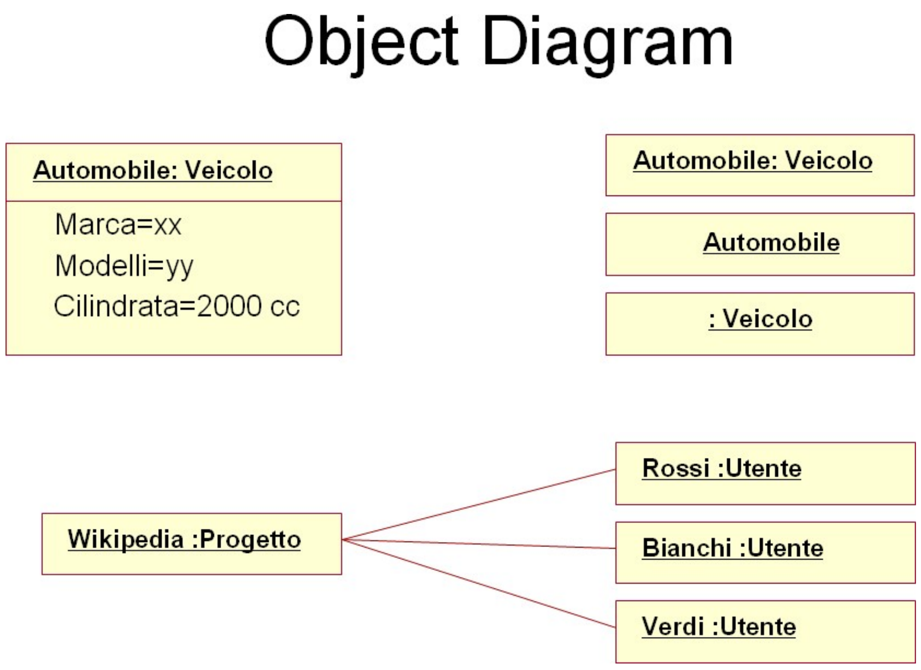


Figure 4: Object diagram example

1.6.4 Component Diagram

Component diagrams are used to represent how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.

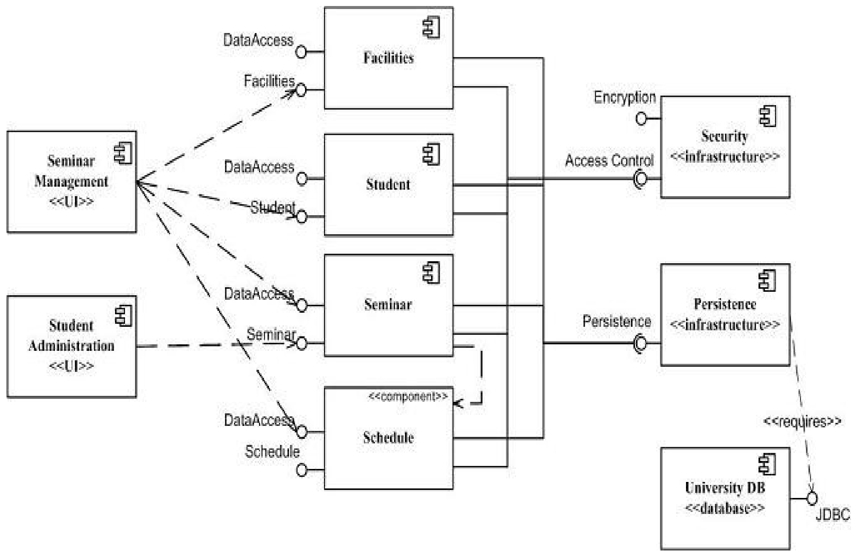


Figure 5: component Diagram example

1.6.5 Deployment Diagram

Deployment Diagrams are used to represent system hardware and its software. It tells us what hardware components exist and what software components run on them. We illustrate system architecture as distribution of software artefacts over distributed targets. An artefact is the information that is generated by system software. They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.

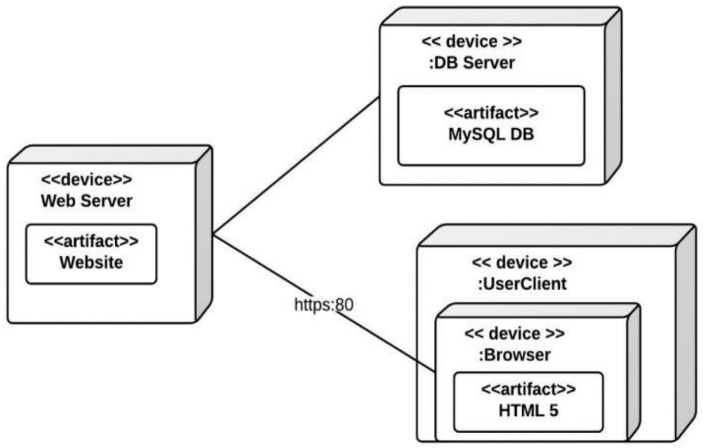


Figure 6: Deployment diagram example

1.6.6 Package Diagram

We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organise class and use case diagrams.

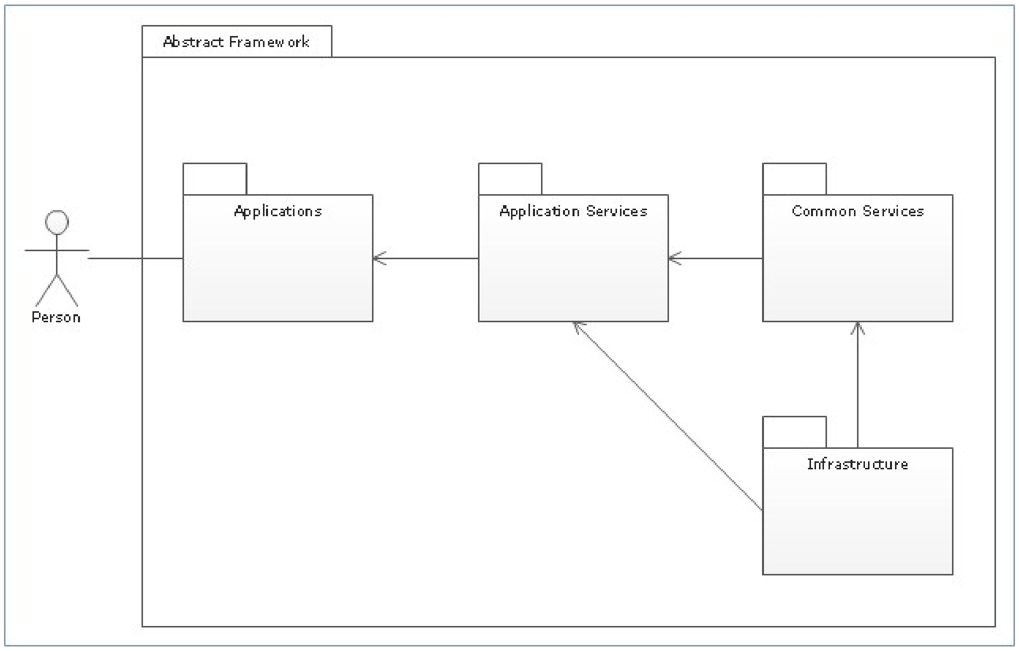


Figure 7: Package diagram example

1.7 Behaviours Diagrams

1.7.1 State Machine Diagrams

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioural diagram and it represents the behaviour using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams.** These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli.

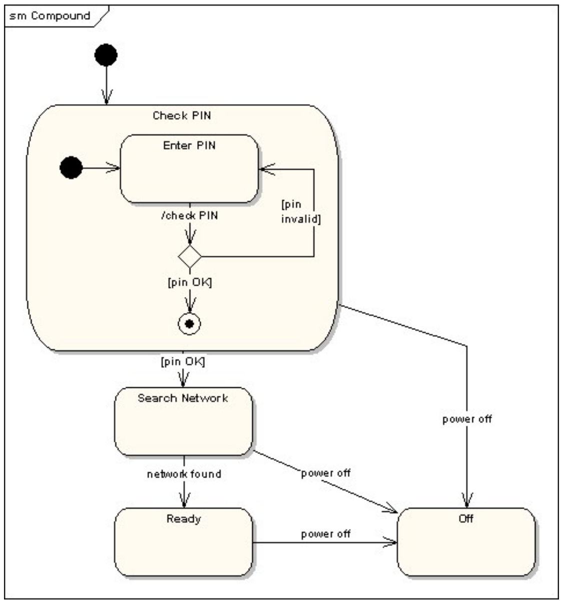


Figure 8: State Machine diagram

1.7.2 Activity Diagrams – We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

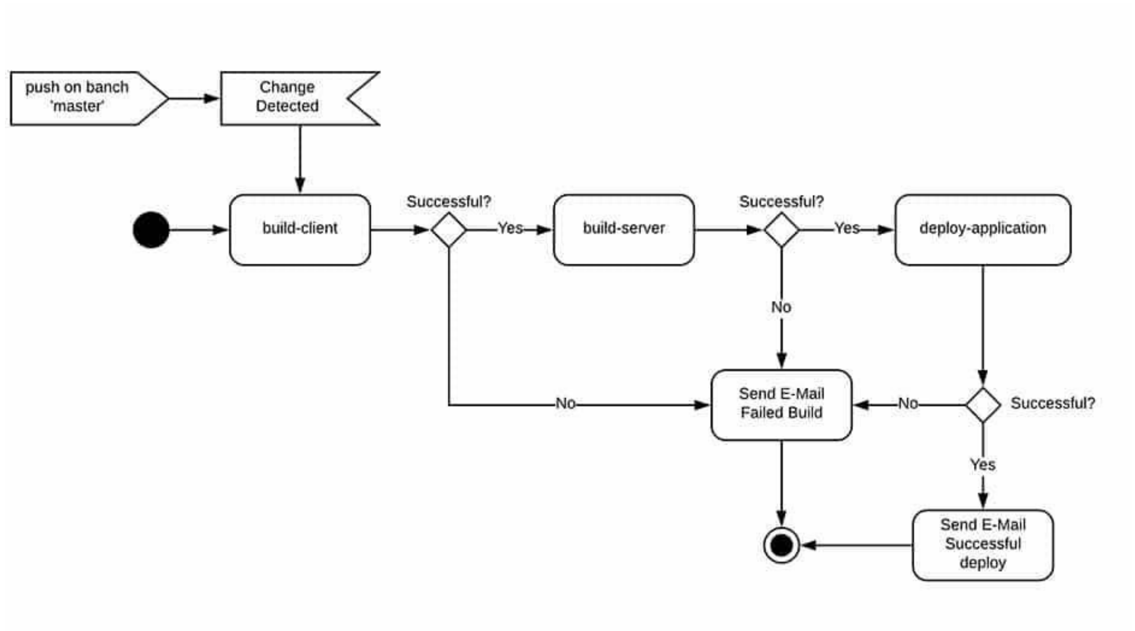


Figure 9: Activity Diagram

1.7.3. Use Case Diagrams – Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high-level view of what the system or a part of the system does without going into implementation details.

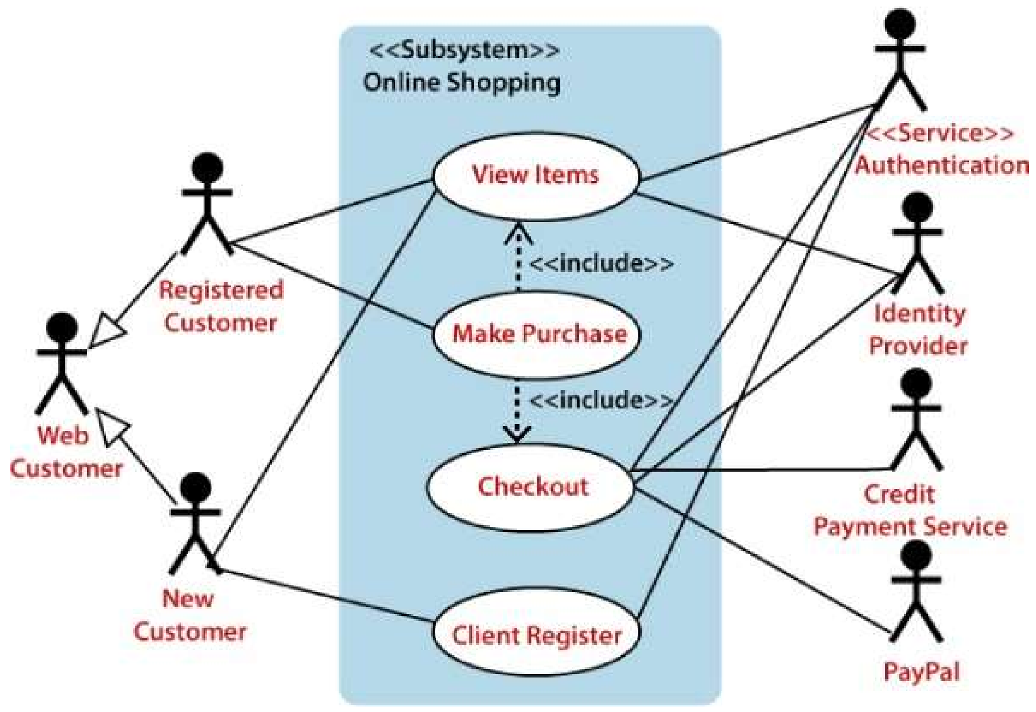


Figure 10: use case diagram

1.7.4 Sequence Diagram – A sequence diagram simply depicts interaction between objects in a sequential order i.e the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

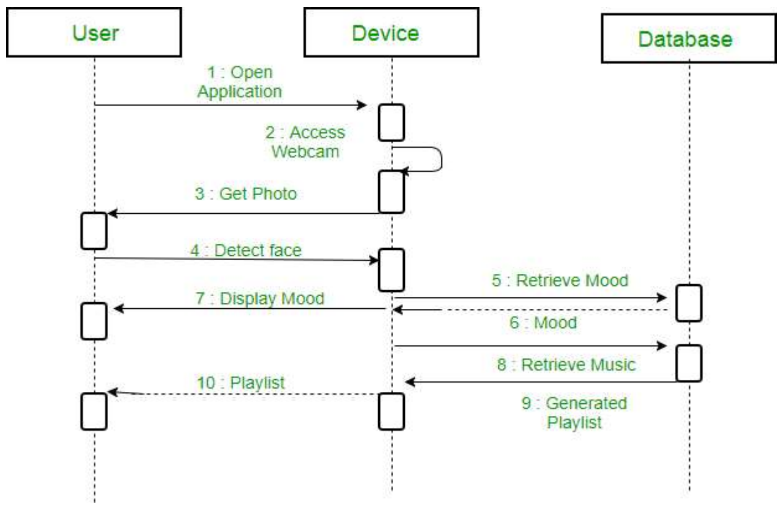


Figure 11: Sequence Diagram

AIRBUS MANAGEMENT SYSTEM

Introduction

An airbus management system (ABMS) is a comprehensive software solution that is designed to help airlines manage their day-to-day operations, optimize their flight schedules, and provide better customer service. An AMS typically includes modules for flight scheduling, crew management, aircraft maintenance, passenger reservations, ticketing, and other essential functions. The flight scheduling module is one of the most important components of an AMS. It enables airlines to create and manage their flight schedules, taking into account factors such as aircraft availability, crew schedules, and maintenance schedules. The scheduling module helps airlines optimize their flight schedules, reduce costs, and increase revenue by ensuring that flights are scheduled in the most efficient way possible. The module also helps airlines manage flight cancellations, delays, and changes.

Employee management

The crew management module is another critical component of an AMS. This module helps airlines manage their flight crew's schedules, track their qualifications and certifications, and ensure that they are available to fly when needed. The module also helps airlines manage crew training, vacation time, and other personnel-related issues. Crew management is critical because it ensures that airlines have qualified and available crews to operate their flights. The aircraft maintenance module of an AMS is another critical component. This module helps airlines manage their aircraft maintenance schedules, track maintenance issues, and ensure that all aircraft are safe and airworthy. The module also helps airlines manage their spare parts inventory, track warranty information, and ensure that all maintenance work is performed on time and according to the manufacturer's specifications. Aircraft maintenance is essential because it ensures that airlines operate safe and reliable aircraft.

Reservation of tickets

The passenger reservation module of an AMS enables airlines to manage their passenger reservations, ticketing, and check-in processes. The module allows airlines to create and modify reservations, issue tickets, and check-in passengers. The module also allows airlines to manage their frequent flyer programs and other customer loyalty programs. The passenger reservation module is critical because it helps airlines provide excellent customer service and ensure that all passengers are properly ticketed and checked-in for their flights. The ticketing module of an AMS allows airlines to manage their ticket sales, fares, and pricing. The module enables airlines to set fares, manage pricing rules, and create promotions and discounts. The module also allows airlines to track ticket sales and revenue, manage refunds and exchanges, and create reports on ticket sales and revenue. The ticketing module is essential because it helps airlines manage their pricing strategies and maximize their revenue.

In addition to these core modules, an AMS may include other features such as fuel management, cargo management, and ground handling management. These additional features can help airlines manage their operations more efficiently and reduce costs. AMS can also integrate with other systems, such as customer relationship management (CRM) and enterprise resource planning (ERP), to provide a comprehensive solution for airlines. By integrating with other systems, airlines can share data across different departments and functions, enabling them to make more informed decisions about their operations.

Cost efficiency and key performance

There are several different types of airline management systems available, ranging from basic systems designed for small regional airlines to complex systems designed for major international carriers. Some systems are developed in-house by airlines, while others are developed by third-party vendors. The cost of an AMS can vary widely depending on its complexity and the number of modules required. One of the key benefits of an AMS is that it enables airlines to make data-driven decisions. The system provides airlines with real-time information on flight schedules, crew availability, maintenance schedules, passenger reservations, and other critical factors. This information enables airlines to make informed decisions about scheduling, pricing, and other aspects of their operations. The system also provides airlines with detailed reports on key performance indicators (KPIs), such as load factor, yield, and revenue per available seat mile (RASM). Another benefit of an AMS is that it can help airlines improve their customer service. The system enables airlines to provide.

Employee management system

An airport employee management system (EMS) is a software solution designed to manage the various functions of airport employees. This system typically includes modules for employee scheduling, time and attendance tracking, payroll management, benefits administration, and other essential functions. One of the most important components of an EMS is the employee scheduling module. This module enables airport management to create and manage employee schedules, taking into account factors such as shift requirements, employee availability, and labor laws. The scheduling module helps ensure that the airport has the right number of employees working at all times to meet the demands of airport operations.

The time and attendance tracking module are another critical component of an EMS. This module enables airport management to track employee work hours and attendance, including arrival and departure times, breaks, and overtime. The module helps ensure that employees are paid accurately and according to their contracted hours. Payroll management is another important function of an EMS. The payroll module enables airport management to manage employee compensation, including salaries, wages, and benefits. The module also allows airport management to generate reports on employee pay and tax deductions and ensure compliance with applicable laws and regulations.

Benefits administration is another critical component of an EMS. The benefits module enables airport management to manage employee benefits, including health insurance, retirement plans, and other perks. The module allows airport management to track employee participation in benefit programs, enroll new employees, and manage open enrolment periods. An EMS may also include other modules to manage employee training, performance evaluations, and other human resource functions. These modules can help airport management ensure that employees have the necessary skills and qualifications to perform their jobs effectively.

One of the key benefits of an EMS is that it can help airport management reduce labor costs. By optimizing employee schedules and tracking employee work hours accurately, an EMS can help reduce overtime costs and ensure that employees are paid only for the hours worked. The system can also help airport management ensure compliance with labour laws and regulations, which can reduce the risk of costly fines and penalties. Another benefit of an EMS is that it can help improve employee productivity and satisfaction. By providing employees with clear schedules and tracking their time and attendance accurately, an EMS can help reduce confusion and increase accountability. The system can also help airport management ensure that employees are compensated fairly and receive the benefits they are entitled to, which can increase employee morale and job satisfaction. An EMS can also provide airport management with real-time data on employee performance, attendance, and other key metrics. This information can help airport management make informed decisions about employee scheduling, compensation, and benefits, and improve overall workforce management. There are several different types of EMS available, ranging from basic systems designed for small airports to more comprehensive systems designed for large international airports. Some systems are developed in-house by airport management, while others are developed by third-party vendors. The cost of an EMS can vary widely depending on its complexity and the number of modules required.

One of the challenges of implementing an EMS is ensuring that it integrates effectively with other airport systems, such as flight scheduling, ground handling, and security systems. Integration can be complex and require significant coordination between different departments and functions. However, once integration is achieved, an EMS can provide a comprehensive solution for airport workforce management. In conclusion, an airport employee management system is a critical component of airport operations. By providing a comprehensive solution for employee scheduling, time and attendance tracking, payroll management, benefits administration, and other essential functions, an EMS can help airport management reduce labour costs, improve employee productivity and satisfaction, and make informed decisions about workforce management.

2.USE CASE FOR FLIGHT BOOKING APP

2.1 Documentation

Booking App of the "***AIRBUS MANAGEMNET SYSTEM***" is an UI interface android/iOS supported app. This app is useful for the user to access, book, control, and query about their airway journeys with ease of automation. This app is the bridge between the airway central system and general public.

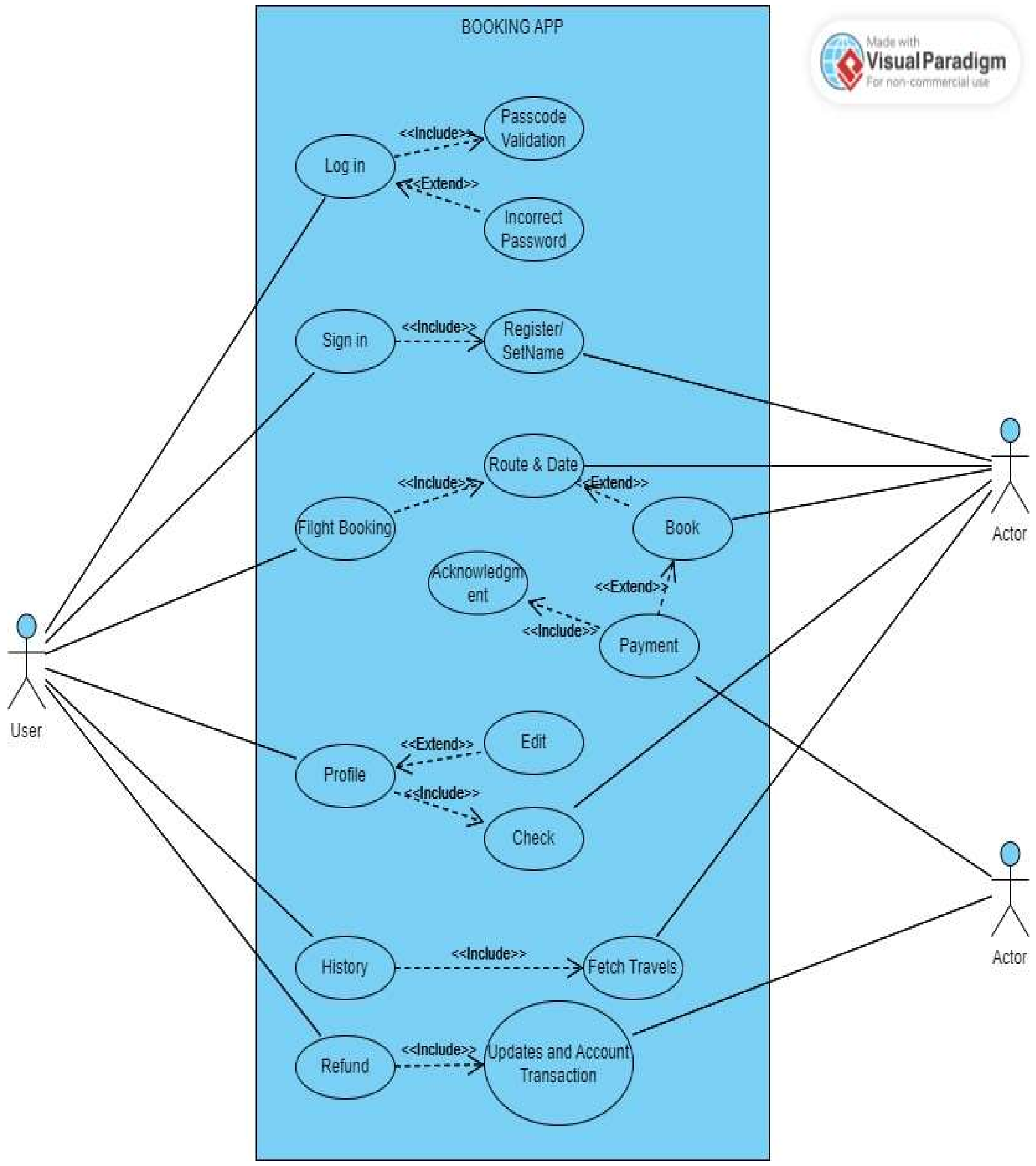


FIGURE 12: Use case for flight booking

2.2 Diagram content summary

- LOGIN
- SIGNIN
- FLIGHT BOOKING
- PROFILE
- HISTORY
- REFUND

2.3 USE CASES DETAILS

2.3.1 LOGIN

Description

Before using the airbus app, app users need to login into their account. This helps in authentication process. This guarantees the central system that the user is generic. This is the login use case that helps in checking credentials. If the account holders turn out to be legitimate, app allow the user to access the account, or it take appropriate steps to notify the account holder and may block on the account upon triggering some event.

Parent

Booking app

Include events

Password Validation

Extend events

Incorrect Password

Use case description

TABLE 1: Login

Name	Use case 01	
Precondition	User needs to have the credentials	
Flow of events	**Actor**	**System response**
	Actor enters login id and password	
		System validates the credentials. Upon successful match, system connects user with central system. If fails, system notifies and take necessary measures to ensure security.
	Actor gets response from system.	

2.3.2 SIGN IN

Description

New user can login after they become an entity of the central system. To register into central system, user provides necessary information from sign in use case. Sign in use case asks for users' information like name, phone, address, government identification id, and other necessary details as per requirement. User then gets a login id, continuing with setting password to the account. The next step involves recovery and security mechanisms.

Parent

Booking app

Include events

Set id and password

Use case description

TABLE 2: SIGNIN

Name	Use case 02 (sign in)	
Precondition	User should not have any accounts with the system.	
Flow of events	**Actor**	**System response**
	Actors chooses sign in option.	
		System takes all the necessary details and directs to set userid and password.
	Actor gets an account	

2.3.3 FLIGHT BOOKING

Description

Users checks/ book flights for journey through flight booking use case. User enters the journey credentials and submits the information. The system then connects to the central system requests flights available on path chosen. The central system responds with details of the availability of the flights on queried date. User gets an UI represented flights availability. If any flights available, users books the flight and makes payment by choosing appropriate bank gated API.

Parent

Booking app

Include events

Route and date, acknowledgment

Extend events

Book, payment

Use case description

TABLE 3: FLIGHT BOOKING

Name	Use case 03 (Flight booking)	
Precondition	User has proper journey plan govt identification for all passengers' users wants to book flight for.	
Flow of events	**Actor**	**System response**
	Actor clicks on Flight Booking icon	
		UI related to Flight booking appears
	Actor enters from and destination of the journey.	
		System requests available Flights that travel through the given path. Receives the flights in the path from central system and presents to the user
	Actor now could check the available flights or proceeds to book flight.	
		If system receives response from user for booking, system sends request for booking to central system. System then redirects user to payment.
	Actor pays for booking, if any.	
		If payment is done by user, an acknowledgment is presented to the user or appropriate message is shown to the user.

2.3.4 PROFILE

Description

User of the Airbus system has a functionality for checking and updating about their self. Through profile use case they can access this functionality. With this use case they can check their profile or make any changes to their profile.

Parent

Booking app

Include events

Check

Extend events

Edit

Use case description

TABLE 4: PROFILE

Name	Use case 04 (PROFILE)	
Precondition	User must first login into the account.	
Flow of events	**Actor**	**System response**
	Actor clicks on profile icon.	
		System presents user details to the Actor.
	Actor may choose to edit the profile or actor leaves.	
		If actor chooses edit, system takes necessary changes from actor updates the central system for changes.

2.3.5 HISTORY

Description

User can view all their activity by history use case. This presents the user with all their travel as well as payments they made in the airbus management system.

Parent

Booking app

Include events

Fetch Travels/ Bookings

Use case description

TABLE 5: HISTORY

Name	Use case 05 (HISTROY)	
Precondition	User needs to login into their account.	
Flow of events	**Actor**	**System response**
	Actor chooses history icon in the system	
		System requests all the activity that the user made through the system from central system. System receives the history form the central system and shows it to the user

2.3.6 REFUNDS

Description

Users can find their any refund related issues from refund use case. It is usual in airbus systems to cancel out flights due unfavourable events. User receive all the required information from this use case.

Parent

Booking app

Include events

Updated and account Transactions

Use case description

TABLE 6: REFUNDS

Name	Use case 06 (REFUND)	
Precondition	User needs to logged into the account	
Flow of events	**Actor**	**System response**
	Actor clicks on refund activity icon	
		System keeps the track for refunds needed to be made and references to bank gates to verify and shows appropriate update to the users.

3.USE CASE FOR MANAGING APP

3.1 Documentation

The airbus management system presents and controls the activity of the employees of the airbus company by Employee management app. The use case for the employee management app list out all the detail key components of it. The airbus management system expects to control the schedules of the staff, keep their work progress in the app, manage their payments and alert with any notification.

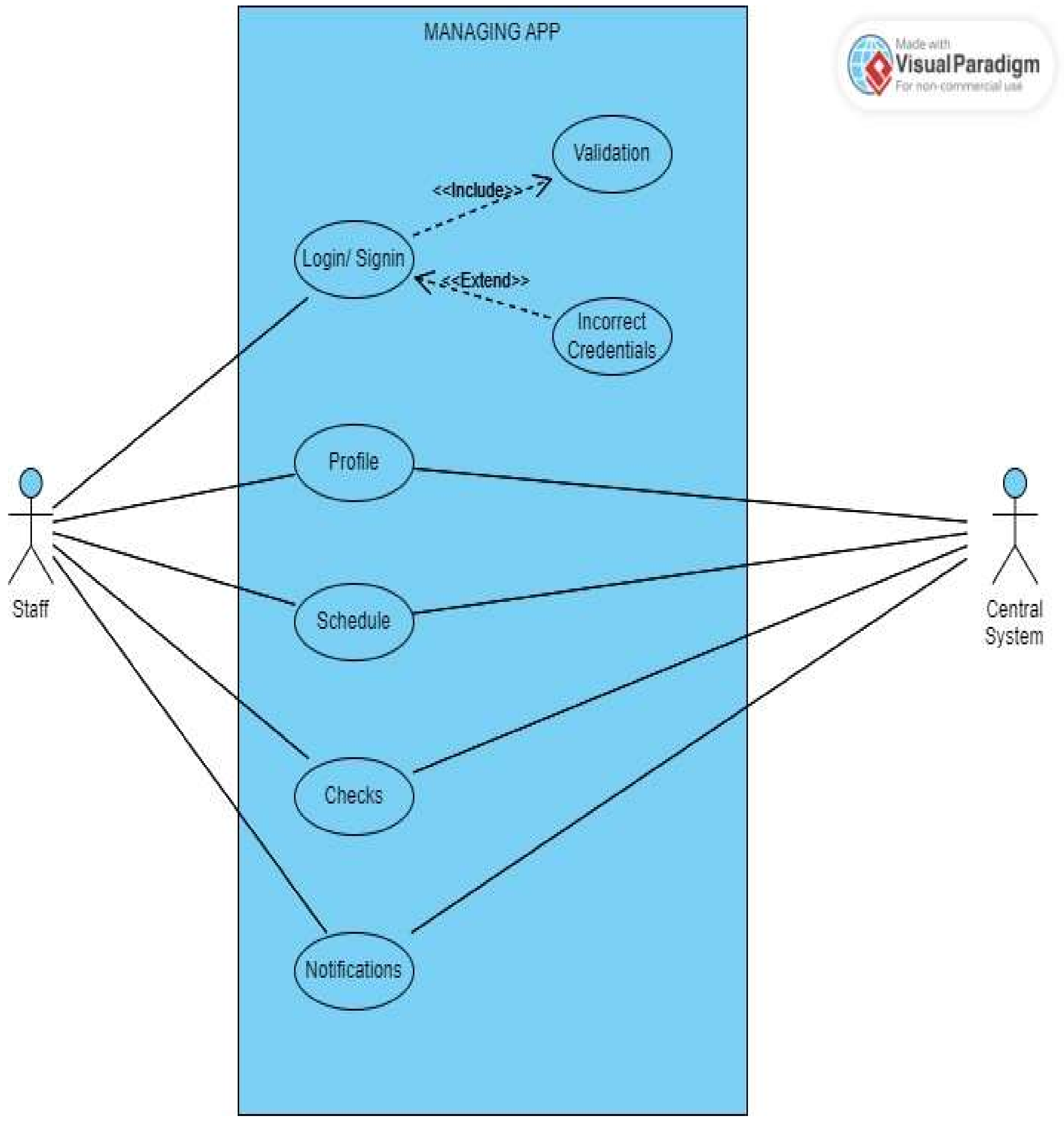


FIGURE 13: USE CASE FOR MAPPING APP FOR STAFF

3.2 Diagram content summary

- LOGIN/SIGN IN
- PROFILE
- SCHEDULE
- CHECK
- NOTIFICATIONS

3.3 USE CASES DETAILS

3.3.1 LOGIN/SIGN IN

Description

Employees of the can get all their work and payment related information from using this app. Airbus management system has designed this interface for efficiently managing them. To achieve this goal, new employees are provided with sign in option and old employees can login to their account in order to finish their tasks.

Parent

Managing app

Include events

Validation

Extend events

Incorrect credentials

Use case description

TABLE 7: LOGIN/SIGNIN

Name	Use case 01 (LOGIN/SIGNIN)	
Precondition	User needs to have the credentials	
Flow of events	**Actor**	**System response**
	Actor enters login/ sign in option	
		For login activity by the user, the checks the credentials and take appropriate steps. In case for sign in activity, case is redirected to sign and account activation page.

3.3.2 PROFILE

Description

Employees of the airbus management system can view their profile and have complete records in them. Profile use case is a system related task maintained by the central system for keeping records and performance parameters of the employee.

Parent

Managing app

Use case description

TABLE 8: PROFILE

Name	Use case 02 (PROFILE)	
Precondition	User needs to logged in.	
Flow of events	**Actor**	**System response**
	Actor tabs profile view	
		System responds with employees' statistics and information.

3.3.3 SCHEDULE

Description

Central system's first most reason for creating this interface is to give the employees proper instruction on the work schedule. Employees of the company can access their work schedule by simply taping the use case Schedule.

Parent

Managing app

Use case description

TABLE 9: SCHEDULE

Name	Use case 03 (SCHEDULE)	
Precondition	Employees must log in.	
Flow of events	**Actor**	**System response**
	Actor tabs schedule use case	
		System presents the workday schedule to the Actor.

3.3.4 CHECKS

Description

Use case check provides the staff of "AIRBUS MANAGMENENT SYSTEM" with accessing their payments/compensation history. This use case provides details of follow up transaction details of and acknowledgements.

Parent

Managing app

Use case description

TABLE 10: CHECKS

Name	Use case 04 (CHECKS)	
Precondition	Actors needs to be logged into their account.	
Flow of events	**Actor**	**System response**
	Actor chooses checks use case.	
		System connects with the central system and requests user payments information. Upon receiving requested information, system presents it to Actor.

3.3.5 NOTIFICATIONS

Description

This notification use case provides the employees with most urgent messages they need to receive. This pings the employees when a notification is received by the system.

Parent

Managing app

Use case description

TABLE 11: NOTIFICATION

Name	Use case 05 (NOTIFICATION)	
Precondition	Actors needs to be logged into their account.	
Flow of events	**Actor**	**System response**
		System receives message for Actor from central system. System pings Actor
	Actor reacts and sees the notification	
		System updates receiving information with the central system or follows up.

4.CLASS DIAGRAM FOR AIRBUS MANAGEMENT SYSTEM

4.1 DESCRIPTION

Class diagram describes the classes that reside in the central server that control all aspects of the use case modularity.

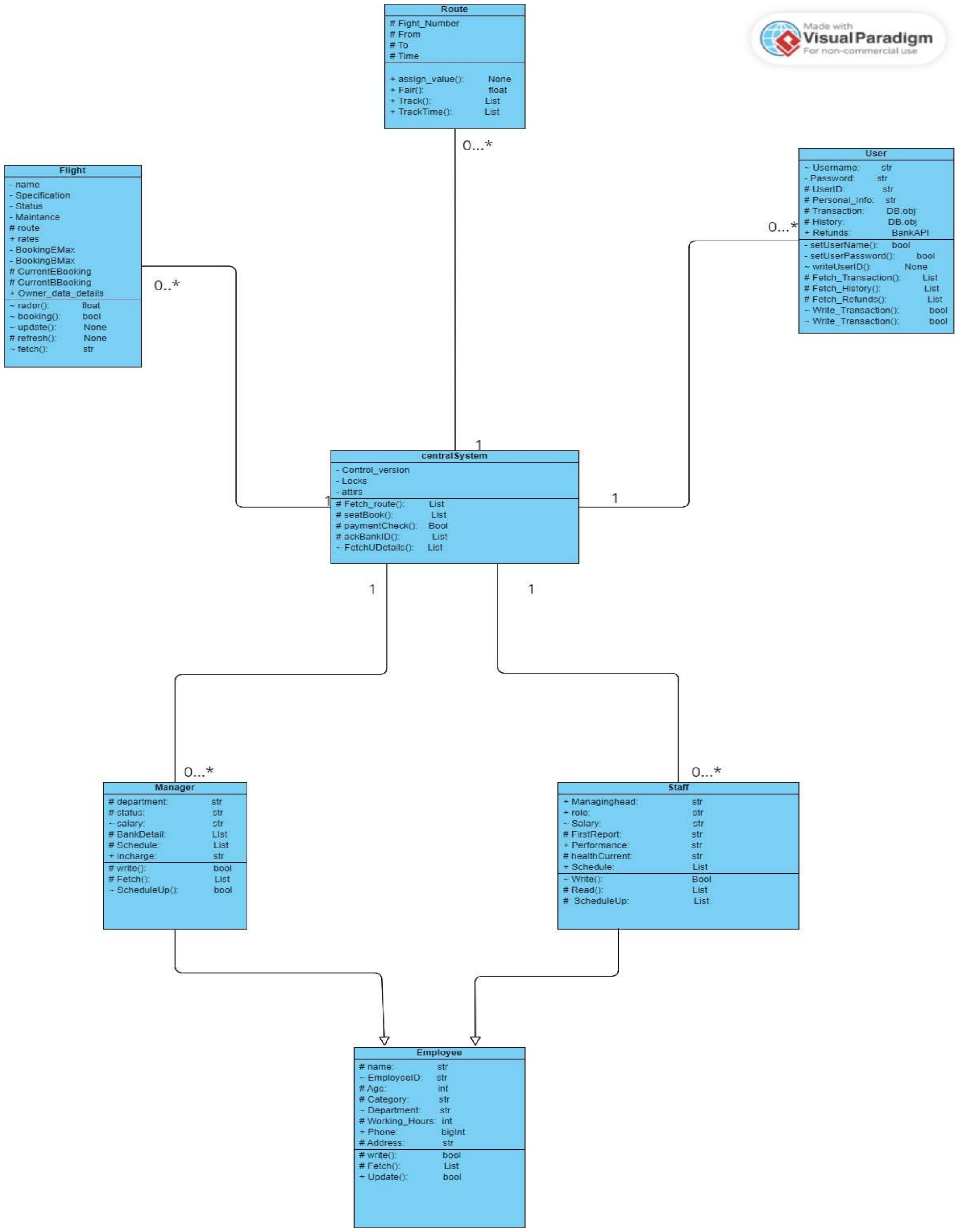


FIGURE 14: CLASS DIAGRAM FOR ABMS

4.2 DIAGRAM CONTENT DETAILS

- Central System
- Flights
- Route
- User
- Manager
- Staff
- Employee

4.3 CLASS DIAGRAM DETAILS

4.3.1 CENTRAL SYSTEM

Description

Central system is the main thread that controls all the aspects of the "airbus management system". This class is what receives information from the user threads and processes, manages, decides, edits and controls every aspect of the central working. This class also take cares of the employees related management work.

Visibility: public

Operation summary:

TABLE 12: CENTERAL SYSTEM

Declaration	Operation	Return type
String []	Fetch_route ()	List
String []	Seat_book ()	List
bool	paymentCheck ()	Bool
Void	AckBankID ()	None
Protected String []	FetchUDetails ()	List

4.3.2 FLIGHTS

Description

Flight class contains all the necessary implementation that is used by its objects for managing a flight. These implementations help in reserving, managing flight details, accessing its current status and etc….

Visibility: protected

Operation summary:

TABLE 13: FLIGHTS

Declaration	Operation	Return type
public float	Rador()	Float
Protected bool	Booking()	Bool
Void	Update()	None

Public void	Refresh()	None
Public string	Fetch()	str

4.3.3 ROUTE

Description

Routs class is implemented to keep current tracks of from to to, i,e ., from to destination. These routes include all the routes objects that has current services (an active service of flight in the way). This route provides a means for central system for checking all the flights that are active on the current route and fetching them when an user tries to check available flight to travel.

Visibility: protected

Operation summary:

TABLE 14: ROUTE

Declaration	Operation	Return type
Protected void	Assign_values()	None
Protected float	Fair()	float
Protected string[]	Track()	List
Protected string[]	trackTime()	List

4.3.4 USER

Description

User class is associated to central system, this class has implementation details that support creating user objects to each individual. This class support managing users. With the help objects of user, central system can validate, respond to user login, maintain their history and fetch their info etc….

Visibility: protected

Association: Associated to central system.

Operation summary:

TABLE 15: USER

Declaration	Operation	Return type
Void	setUserName()	None
Void	setPassword()	None
Bool	WirteUserID()	bool
Protected string[]	Fetch_Transactions()	List
Protected String[]	Fetch_Refund()	List
Protected bool	WriteTransaction()	Bool
Protected bool	WriteRefunds()	Bool

4.3.5 MANAGER

Description

Manager class is associated with central system. This class lets create manager entities in the central system and help to build bridge between managers and his/her staff.

Visibility: protected

Association: Associated to central system.

Operation summary:

TABLE 16: MANAGER

Declaration	Operation	Return type
Bool	Write()	Bool
Protected string[]	Fetch()	List
Public string[]	Scheduleup()	List

4.3.6 STAFF

Description

Staff class provides the date structure that is required to manage the whole employees of the company by the central system. This class has bear algorithms that help in reducing the effort of maximum work the workforce as manager.

Visibility: protected

Association: Associated to central system.

Operation summary:

TABLE 17: STAFF

Declaration	Operation	Return type
Public book	Write()	Bool
Public string[]	Read()	List[]
Public string[]	ScheduleUp()	String[]

5.SEQUENCE DIAGRAM FOR AIRBUS MANAGEMENT SYSTEM

1. FLIGHT BOOKING

5.1 DOCUMENTATION

The sequence diagram depicts the flow of working from actor to the all corresponding function of classes.

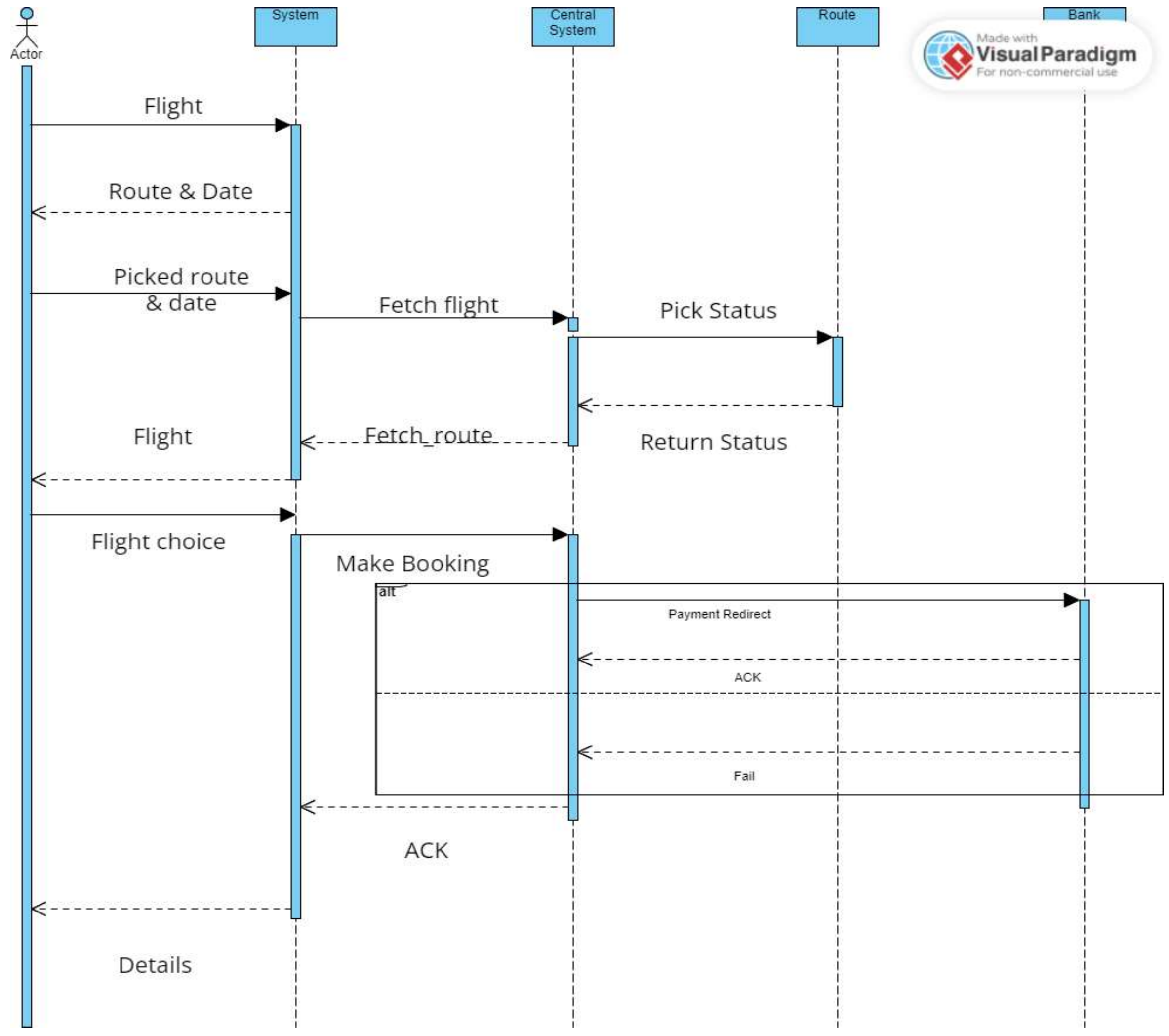


FIGURE 15: SEQUENCE DIAGRAM FOR FLIGHT BOOKING

5.2 Diagram content details

- Actor
- System
- Central system
- Route
- Bank

5.3 Flow process for flight booking.

TABLE 18: FLIGHT BOOKING SEQUENCE

S. no	Actor	System	Central System	Route
1.	Actor tabs on flights icon.			
2.		System responds with requesting route and date of journey.		
3.	User enter route and date of journey			
4.		System requests Central system, flight available on the route with specified date		
5.			Central system receives the request from the user entity. Central system then reaches class route's track function for getting the available flights on the route specified	
6.				Track function of the central system fetches all the available flights on the routes and return it as a list to central system.
7.			Receives list of flights available as list from track function. Central system then returns the available flights to the system.	
8.		System receives the requested information from the central system through network. System then uses its UI models to present to the actor.		

9.	Actor checks the available flights and if wishes continues with booking any flight.			
10.		If option of booking flight is chosen then, system reaches to the central system for booking of the flight.		
11.			Central system receives request of flight booking from system entity. Central system uses real time synchronization steps and responds to system with updated flight status or if flight is available, then central system sends appropriate message to system to continue with payment for flight.	
12.		if flight available, system continues with payment and asks user to make payment for booking.		
13.	User receives either fail as response or to continue with payment. If payment portal opens, then user selects payment method. i.e., bank.			
14.		System connects to banks by API and asks user with all details		
15.	User enters details and pay for booking.			

16.		System requests payment status booking status from central system.		
17.			Central system receives payment and reference id of the payment, with real time syn mechanisms, flight is booked for the user. The updated information is sent to the central system	
18.		System receives appropriate message and corresponding show to the user.		
19.	User gets final status of the bookings.			

6.BOOKING HISTORY

6.1 DOCUMENTATION

The sequence diagram depicts the flow of working of history checking.

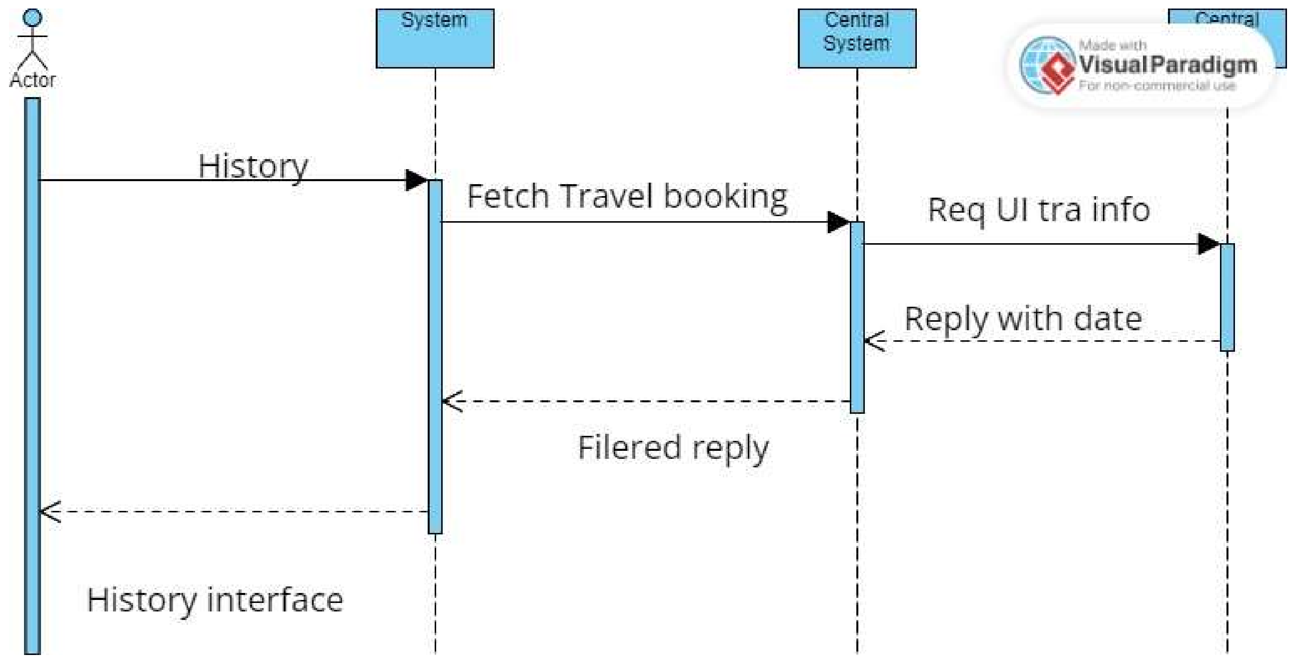


FIGURE 16: SEQUENCE FOR HISTORY

6.2 Diagram content details

- Actor
- System
- Central system
- Central database

6.3 Flow process for flight booking history.

TABLE 19: FLOW OF HISTROY

S. no	Actor	System	Central System	base
20.	Actor tabs on history icon.			
21.		System responds with requesting fetch Travel booking information form central system.		
22.			Central system receives request for user travel history. Central system then refers to the central database for user history.	
23.				The base system receives unique id for history. It fetches and returns the data to the central system.
24.			Central system receives the data. Central system then sends the data to the system.	
25.		System receives the response form the central system. It creates an UI interface and sends it to the user		
26.	User gets the history of its activity.			

7.ACTIVITY DIAGRAM FOR FLIGHT BOOKING

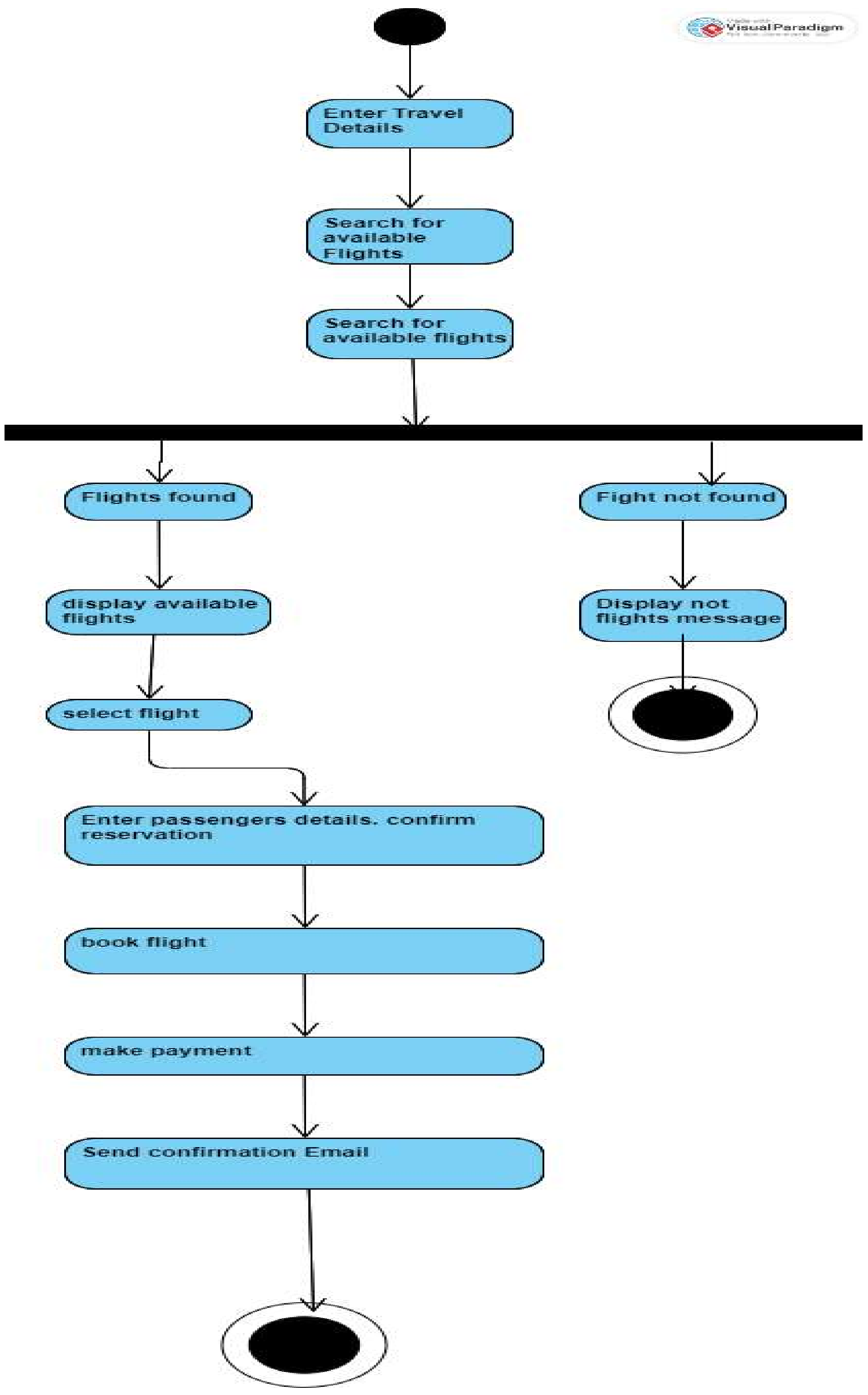

Enter Travel Details
Search for available Flights
Search for available flights
Flights found
Fight not found
display available flights
Display not flights message
select flight
Enter passengers details. confirm reservation
book flight
make payment
Send confirmation Email

FIGURE 17: ABMS ACTIVITY DIAGRAM

7.1 DESCRIPTION

TABLE 20: ACTIVITY FLOW

Step	***Transaction Form***	***Inputs***	***Outputs***
1	*Start*		
2	*Enter Travel Details*	*Destination, Date, Number of Passengers*	
3	*Search for Available Flights*	*Travel details*	*List of available flights*
4	*Display Available Flights*	*List of available flights*	
5	*Select Flight*	*Selected flight number*	
6	*Enter Passenger Details*	*Passenger information*	
7	*Confirm Reservation*	*Confirmation information*	
8	*Book Flight*	*Booking information*	*Confirmation number*
9	*Send Confirmation Email*	*Confirmation number, Email address*	*Email confirmation*
10	*Exit*		

8.STATE CHART FOR FLIGHT BOOKING

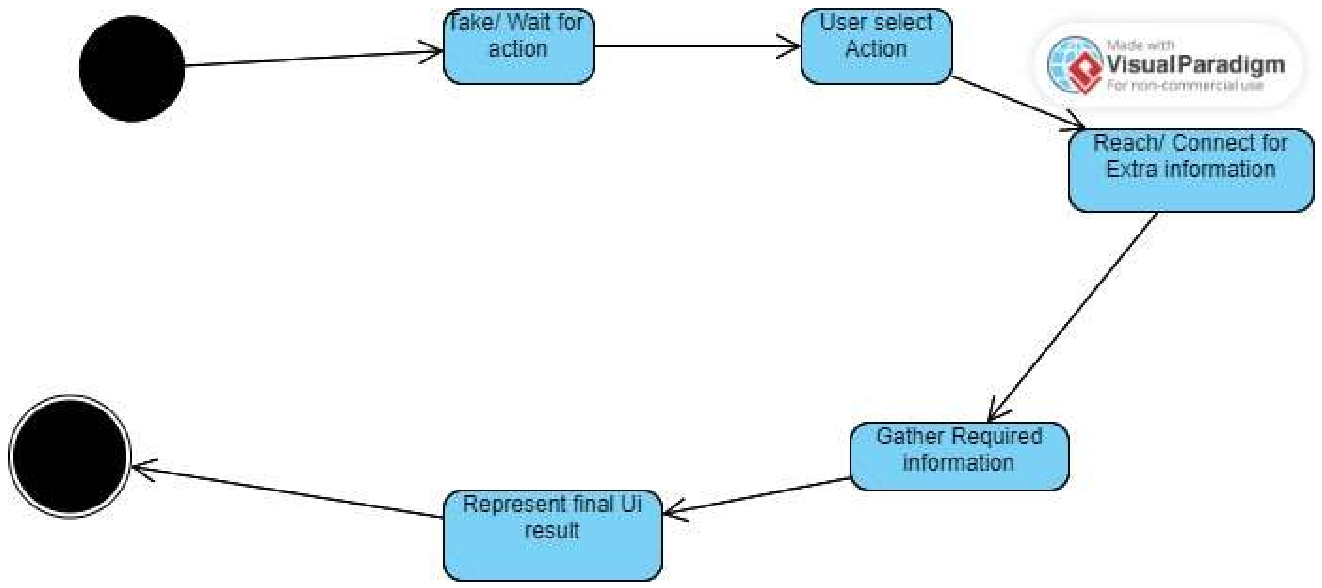

Take/ Wait for action
User select Action
Made with Visual Paradigm For non-commercial use
Reach/ Connect for Extra information
Gather Required information
Represent final Ui result

FIGURE 18: FLIHT BOOKING STATE CHART

8.1 DESCRIPTION

TABLE 21: STATE DESCRIPTION

State	**Description**
Take/Wait for Action	Initial state where the system waits for user action to initiate the reservation process. The system displays a message or a prompt to the user asking them to perform a specific action, such as clicking a button to start the reservation process.
User Select Action	The system allows the user to select an action to initiate the reservation process. Available actions may include searching for flights, selecting a flight, booking a flight, or making a payment. Once the user selects an action, the system transitions to the corresponding state for that action.
Reach/Connect for Extra Information	The system connects to external sources or APIs to gather additional information that may be required for the reservation process, such as flight schedules, seat availability, or passenger information. Once the system retrieves the required information, it transitions to the next state.
Gather Required Information, Represent Final UI Result	In this state, the system displays the final user interface (UI) to the user, where they can enter their required information to complete the reservation process. The UI may include input fields for passenger details, payment information, and other relevant information. Once the user enters all the required information, the system processes the data and displays the final result to the user.

The transitions between these states are triggered by user actions and events. For example, the transition from the "Take/Wait for Action" state to the "User Select Action" state is triggered when the user clicks a button to initiate the reservation process. Similarly, the transition from the "User Select Action" state to the "Reach/Connect for Extra Information" state is triggered when the user selects an action that requires additional information, such as selecting a specific flight. The transitions may also include actions or conditions that must be satisfied before the transition can occur, such as validating user input or checking for seat availability.

9.COMPONENT DIAGRAM FOR AIRBUS MANAGEMENT SYSTEM

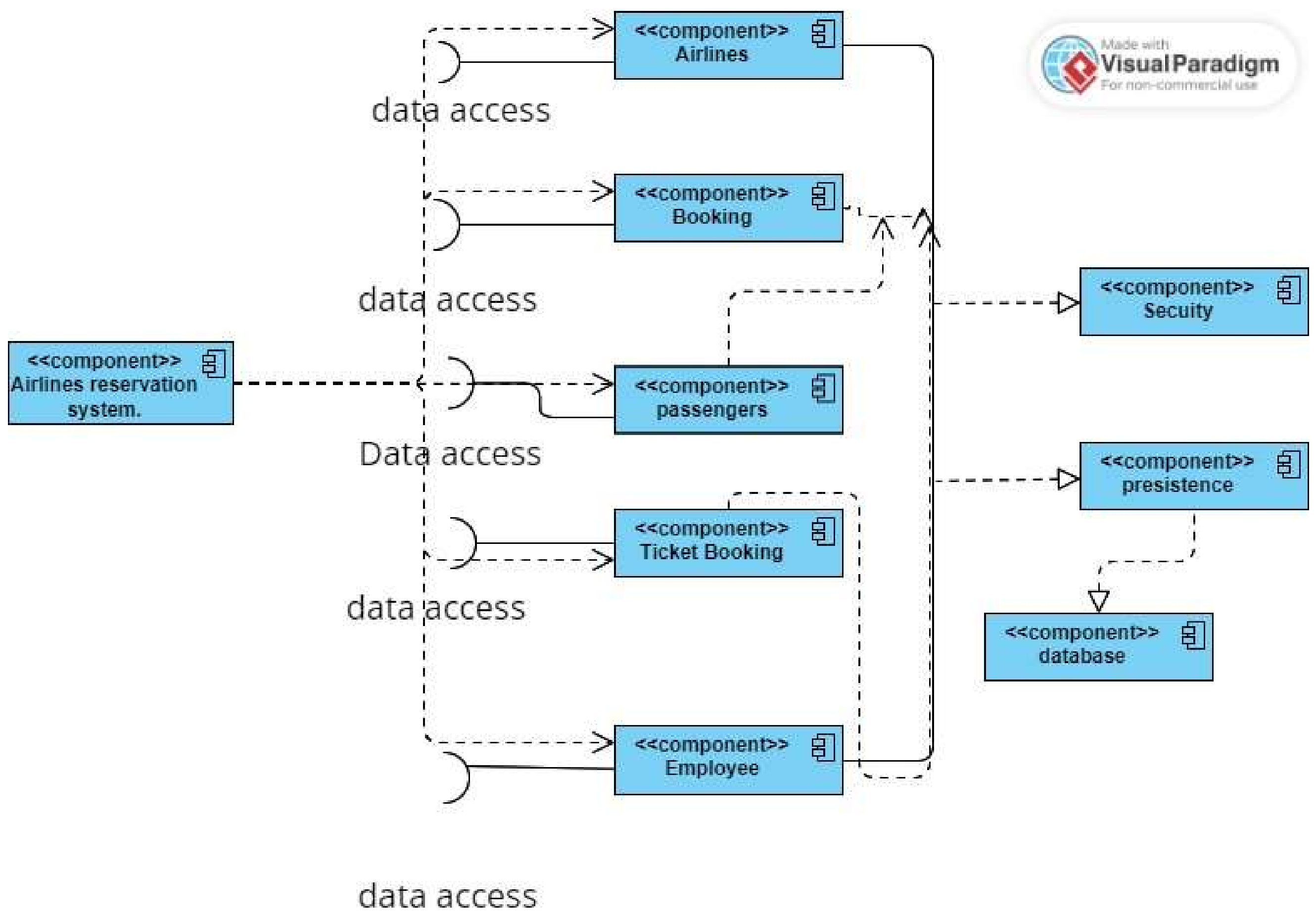


FIGURE 19: ABMS COMPONENT DIAGRAM

9.1 Description

At the top of the diagram, the "Passenger UI" component represents the user interface of the system, which includes several forms for searching, reserving, and managing flights. The "Web Server" component provides the backend of the system and includes a "Web API" for communicating with the client.

The "Booking Management" and "Flight Management" components represent the core functionality of the system. The "Booking Service" component manages the booking of flights, including payment processing, confirmation emails, and notifications. The "Payment Gateway" component handles secure payment processing. The "Confirmation Email" component sends confirmation emails to customers. The "Notification Service" component sends notifications to customers about their bookings. The "Customer Service" component provides support for customers. The "Reservation Information" component stores reservation data.

The "Flight Database" component stores flight data, which is accessed by the "Flight Database Interface" component. The "Flight Schedule Service" component manages flight schedules. The "Flight Search Service" component searches for available flights. The "Flight Reservation Service" component manages flight reservations. The "Aircraft Management" component manages aircraft information. The "Crew Management" component manages crew information.

Overall, the component diagram shows how the different parts of the airline reservation system work together to provide a comprehensive booking and management solution for customers and airline staff.

10.DEPLOYMENT DIAGRAM FOR AIRBUS MANAGEMENT SYSTEM

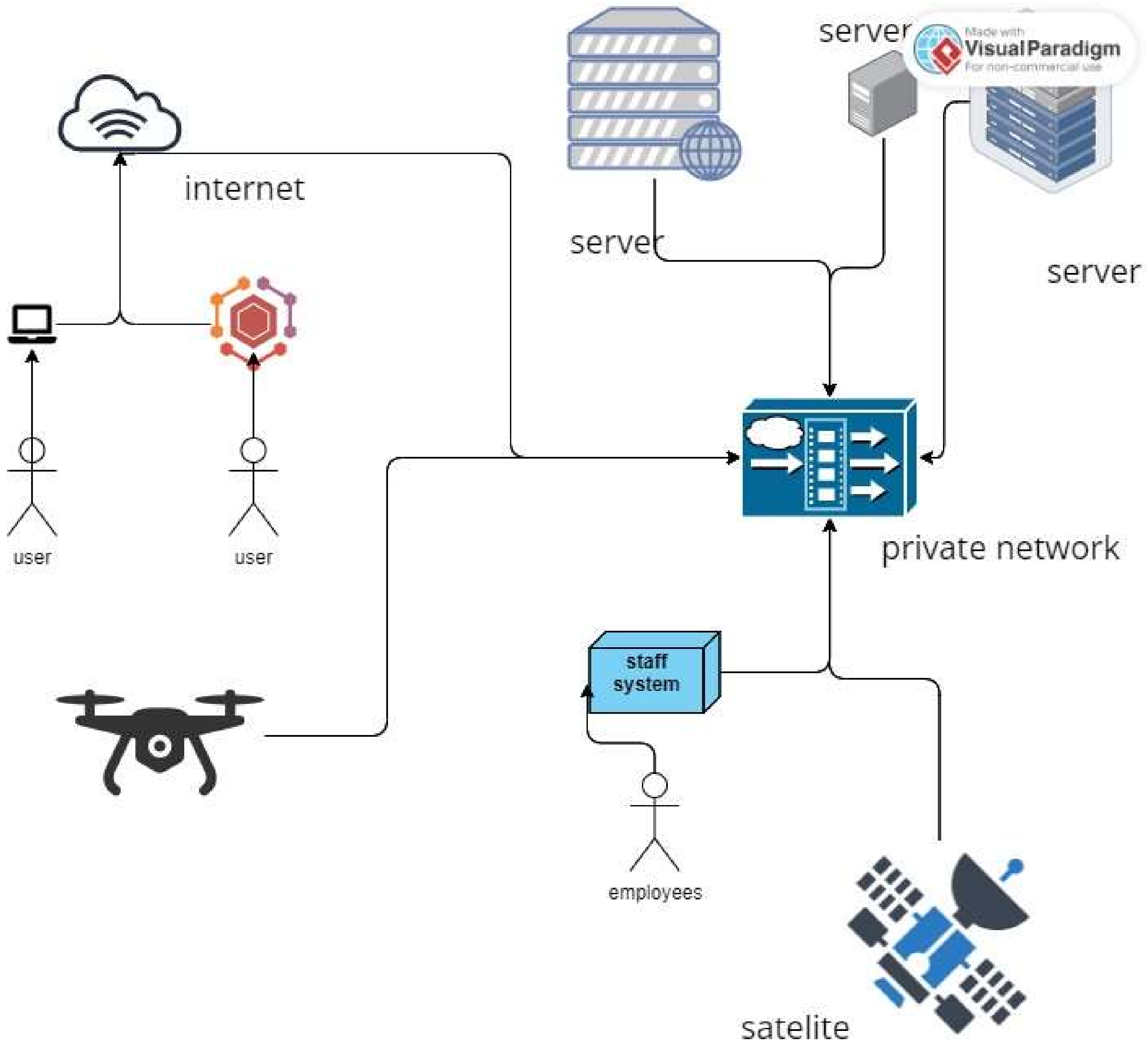


FIGURE 20: ABMS DEPLOYMENT DIAGRAM

10.1 DESCRIPTION

The deployment diagram consists of nodes that represent physical or virtual machines such as servers, databases, web servers, and client machines. These nodes are interconnected by communication paths, such as network connections, to indicate how the components of the system communicate with each other.

The main components of the airline reservation system include the web application, the database server, and the payment gateway. The web application is deployed on a web server and is accessed by users through a web browser. The web server communicates with the application server, which processes user requests and interacts with the database server to retrieve and store data.

The database server stores all the data related to the airline reservation system, such as flight schedules, ticket prices, customer information, and payment details. The payment gateway is used to process online payments for reservations and is typically deployed on a separate server.

In addition to these components, the deployment diagram may also include other hardware or software components, such as load balancers, firewalls, and caching servers, to ensure high availability, scalability, and security of the system.

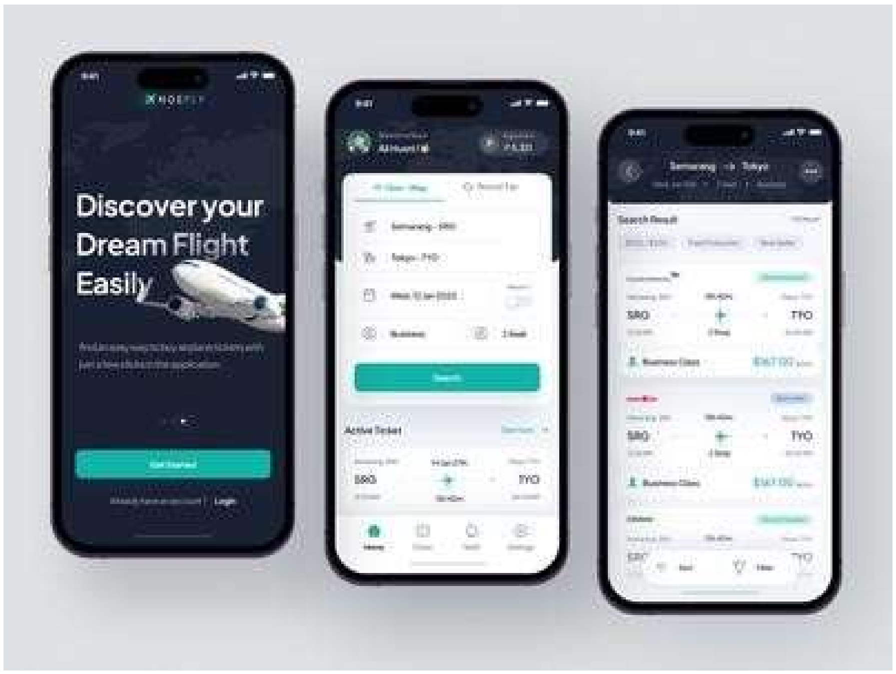
Overall, the deployment diagram for an airline reservation system provides a high-level view of the system architecture, showing how the system components are distributed across different nodes and how they interact with each other to provide a seamless and efficient user experience.

11.TEST CASE FOR AIRBUS MANAGEMENT SYSTEM

TABLE 22: TEST CASES

Test Case Description	**Expected Result**
User enters valid flight information and clicks search button	System displays a list of available flights matching the entered criteria
User enters invalid flight information and clicks search button	System displays an error message indicating that the entered information is invalid
User selects a flight and clicks on book button	System prompts user to enter passenger details and payment information
User enters valid passenger and payment information and clicks on confirm button	System displays a confirmation message with the booking details
User enters invalid passenger or payment information and clicks on confirm button	System displays an error message indicating that the entered information is invalid
User cancels a previously booked reservation	System prompts user to confirm the cancellation
User confirms cancellation of reservation	System displays a message confirming the cancellation
User tries to cancel a reservation that is already cancelled	System displays an error message indicating that the reservation is already cancelled
User tries to book a flight that is already fully booked	System displays an error message indicating that the flight is fully booked
User tries to book a flight that is unavailable due to technical issues	System displays an error message indicating that the flight is unavailable due to technical issues
User tries to book a flight without entering all required information	System displays an error message indicating that some information is missing

12.AIRBUS MANAGEMENT SYSTEM PRIVEW



13.CONCLUSION

Airbus management system is user-friendly, secure, reliable, and scalable. It allows users to easily search for flights, book and manage reservations, and provide a seamless experience from start to finish. Additionally, it has robust security measures in place to protect users' personal and financial information, and it is designed to handle a large volume of traffic without experiencing downtime or performance issues. Thus, airbus management system provides significant benefits to airlines, travel agencies, and travellers, including increased efficiency, improved customer service, and the ability to manage bookings in real-time. As the travel industry continues to evolve, the importance of airline reservation system software is likely to grow, with new features and functionality added to meet the changing needs of travellers and travel businesses.

TADA

DESCRIPTION

A vehicle rental system is a complex system that involves multiple components and processes. It allows customers to rent vehicles for a certain period of time, either for personal or business use. In recent years, vehicle rental systems have evolved to offer additional features such as driver drop and pickup, as well as opportunities for individuals to become drivers themselves. In this article, we will explore the different aspects of a vehicle rental system, from the rental facility to the application program interface (API) interacting with the user and back-end.

Vehicle Rental Facility

The vehicle rental facility is the first component of a vehicle rental system. It is responsible for providing customers with a variety of vehicles to choose from, ensuring that they are properly maintained, and offering various rental options such as daily or weekly rentals. The rental facility is typically staffed with customer service representatives who assist customers in selecting a vehicle and making a reservation.The rental facility also plays a critical role in the driver drop and pickup feature. In this case, the facility will also provide drivers who are properly licensed and trained to operate the rental vehicle. Drivers can be hired by the rental facility or can be contracted through a third-party service. The facility will also manage the scheduling and logistics of the driver pickup and drop-off, ensuring that drivers are assigned to the appropriate customers at the right time.

Application Program Interface (API)

The API is the interface between the vehicle rental system and the user. It is responsible for managing customer accounts, reservations, and payments. The API can be accessed through a website or a mobile application, which provides customers with a user-friendly interface to search for available vehicles and make bookings.The API is also responsible for the driver drop and pickup feature. Customers can select the driver option when making a reservation, and the API will automatically assign a driver to the reservation based on availability and location. The API can also track the location of the driver in real-time, providing customers with accurate arrival times and allowing them to communicate with the driver during the ride.

Back-End

The back-end is the behind-the-scenes component of the vehicle rental system. It is responsible for managing the database, ensuring that all customer information is secure, and processing payments. The back-end also manages the logistics of the driver drop and pickup feature, assigning drivers to reservations and tracking their location in real-time.The back-end also provides the functionality for individuals to become drivers themselves. This feature allows individuals who meet certain requirements to apply to become a driver for the rental facility. The back-end will manage the application process, ensuring that all applicants meet the necessary criteria such as having a valid driver's license and a clean driving record. Once approved, the back-end will assign the new driver to reservations and manage their schedules.

Driver Drop and Pickup

The driver drops and pickup feature is a relatively new addition to vehicle rental systems. It allows customers to rent a vehicle with a driver who will pick them up and drop them off at their desired location. This feature is particularly useful for customers who do not have access to a vehicle or do not want to drive themselves. When a customer selects the driver drop and pickup option, the rental facility or the API will automatically assign a driver to the reservation based on availability and location. The driver will then arrive at the customer's desired location at the scheduled time, assist the customer with their luggage, and provide a safe and comfortable ride to their destination. During the ride, the customer can communicate with the driver through the rental facility or API, providing real-time feedback on the quality of the service. Once the ride is complete, the rental facility or API will process the payment and provide the customer with a receipt.

Become a Driver

The become a driver feature is an innovative addition to vehicle rental systems. It allows individuals who meet certain requirements to apply to become a driver for the rental facility. This feature is particularly useful for individuals who have a valid driver's license and enjoy driving, but do not have their own vehicle or are looking for a flexible work schedule. To become a driver, individuals must first apply through the rental facility's website or mobile application. The back-end will then manage the application process, which typically involves verifying the applicant's driver's license, checking their driving record, and conducting a background check. Once approved, the new driver will be added to the rental facility's pool of available drivers. The rental facility or API will then assign the driver to reservations based on their availability and location. The driver will then pick up the rental vehicle at the facility and proceed to the customer's desired location. During the ride, the driver must ensure the safety and comfort of the customer, as well as provide excellent customer service. The rental facility or API will provide the driver with detailed instructions on the pickup and drop-off locations, as well as any specific requests from the customer.

Once the ride is complete, the rental facility or API will process the payment and provide the driver with a commission or fee for their services. The driver can then return the rental vehicle to the facility and wait for their next assignment.

The back-end is responsible for managing the database, ensuring that all customer information is secure, and processing payments. It also manages the logistics of the driver drop and pickup feature and allows individuals to become drivers themselves.

The driver drops and pickup feature is a relatively new addition to vehicle rental systems, allowing customers to rent a vehicle with a driver who will pick them up and drop them off at their desired location. The become a driver feature allows individuals to apply to become a driver for the rental facility and provides them with flexible work options.

Overall, the vehicle rental system provides a convenient and flexible solution for individuals and businesses who need access to vehicles for personal or business use, as well as for those who are looking for additional income opportunities.

UML DIAGRAMS OF TADA

Use Case Diagram

The above is a use case diagram representing our vehicle rental system, it is a graphical representation of the actors (users, systems, or other entities) and the interactions they have with a system to achieve a specific goal. It is a high-level view of the system and helps to identify the key features and functionalities that the system should provide. Use case diagrams typically include actors, use cases, and relationships between them. Actors are the users or other systems that interact with the system, while use cases are the specific actions or tasks that the system can perform. Relationships between actors and use cases are shown as lines connecting them, and can represent different types of interactions such as "uses," "extend," or "includes."

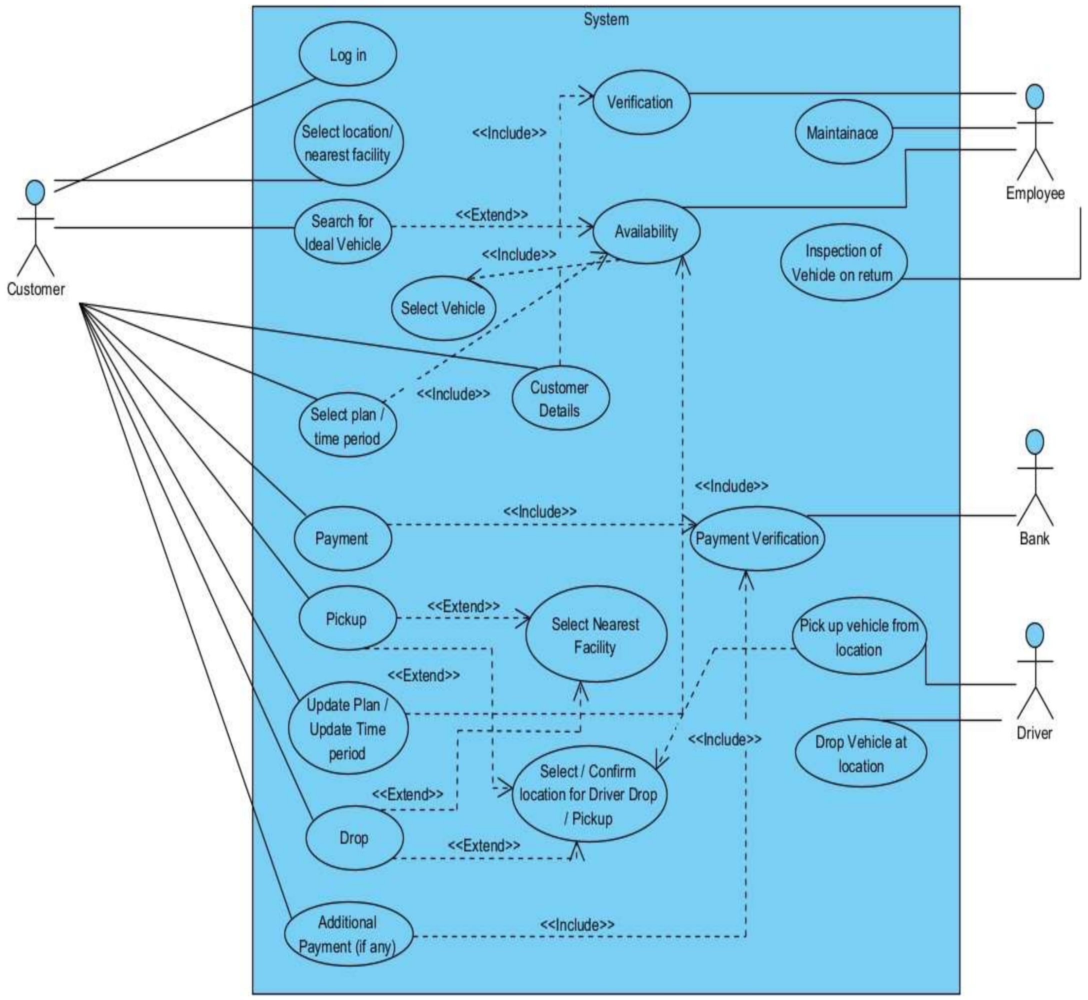


Fig-34: Use case diagram for TADA

In the above Use case Diagram, the actors are:

- **Customer**

This actor represents the people who rent vehicles from the rental company. The key use cases for this actor would include searching for available vehicles, making a reservation, picking up a vehicle, returning a vehicle, and paying for a rental. In addition customer could select options from Driver drop/ pick up, here a driver from the company is assigned to deliver the vehicle to the client's location or pick up the car from the location where the ride ends.

- **Employee**

This actor represents the employees who work for the rental company, such as rental agents and maintenance staff. The key use cases for this actor would include managing vehicle inventory, processing rental reservations, inspecting returned vehicles, and handling customer service inquiries.

- **Bank**

This use case represents the process of handling rental payments, including calculating rental fees, accepting payment, and issuing refunds if necessary.

- **Driver**

This use case represents the process of assigning a driver to the vehicle pickup or drop location.

Overall, the above diagram provides a visual overview of the key features and interactions of a system and can be a useful tool for communication and collaboration among stakeholders during the software development process.

Use case diagrams typically include actors, use cases, and relationships between them. Actors are the users or other systems that interact with the system, while use cases are the specific actions or tasks that the system can perform.

Relationships between actors and use cases are shown as lines connecting them, and can represent different types of interactions such as “uses,” “extend," or "includes.

Class Diagram

The above diagram is a graphical representation of the classes, objects, and relationships involved in a system. It is a static view of the system and shows the various entities in the system, their attributes and methods, and the relationships between them.

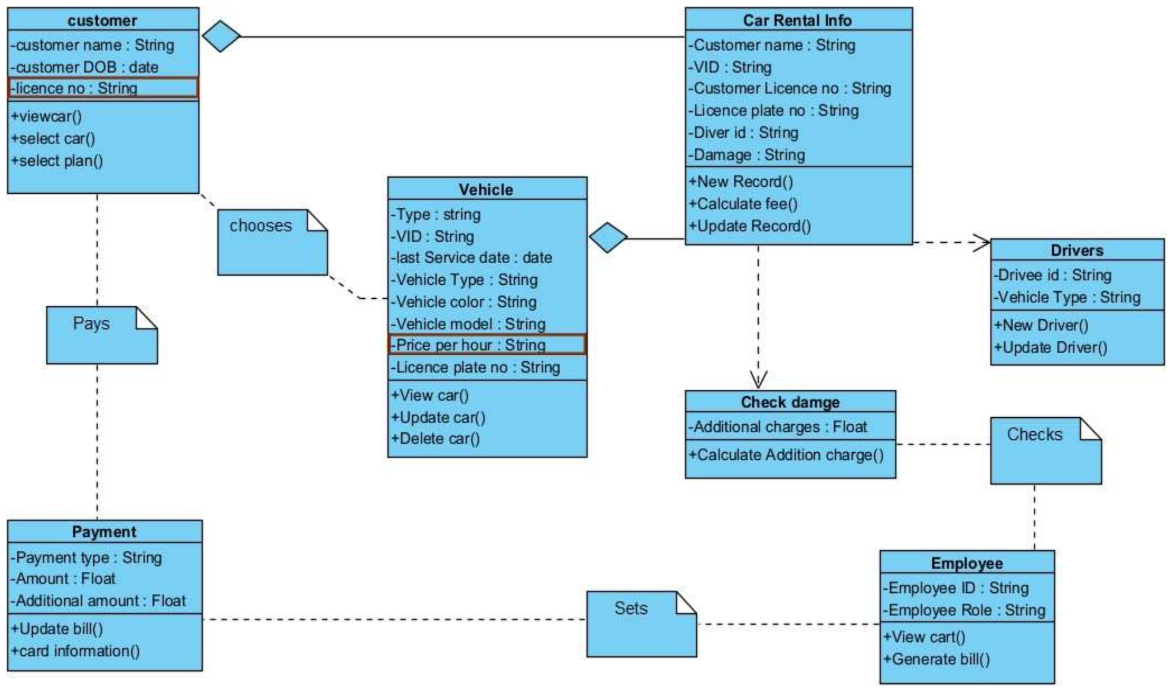


Fig-35: Class Diagram for TADA

In a class diagram, classes are represented as rectangles, with the name of the class at the top and the attributes and methods listed below.

Relationships between classes are shown as lines connecting them and can represent different types of relationships such as association, aggregation, or inheritance. Attributes are the data fields of the class, while methods are the behaviors or operations that the class can perform. So, the classes in the above diagram are:

Vehicle

This class represents the various types of vehicles available for rent, such as cars, trucks, vans, and motorcycles. It includes attributes such as color, model, year, License plate number, Rental price etc.

Customer

This class represents the people who rent vehicles from the rental company. It includes attributes such as name, address, License number, contact information etc.

Employee

This class represents the employees who work for the rental company, such as rental agents and maintenance staff. It includes attributes such as name, job title, contact information etc.

Car Rental Information

This class represents a rental transaction, which would typically involve a customer renting a vehicle for a specified period. It includes attributes such as rental date, return date, rental fee etc.

Check damage

This class makes sure the employee checks for any damage to the vehicle upon its return. It includes attributes such as additional charges etc.

Driver

This class represents the Driver who is assigned if needed for pick up / drop. It includes attributes such as driver id, Vehicle type etc.

Payment

This class represents the payment methods used to pay for rentals, such as credit cards or cash. It includes attributes such as payment type, additional charges, payment amount etc.

Overall, the above diagram provides a basic overview of the key entities and relationships in a vehicle rental system and could serve as a starting point for further development and refinement.

Sequence Diagram

The above diagram shows the interactions between objects or components in a system over time. It is a dynamic view of the system and illustrates the flow of messages between objects or components. Sequence diagrams are often used in software development to help teams understand and design the behavior of a system, particularly in the context of use cases or scenarios.

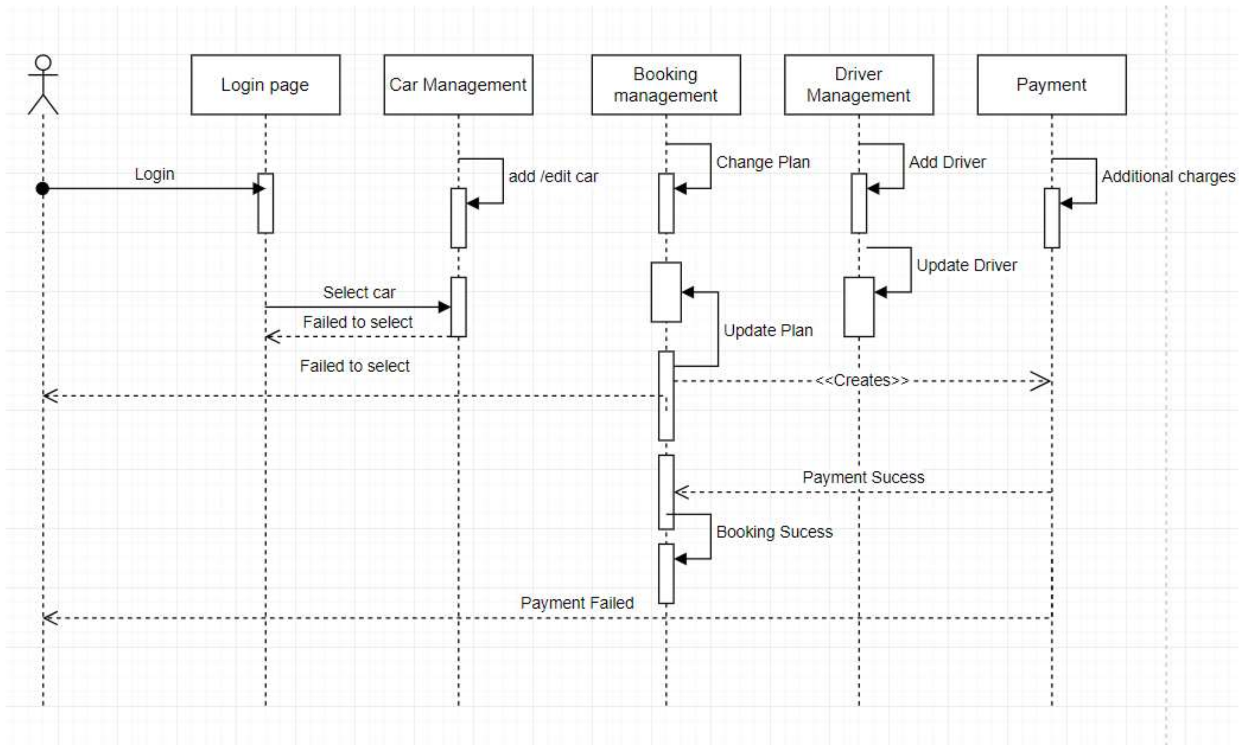


Fig-36: Sequence diagram for TADA

- **Customer searches for available vehicles**

The sequence begins with the customer searching for available vehicles. The customer sends a request message to the vehicle inventory system, which responds with a list of available vehicles that match the customer's search criteria.

- **Customer makes a reservation**

The customer selects a vehicle from the list and sends a reservation request message to the rental reservation system. The rental reservation system confirms the reservation by sending a reservation confirmation message back to the customer.

- **Customer picks up the vehicle**

The customer arrives at the rental location to pick up the reserved vehicle. The rental agent confirms the reservation by checking the customer's information and sends a message to the vehicle inspection system to prepare the vehicle for rental.

- **(optional) Driver drop and pick up**

The customer is given a choice of picking up the vehicle from the facility or a driver could be assigned to drop the vehicle at the location the same feature is also available for dropping.

- **Vehicle inspection and handover**

The vehicle inspection system checks the vehicle for any damage or maintenance issues and sends an inspection report to the rental agent. The rental agent confirms the inspection report and hands over the vehicle to the customer.

- **Customer returns the vehicle**

The customer uses the rental vehicle and returns it to the rental location at the end of the rental period. The rental agent inspects the vehicle for any damage or maintenance issues and sends an inspection report to the vehicle inspection system.

- **Payment and finalization**

The rental agent calculates the rental fee based on the rental period and any additional charges such as fuel or damage fees. The customer pays the rental fee, and the rental transaction is finalized.

Overall, this diagram provides a visual representation of the interactions between various components of a vehicle rental system, including the customer, rental reservation system, vehicle inventory system, rental agent, and vehicle inspection system. It shows the flow of messages and events that occur during the rental process and can aid in the design and implementation of software for a vehicle rental system.

Activity Diagram

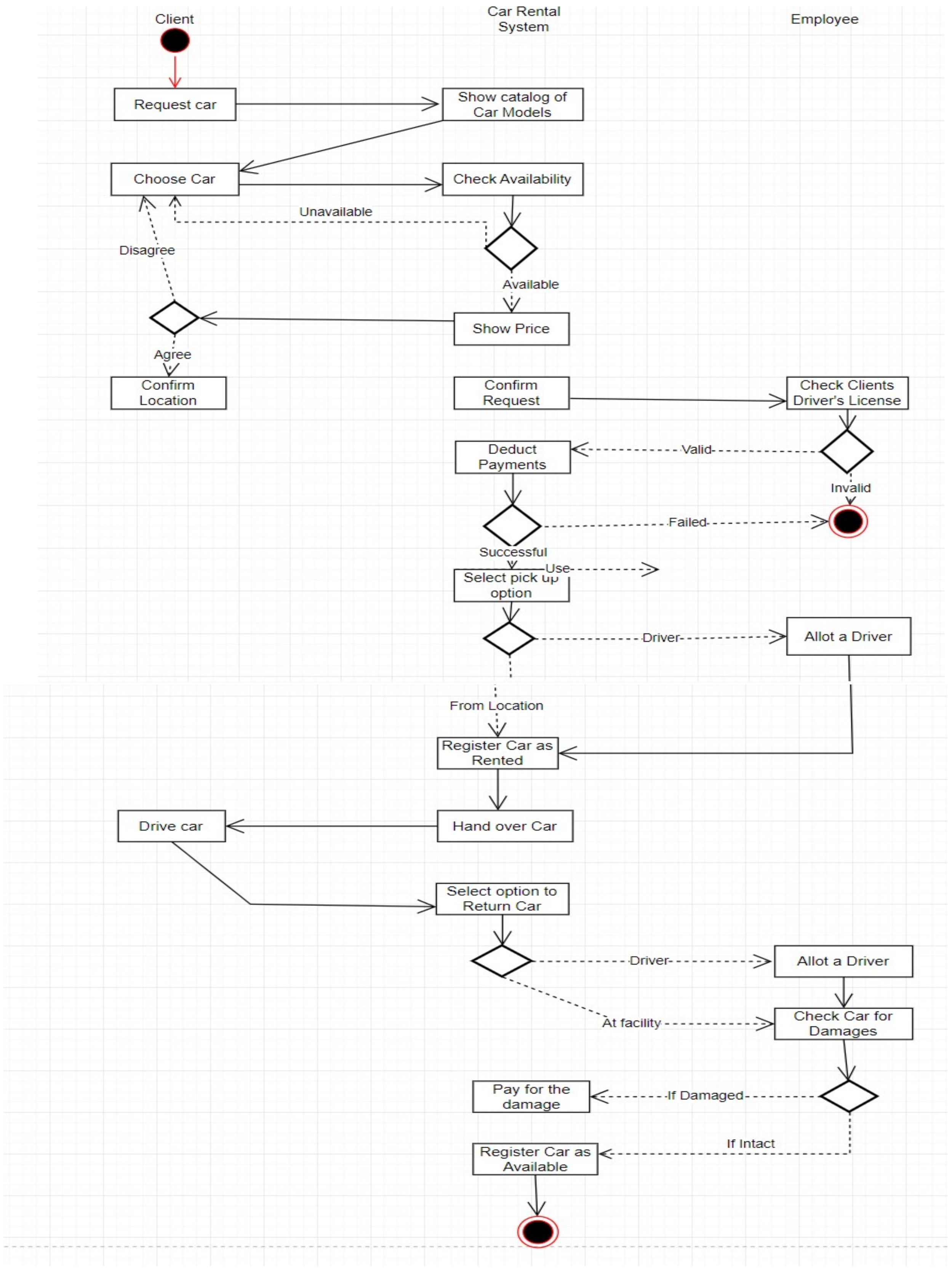


Fig-37: Activity diagram for TADA

The diagram begins with the user initiating the process of renting a vehicle. The first activity might be to search for available vehicles based on certain criteria, such as the type of vehicle, location, and rental period.

Once the user has found a suitable vehicle, the next activity could be to reserve the vehicle and provide payment information. This would involve verifying availability, calculating the rental cost, and processing the payment.

After the reservation is confirmed, the user would then proceed to the next activity of picking up the vehicle at the rental location. This activity would involve checking the user's identification, verifying payment, and providing the user with the keys and rental agreement.

There the user will get an option for selecting driver pickup or drop option where the user can opt for a driver to drop the vehicle at the location or pick up from the drop point.

The user would then proceed to the next activity of inspecting the vehicle for any damage and confirming that it is in good condition. This would be followed by the activity of driving the vehicle for the rental period.

At the end of the rental period, the user would return the vehicle to the rental location and complete the final activity of checking the vehicle for any damage and settling any additional charges, such as fuel costs or late fees.

Overall, this diagram outlines the steps involved in the rental process, from searching for available vehicles to returning the vehicle at the end of the rental period. It would provide a clear and visual representation of the various activities involved, helping to ensure a smooth and efficient rental process.

Deployment Diagram

The above diagram for a vehicle rental system depicts the physical architecture of the system and how the various software and hardware components are deployed and interconnected.

At the top level, the diagram would include nodes that represent the physical components of the system, such as servers, workstations, routers, and switches. Each node would be connected by network links that represent the communication channels between them.Within each node, the diagram would include components that represent the various software applications and services that are deployed on that node. For example, there might be components for the rental application, the reservation system, the payment gateway, and the database server.

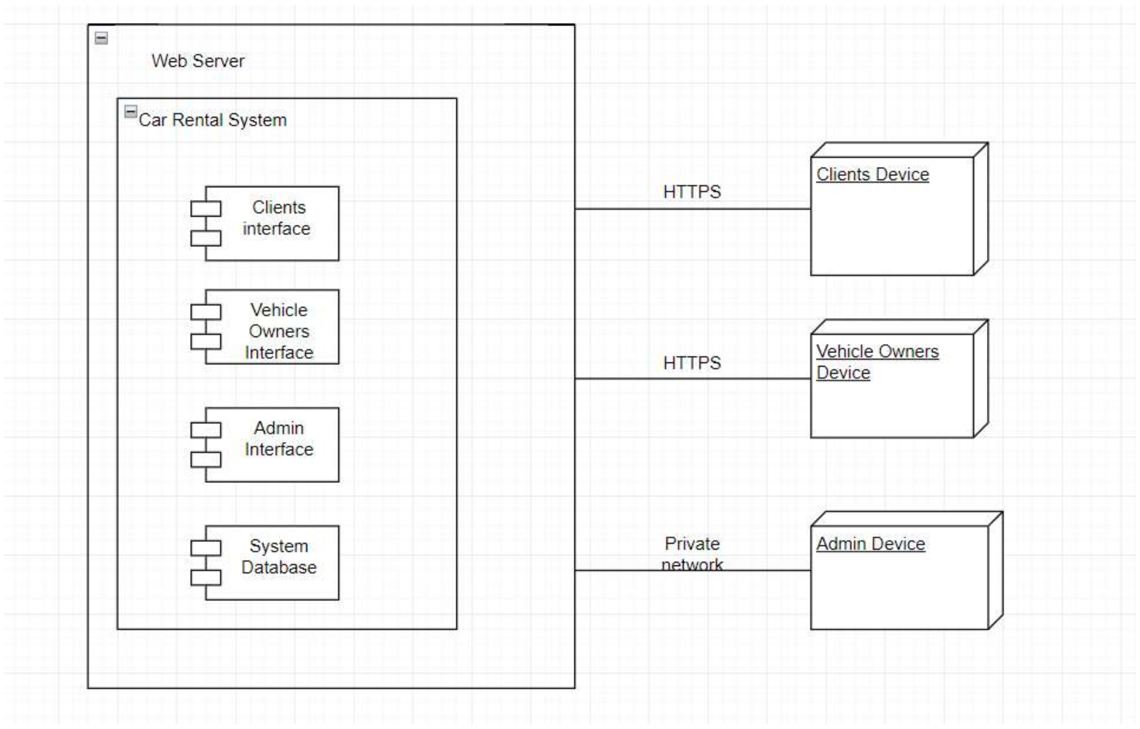


Fig-38: Deployment Diagram for TADA

The deployment diagram would also depict the relationship between the various components and nodes. For example, the rental application might be deployed on a server node, which is connected to the reservation system component deployed on another server node. The rental application might also communicate with the payment gateway component, which is deployed on a separate node that is connected to the rental application node via a network link. It would help developers and system administrators to understand the deployment architecture and troubleshoot issues related to system performance or connectivity.

Component Diagram

The above diagram for a vehicle rental system would depict the organization of the software components that make up the system and how they interact with each other.

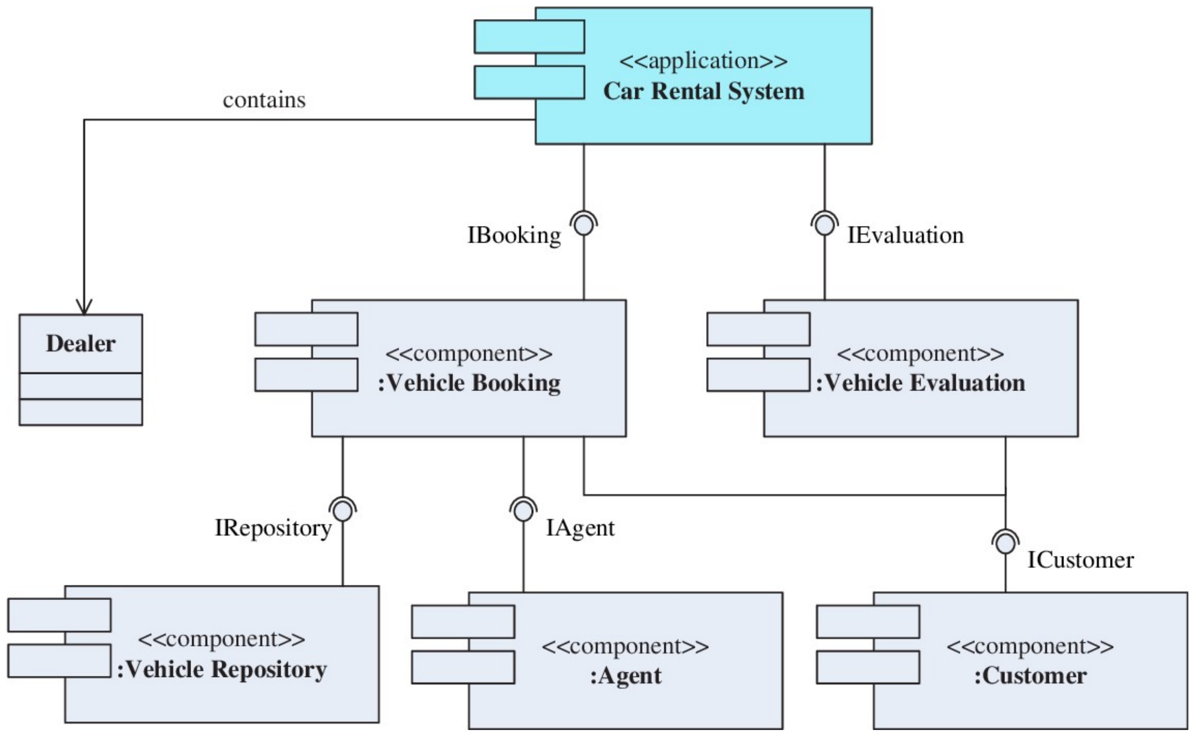


Fig-39: Component diagram for TADA

At the top level, the diagram includes components that represent the major software applications and services that make up the system. For example, there might be components for the rental application, the reservation system, the payment gateway, and the database server.

Within each component, the diagram would include smaller sub-components that represent the individual modules or functions of that component. For example, within the rental application component, there might be sub-components for customer registration, vehicle selection, payment processing, and booking confirmation.

The component diagram would also depict the interfaces and dependencies between the various components and sub-components. For example, the rental application might depend on the reservation system component to provide real-time availability and pricing information for vehicles. The payment processing sub-component might depend on the payment gateway component to securely process customer payments.

The component diagram provides a visual representation of the software architecture of the vehicle rental system, showing how the various components and sub-components are organized and how they interact with each other. It would help developers to understand the software structure and design of the system, and to make decisions about how to implement new features or modify existing ones.

State Chart Diagram

The above diagram is a visual representation of the various states that an object or system can go through during its lifecycle. It typically depicts the states, events, and transitions that occur within a system. The diagram will also show the events that trigger transitions between these states.

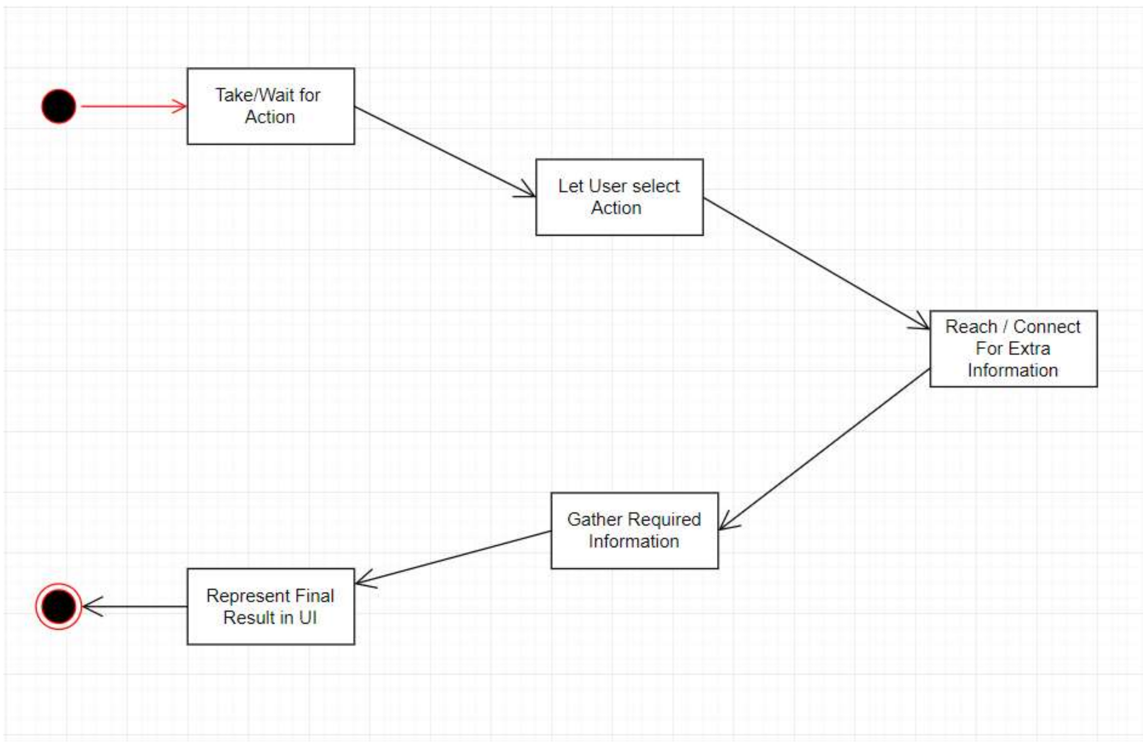

Take/Wait for Action
Let User select Action
Reach / Connect For Extra Information
Gather Required Information
Represent Final Result in UI

Fig-40:State chart for TADA

Overall, this diagram for a vehicle rental system would provide a clear and concise view of the rental transaction lifecycle, helping to identify potential issues and areas for improvement in the system.

TEST CASES

TEST CASE	SUCCESS	FAILED
LOGIN	If the Log in is successful, the user is directed to the next page to give location.	If Log in has failed, the user is redirected back to login page again. In case of multiple failed attempts on a single account, the account is temporarily deactivated.
LOCATION DETECTION	If the Location given by the user is correct and facilities are available within the range, then direct to next page.	If the location is wrong or no facility is available nearby then, thank the user for opting us and redirect to login page.
VEHICLE SELECTION	Shows the available vehicles	In case of a scenario where vehicles are not available show vehicles are not available redirect to another nearest facility
VEHICLE CONFIRMATION	Upon selection of vehicle, it is further inspected and handed over to the customer. The vehicle is then registered as rented	Nil
DRIVE AVAILABILTY	Allot a driver to drop / pick up the vehicle to/from the customer	driver not available alert customer.
DRIVE CONFIRMATION	Get confirmation from the driver for delivery and give the correct drop location provided by the user	Wrong location given to the driver resulting delayed delivery.
LICENSE VERIFICATION	Checks whether the user having permit to drive the vehicle.	If license is not verified then customer is redirected out and vehicle is not rented
BILL GENRATION	Get the fare details of the booked vehicle	Upon some error the bill generated has wrong amount.
PAYMENT	Transaction of the payment will be done, if successful vehicle is handed over to the customer.	If payment is failed customer is given three more tries to pay or else they are directed to startup page
VEHICLE DAMAGE CHECK	If no damages are found, register the vehicle as available to the next customers.	If damages are found extra charges based on the damages are added to the customers bill

Table no. 22

WEBSITE PREVIEW

ON START UP

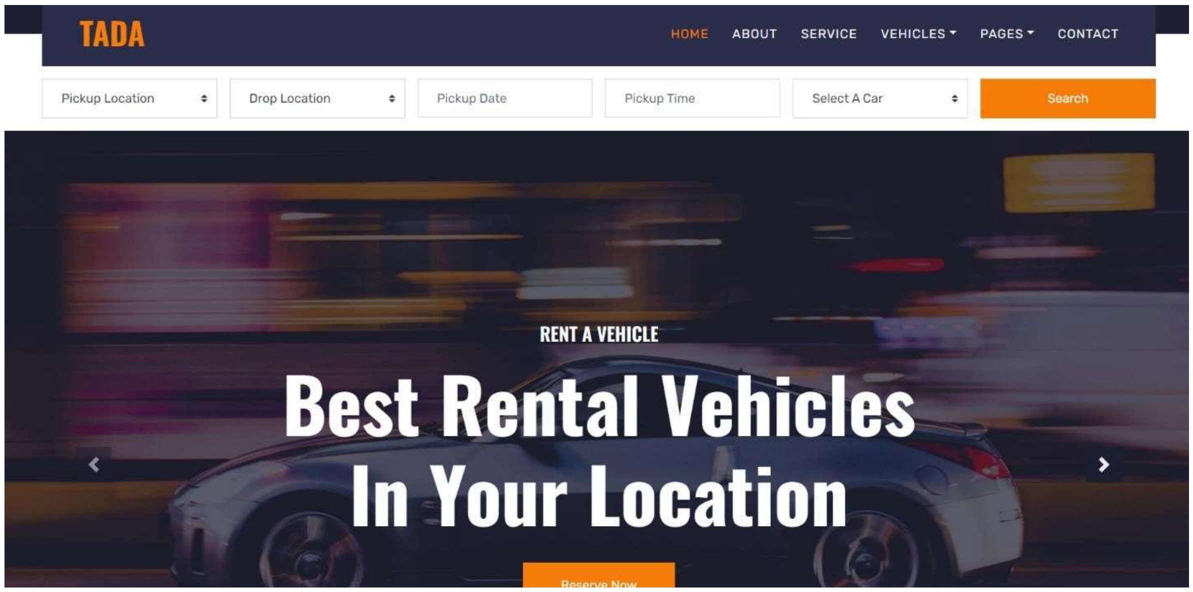
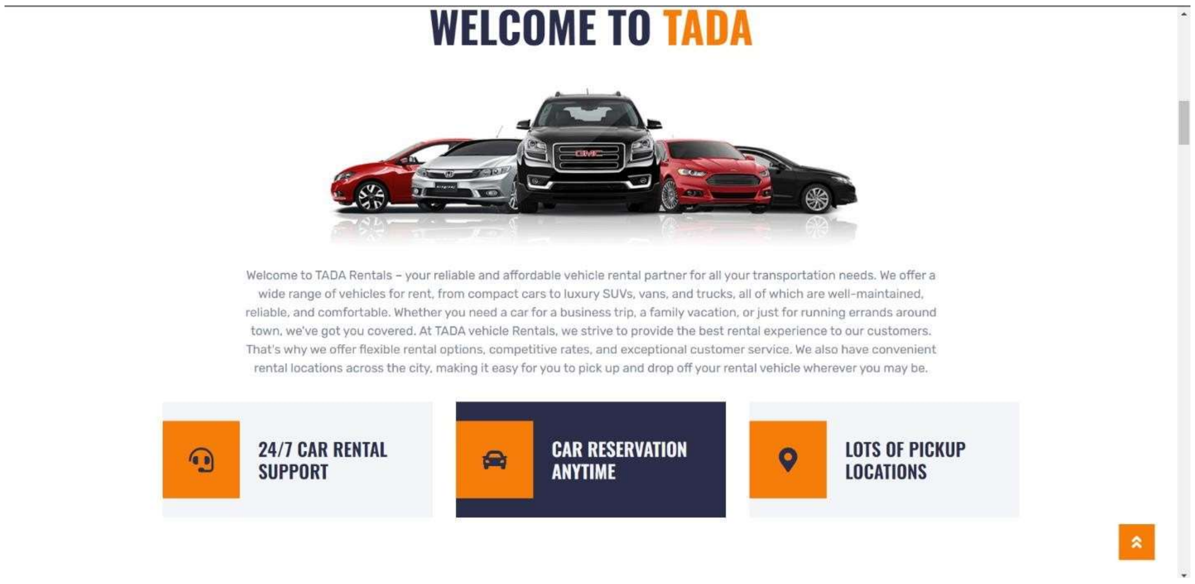


Fig-41: Startup page

INTRODUCTORY PAGE



WELCOME TO TADA

Welcome to TADA Rentals – your reliable and affordable vehicle rental partner for all your transportation needs. We offer a wide range of vehicles for rent, from compact cars to luxury SUVs, vans, and trucks, all of which are well-maintained, reliable, and comfortable. Whether you need a car for a business trip, a family vacation, or just for running errands around town, we've got you covered. At TADA vehicle Rentals, we strive to provide the best rental experience to our customers. That's why we offer flexible rental options, competitive rates, and exceptional customer service. We also have convenient rental locations across the city, making it easy for you to pick up and drop off your rental vehicle wherever you may be.

Fig-42: Introductory page

PREFENCES PAGE

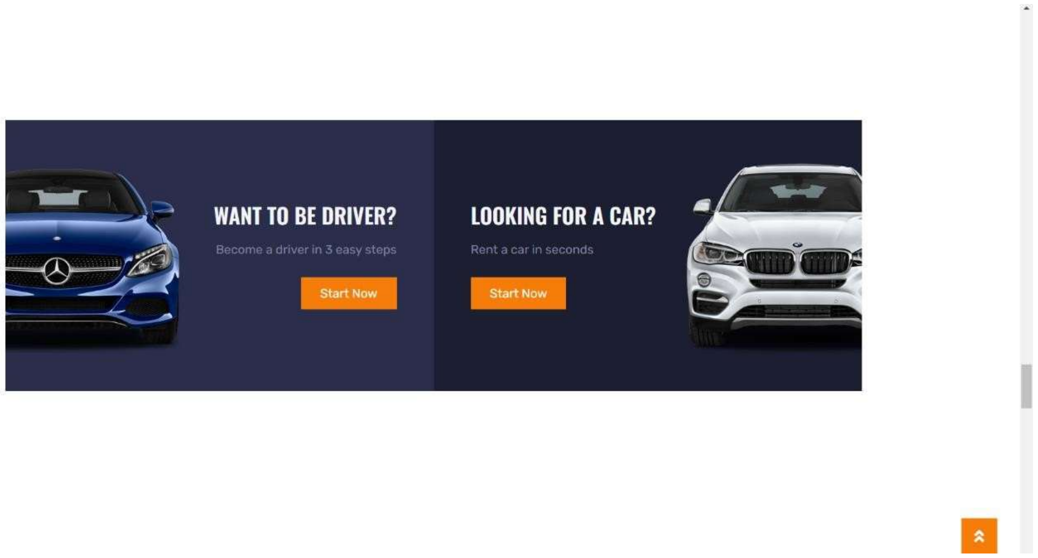


Fig-43: Preferences Page

VEHICLE SELECTION PAGE

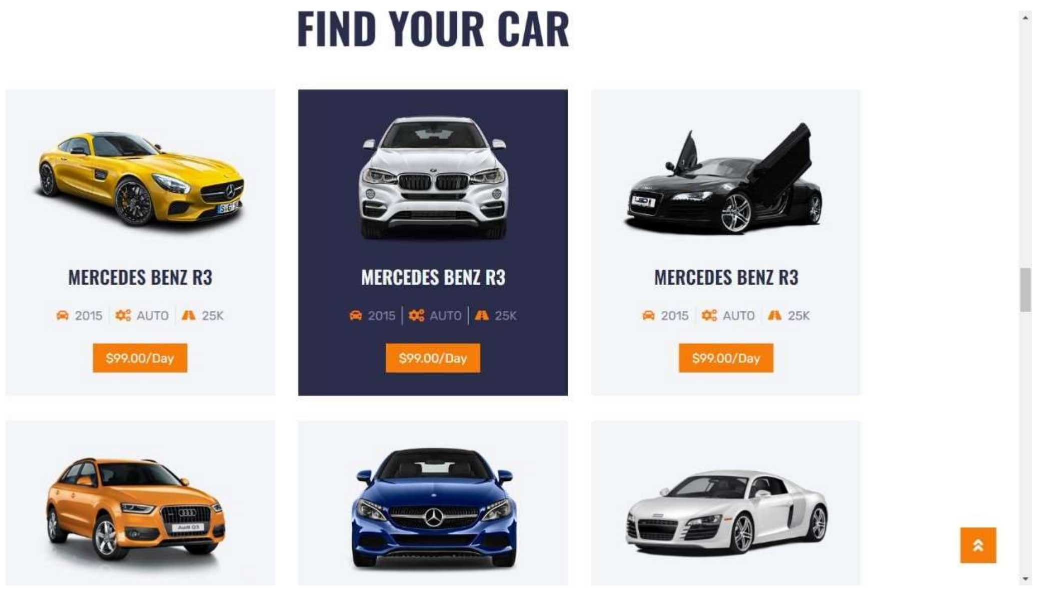
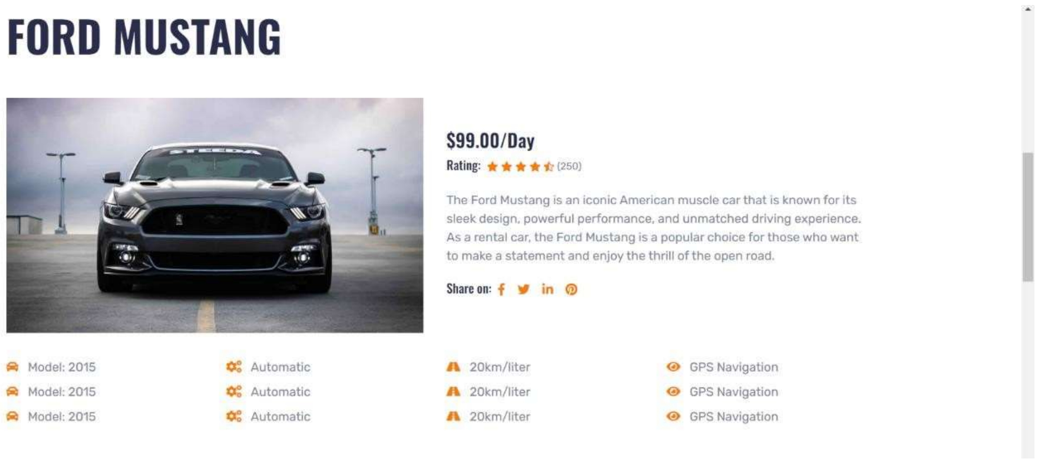


Fig-44: Vehicle Selection Page

VEHICLE DESCRIPTION PAGE



FORD MUSTANG

$99.00/Day

Rating: ★★★★☆ (250)

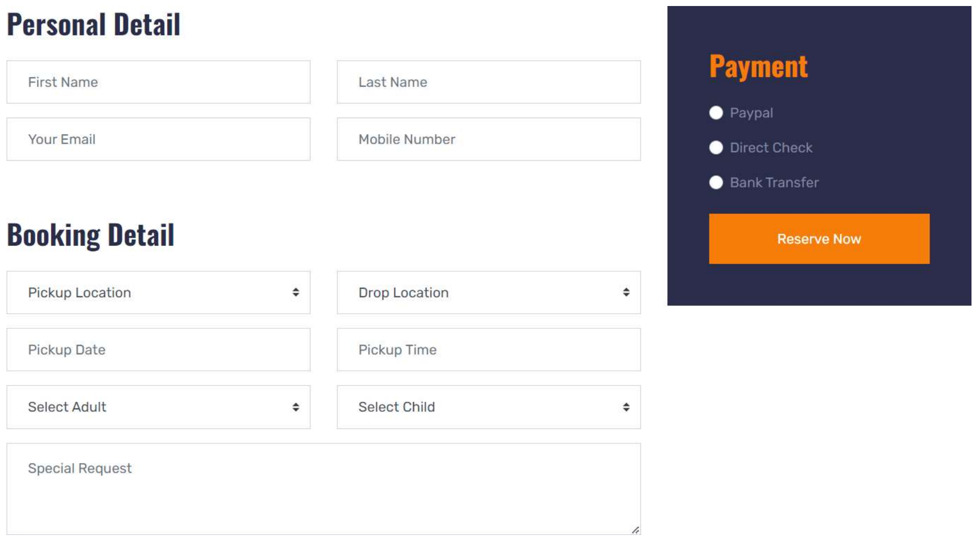
The Ford Mustang is an iconic American muscle car that is known for its sleek design, powerful performance, and unmatched driving experience. As a rental car, the Ford Mustang is a popular choice for those who want to make a statement and enjoy the thrill of the open road.

Share on:

Model: 2015	Automatic	20km/liter	GPS Navigation
Model: 2015	Automatic	20km/liter	GPS Navigation
Model: 2015	Automatic	20km/liter	GPS Navigation

Fig-45: Vehicle Description Page

USER/BOOKING DETAILS AND PAYMENT PAGE



Personal Detail

First Name | Last Name

Your Email | Mobile Number

Booking Detail

Pickup Location | Drop Location

Pickup Date | Pickup Time

Select Adult | Select Child

Special Request

Payment

- Paypal
- Direct Check
- Bank Transfer

Reserve Now

Fig-46: Booking details and payment page

CONCLUSION

At TADA, we strive to provide a vehicle rental system that is easy to use and meets the needs of our customers. We understand the importance of safety and maintenance when renting a vehicle, and we take every measure to ensure our vehicles are in excellent condition before and after each rental. Our customer service support and communication channels are available to assist customers with their rental needs and answer any questions they may have. We are committed to considering the needs of a diverse range of customers, including those with special requirements, and continuously adapt our rental system to provide the best possible service. At TADA, we are dedicated to providing a seamless and hassle-free rental experience, prioritizing safety and maintenance, effective communication, and excellent customer service.

In conclusion, a vehicle rental system can be beneficial for both the company and customers by offering a variety of vehicles, flexible rental options, and convenient features such as online reservations and customer support. Prioritizing safety and maintenance are crucial for customer satisfaction.

14. ABBERVATIONS

Abbreviation	***Meaning***
UML	Unified Modelling Language
ARS	Airline Reservation System
PNR	Passenger Name Record
GDS	Global Distribution System
CRS	Computer Reservation System
OTA	Online Travel Agency
IATA	International Air Transport Association
FAA	Federal Aviation Administration
ATC	Air Traffic Control
FFP	Frequent Flyer Program
ABMS	Airbus management system