DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME 2210
Object Oriented Analysis and Design

TICKETWISE

by
2021510061 Ezgi Tan
2021510062 İrem Tekin
2021510072 Kaan Yılmaz

CHAPTER ONE

INTRODUCTION

- **Project Description:**

In our ticket sales application, there will be 2 types of users: customer and company. Initially, there will be different login options for both users. Company: can provide services by entering the agent's information and making it accessible to many people; can make fare, route and date changes; can see ticket fields; Customer: can see matching tickets by selecting vehicle, date and route; can buy tickets for future dates;can also see vehicle occupancy status.

- **Project Purpose:**

The aim of the project is to provide a seamless ticketing system where the customer receives travel services and the company provides travel services. Company: will be able to reach more people and offer more services; at the same time, with customers' route information, aircraft occupancy information, etc. will be able to create more efficient plans for the future. The customer: will be able to get more travel options and at the same time make a more economical decision by comparing prices.

- **Project Scope:**

The scope of the project encompasses the development of a ticket sales application catering to two primary user types: customers and companies.

Features for Company Users:

Login Options: Company users will have dedicated login interfaces tailored to their needs.

Service Provision: Companies can input agent information, facilitating access to a wide audience.

Flexibility in Services: Companies can make adjustments to fare, route, and date parameters as needed.

Ticket Management: Access to view ticket fields and manage ticketing operations.

Features for Customer Users:

Login Options: Customers will have distinct login functionalities tailored to their requirements.

Ticket Search: Customers can search for matching tickets based on vehicle, date, and route preferences.

Ticket Purchase: Customers can buy tickets for future travel dates.

Vehicle Occupancy Status: Ability to view the occupancy status of vehicles.

CHAPTER TWO

REQUIREMENTS

Specific Requirements

2.1 External Interfaces

- **User Interface**
- **Company Interface**
- **Vehicle Interface**

2.2 Functions

Customer:

- addCustomer(String name, String surname, String email, String phoneNumber, String gender, int identity, String birthday, String password, boolean policy): To add a new customer. (void)
- login(String email, String password): For the customer to log in. (boolean)
- writeToFile(String filename, Customer customer): In order to keep customer information constantly, it writes customer information to txt. (void)
- readFromFile(String filename): Reads customer information from the txt where the information is stored. (void)
- displayCustomerInfo(int identity): To display customer information. (Customer)
- getter/setter(): For accessing and modifying customer attributes. (void)

Company:

- addCompany(String companyName, String companyPhoneNumber): To add a new company. (void)
- loginCompany(String companyName, String password): For the company to log in. (boolean)
- writeToFile(String filename, Customer customer): In order to keep company information constantly, it writes company information to txt. (void)
- readFromFile(String filename): Reads company information from the txt where the information is stored. (void)
- displayCompanyInfo(String companyName): To display company information. (Company)
- getter/setter(): For accessing and modifying company attributes. (void)
- addVehicle(Vehicle vehicle, Company company, int capacity, int numberPlate): To add a new vehicle to the company. (void)
- removeVehicle(int numberPlate): To remove a vehicle from the company. (void)

Bus:

- checkAvailableBuses(String originStation, String destinationStation, String departureTime): To check available buses. (boolean)
- addBusTrip(String originStation, String destinationStation, String departureTime, int travelTime, String date, boolean wirelessConnection, boolean catering, Vehicle vehicle, int journeyNumber): To add a new bus trip. (void)
- removeBusTrip(String originStation, String destinationStation, String departureTime, Vehicle vehicle): To remove a bus trip. (void)
- getter/setter(): For accessing and modifying bus attributes. (void)
- displayBusInfo(String originStation, String destinationStation, String departureTime): To display bus trip information. (Bus)

Train:

- checkAvailableTrains(String originStation, String destinationStation, String departureTime): To check available trains. (boolean)
- addTrainTrip(String originStation, String destinationStation, String departureTime, int travelTime, String date, boolean wirelessConnection, boolean catering, Vehicle vehicle, int journeyNumber): To add a new train trip. (void)
- removeTrainTrip(String originStation, String destinationStation, String departureTime, Vehicle vehicle): To remove a train trip. (void)
- getter/setter(): For accessing and modifying train attributes. (void)
- displayTrainInfo(String originStation, String destinationStation, String departureTime): To display train trip information. (Train)

Plane:

- checkAvailablePlanes(String originStation, String destinationStation, String departureTime, Vehicle vehicle): To check available planes. (boolean)
- addPlaneTrip(String originStation, String destinationStation, String departureTime, int travelTime, String date, boolean wirelessConnection, boolean catering, Vehicle vehicle, int journeyNumber): To add a new plane trip. (void)
- removePlaneTrip(String originStation, String destinationStation, String departureTime, Vehicle vehicle): To remove a plane trip. (void)
- getter/setter(): For accessing and modifying plane attributes. (void)
- displayPlaneInfo(String originStation, String destinationStation, String departureTime): To display plane trip information. (Plane)

Policy:

- displayTermsAndConditions(): To display terms and conditions. (void)
- fileOperations(): To read the KVKK (Personal Data Protection Law) text. (void)
- checkPolicy(boolean userConfirmation): To check acceptance of KVKK. (boolean)
- getter/setter(): For accessing and modifying policy attributes. (void)

Payment:

- processPayment(int debitCardNumber, String debitCardName, String debitCardDate, int cvv): To process a payment. (boolean)
- getter/setter(): For accessing and modifying payment attributes. (void)
- displayPayment(): To display the payment screen. (void)

Search:

- searchForTicket(Customer customer): To search for a ticket. (boolean)
- getter/setter(): For accessing and modifying search attributes. (void)
- displaySearch(): To display the search screen. (void)

2.3 Performance Requirements

The website needs to respond quickly and users need to complete ticket search and purchase.

It must be resistant to heavy traffic and be able to meet high user demands.

2.4 Logical Database/File System Requirements

To store user profile information, vehicle information, ticket information, policy information, and payment information, the usage of .txt files.

2.5 Design Constraints

- A desktop interface will be created using Java Swing.
- Necessary measures should be taken for the security of user information. (for example, password hashing and secure session management)

2.6 Software System Quality Attributes

- **Reliability:** The system must process user information correctly and make ticket reservations without errors.
- **Usability**: A user-friendly desktop interface should be designed, and users should be able to easily search for trips and purchase tickets.
- **Portability:** The application to be written in Java must be able to run on different operating systems.
- **Flexibility:** The system should be able to adapt to changing requirements and ensure that new features are easily added and existing features can be updated.
- **Testability:** Different components and functions of the software should be testable separately. This simplifies the debugging and quality control processes.
- **Functionality:**
  Ensuring that the system delivers all intended features and functions accurately and effectively, meeting user requirements and expectations.

2.7 Object Oriented Models

- 2.7.1 Analysis Class Model (Static Model)

- **Customer Class:**

  The Customer class encompasses information related to the customer, including personal details (name, surname, email, phone number, etc.), and manages their interactions within the system. This class acts as a bridge between the user and the application functionality.

- **Company Class:**

  The Company class represents companies involved in ticket sales, storing details such as name, address, and contact information. It acts as a container to organize and manage companies within the system, facilitating interactions with different ticket-selling organizations and providing necessary information for transactions.

- **Bus Class:**

  The Bus class handles specific bus-related properties such as seating capacity, routes, departure times, and ticket pricing. It manages tasks like bus ticket availability checks and fare calculations. This class serves as a model for managing bus-related functions within the application and allows for the display of ticket information.

- **Plane Class:**

  The Plane class controls airplane ticket transactions and includes details such as flight schedules and ticket pricing. It manages aspects like flight availability, seat availability, and ticket structures. This class enables users to reserve airplane tickets, view flight information, and efficiently plan their travel.

- **Train Class:**

  The Train class manages train ticket functions and includes train specifications and ticket pricing. It handles tasks such as train schedules and fare management for train journeys. This class allows users to view train tickets, check availability, and plan their travels efficiently.

- **Policy Class:**

  The Policy class manages the business policies of the application, including other operational guidelines and rules. It contains rules such as data protection regulations and provides necessary information to customers about these policies, ensuring compliance with established regulations. This class serves as a repository for storing and managing business rules and personal data protection law guidelines within the system.

- **Payment Class:**

  The Payment class manages payment transactions and processes ticket purchases made by customers. It securely manages payment information, facilitates payment processes through various methods, and ensures the integrity and confidentiality of financial transactions. This class plays a crucial role in facilitating smooth and secure payment transactions within the application.

- **Menu Class:**

  The Menu class manages the application's main menu and handles user interactions. It facilitates access to different functionalities and interfaces within the system, serving as an access point for users to various features offered by the application.

- 2.7.2 Analysis Collaborations (Dynamic Model)

- **Search Class:**

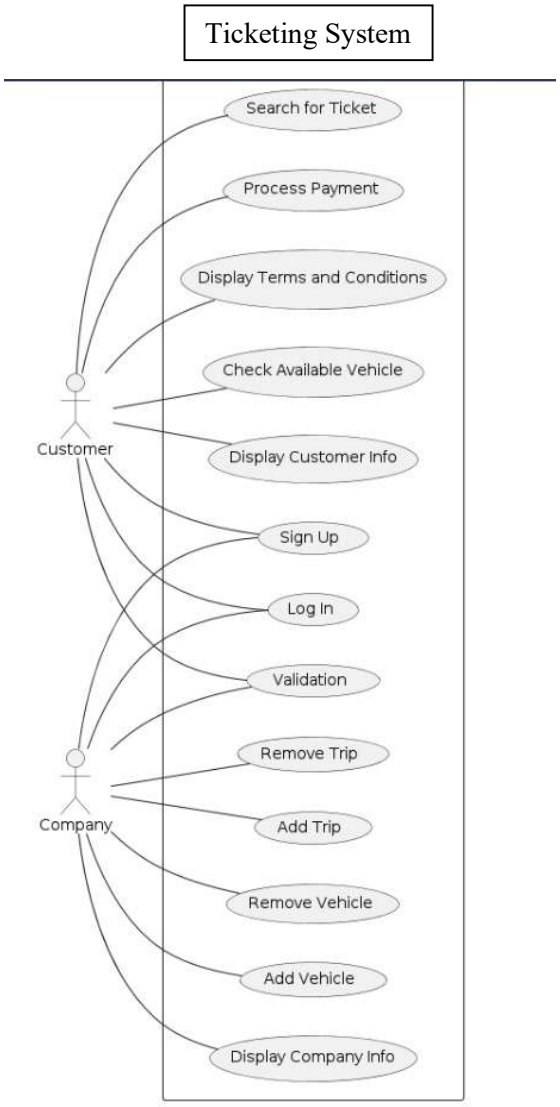
  The Search class performs ticket searches based on criteria specified by the user and efficiently finds suitable travel options. It uses search algorithms to fetch available tickets that match user preferences. This class optimizes the process of discovering and selecting suitable travel options within the system, enhancing the user experience.

CHAPTER THREE

UML DIAGRAMS

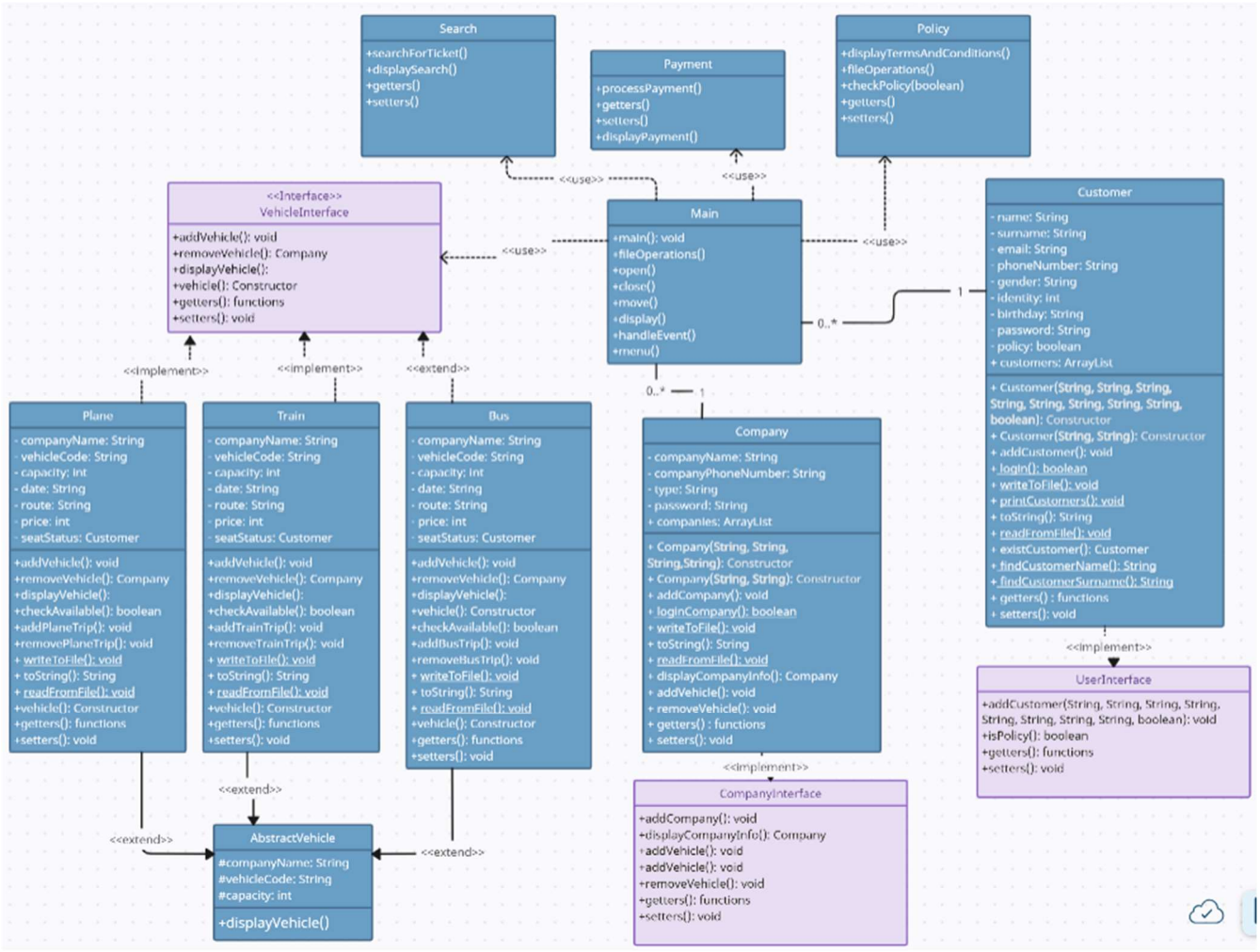
Use Case Diagram:

The presented use case diagram depicts the functions and user interactions of a ticket sales system. There are two main actors in the diagram: 'Customer' and 'Company'. The customer actor has the authority to initiate various use cases such as ticket search, pay transactions, current vehicle control and is in direct interaction with the system. Dec. In addition, customers can register in the system, log in and view their personal information. On the other hand, the company actor performs administrative operations such as adding and removing journeys and vehicles to the system. The interactions between both actors are managed and coordinated through the use cases provided by the system.



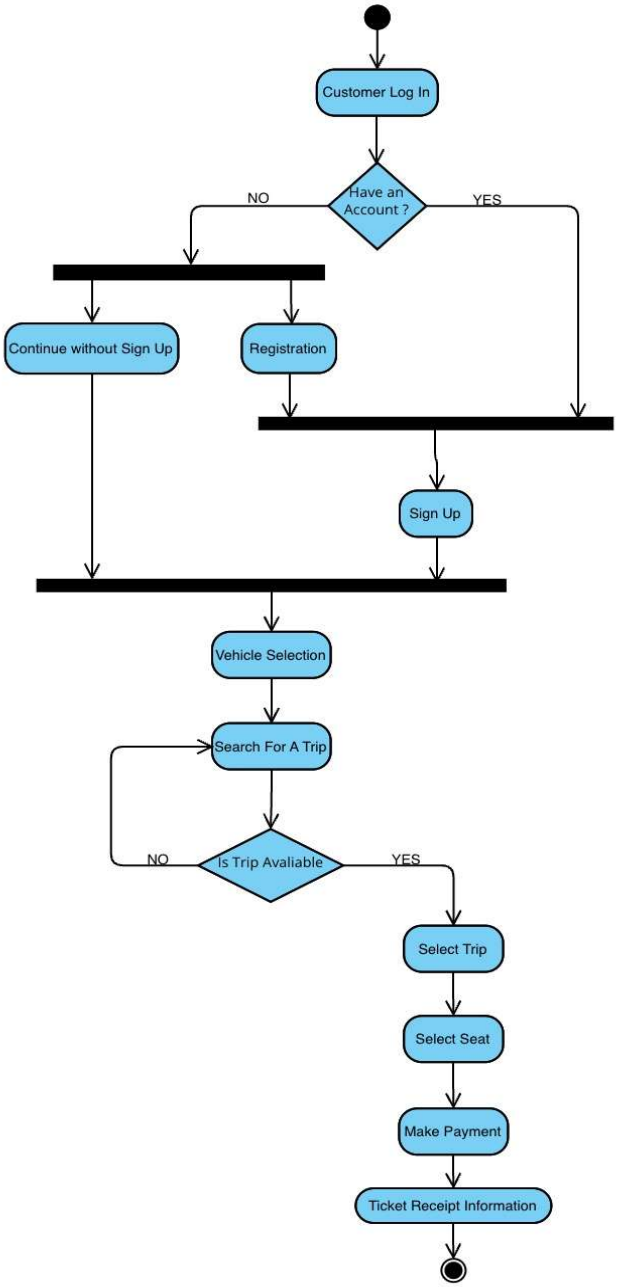
Class Diagram:

This comprehensive class diagram manages ticket sales automation for various transportation vehicles within a company. It includes classes and methods for planes, trains, and buses, utilizing inheritance and interfaces. AbstractVehicle contains common properties like companyName, vehicleCode, and capacity for all vehicles. Plane, Train, and Bus Classes extend AbstractVehicle and add ticket-specific attributes such as date, route, and price, with methods to manage tickets and journey details. Company Class manages companies and their vehicles, organizing ticket sales. Customer Class handles customer information and ticket purchasing processes. UserInterface offers interfaces for user interaction and displays ticket transactions. CompanyInterface and VehicleInterface ensure consistency in company and vehicle management methods. Search, Payment, Main, and Policy Classes address key components of ticket sales, like searching, payment processing, application operations, and terms handling.

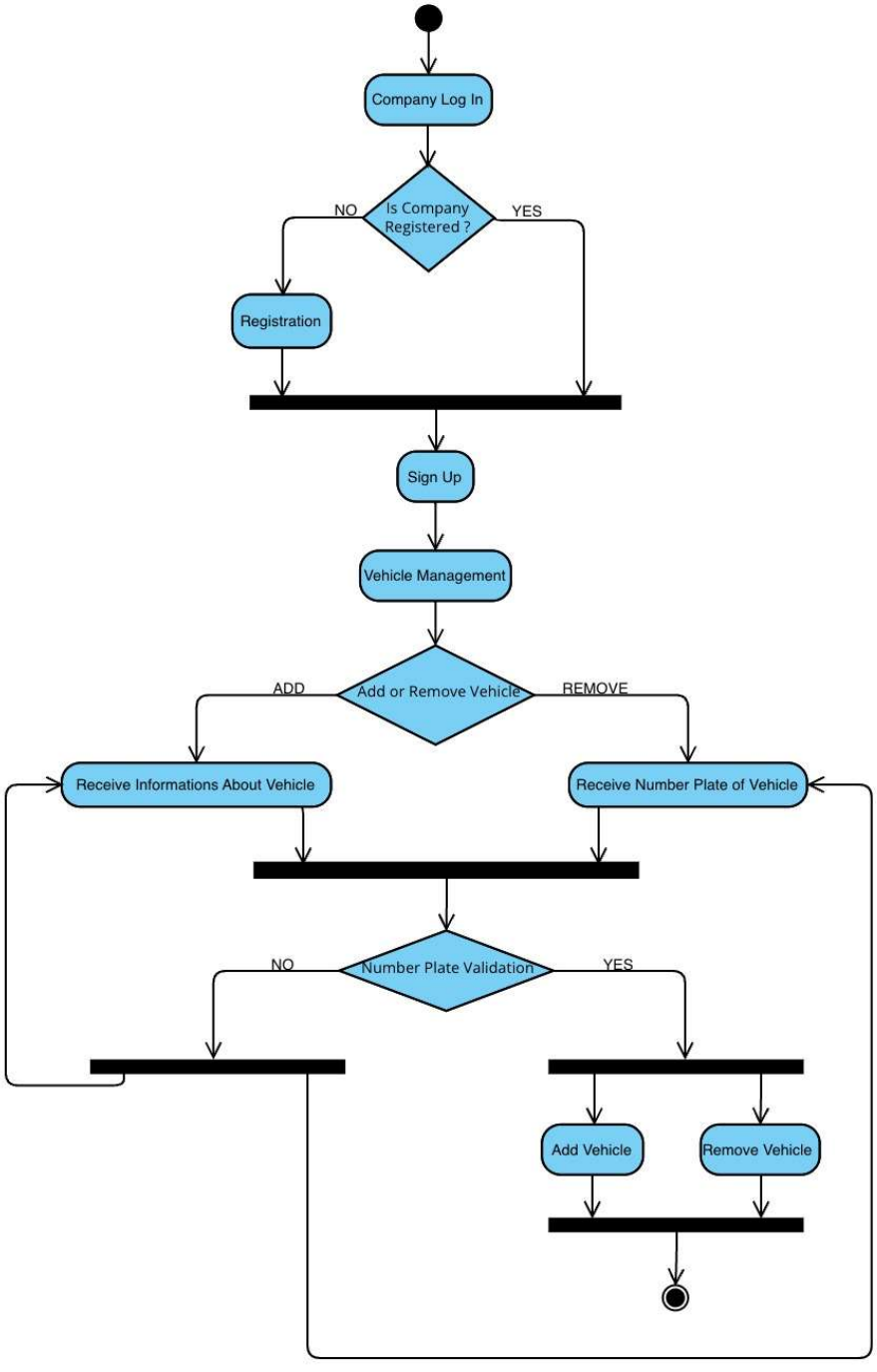


Activity Diagrams:

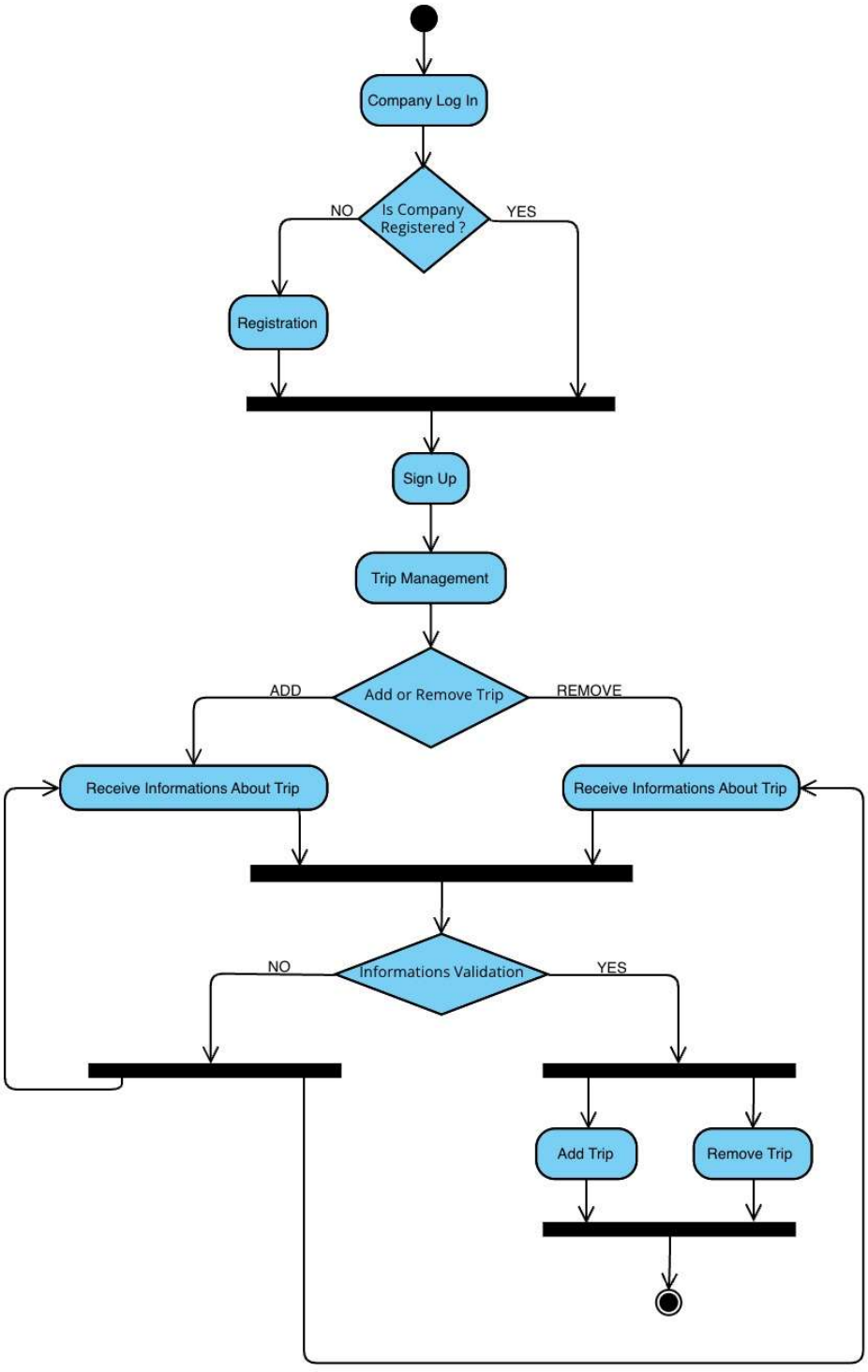
1. This activity diagram represents the customer interaction process with a transportation service platform. The customer starts with a decision point where they have to log in. Those who have an account can proceed directly, while those who do not have the option to continue without logging in or create an account. Those who register make direct transactions. After login or registration, customers select vehicle type and search for travel. If a trip is found, the process continues; otherwise it will terminate or the customer will restart the call. Once a suitable trip is found, customers select their preferred trip and select a seat before paying. Finally, the interaction ends with the ticket receipt information. Decision points represent choices, and rectangular blocks represent actions or processes.



2. This activity diagram shows the processes for logging into and registering vehicles in a company's vehicle management system. The diagram starts with the company's entry into the system and checks whether the company is registered. If not registered, the company must be registered; If registered, it indicates that you can log in to the system. After logging in, proceed to the 'Vehicle Management' step. Here, the user can either add a vehicle or delete an existing vehicle. To add a vehicle, information about the vehicle is obtained and license plate verification is performed. If the verification is successful, the tool is added to the system. For vehicle deletion, the vehicle's license plate is taken and removed from the system after verification. Both processes are terminated. These processes allow the company to effectively manage its vehicle fleet.



3. This activity diagram illustrates the entry and trip registration processes of a company's trip management system. It commences with the company's entry into the system and checks whether the company is registered. If not, it indicates the need for registration; if yes, it specifies that the company can log in. Upon logging in, it proceeds to the 'Trip Management' step. Here, the user can either add a trip or delete an existing one. For trip addition, trip-related information is gathered, and verification is performed. If verification is successful, the trip is added to the system. For trip deletion, information related to the trip to be deleted is collected, and after verification, it is removed from the system. Both processes are concluded.



Sequence Diagrams:

1. This sequence diagram shows the interactions between Customers, Companies, and their respective processes within a system for login and signup scenarios. Two main conditional flows are shown: one for existing entities and one for non-existing ones.

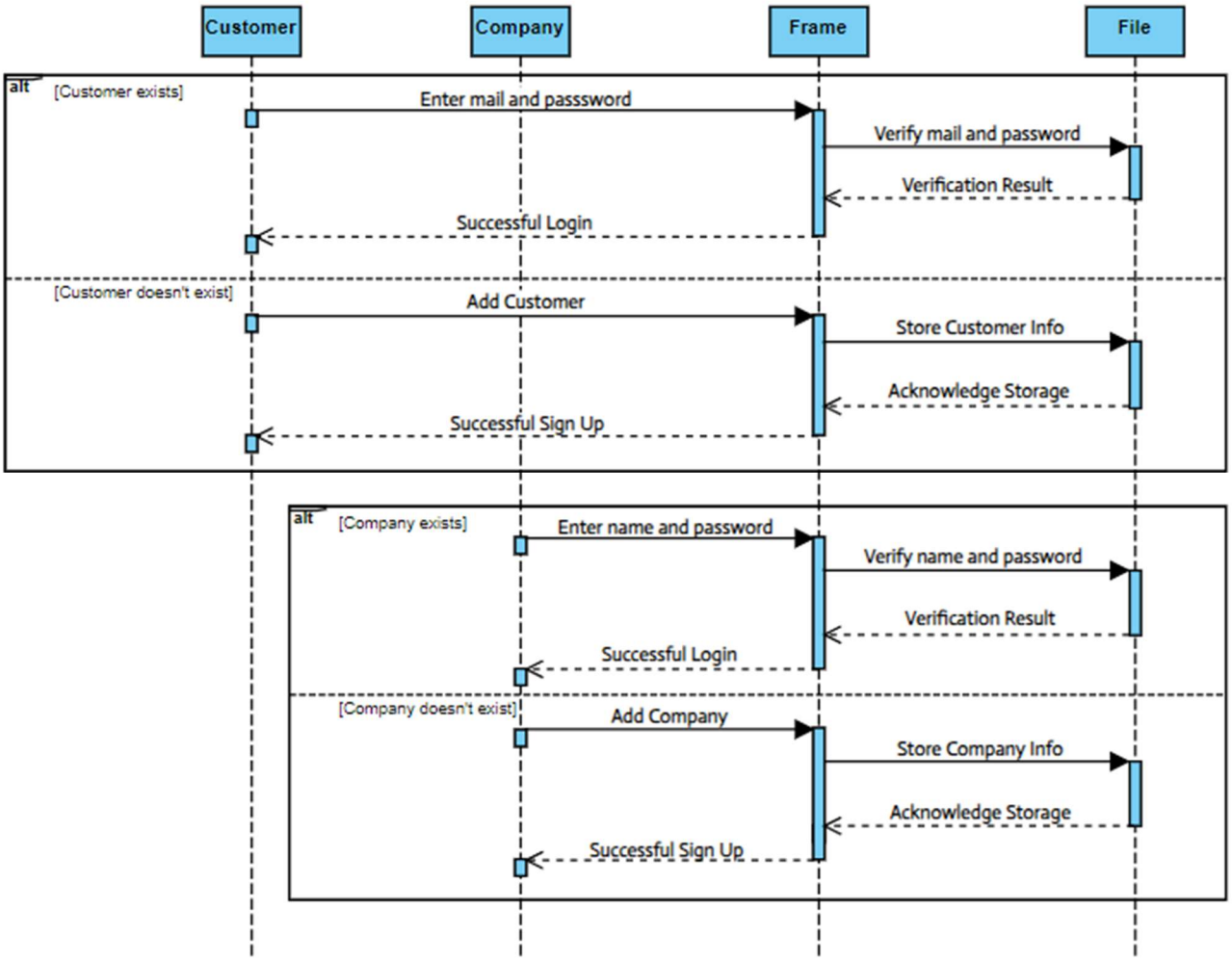
   For customers:

   Existing customers enter their email and password, resulting in verification and a “Successful Login” response. Non-existing customers go through an "Add Customer" process where details are provided and confirmation is received with a "Successfully Signed Up" message.

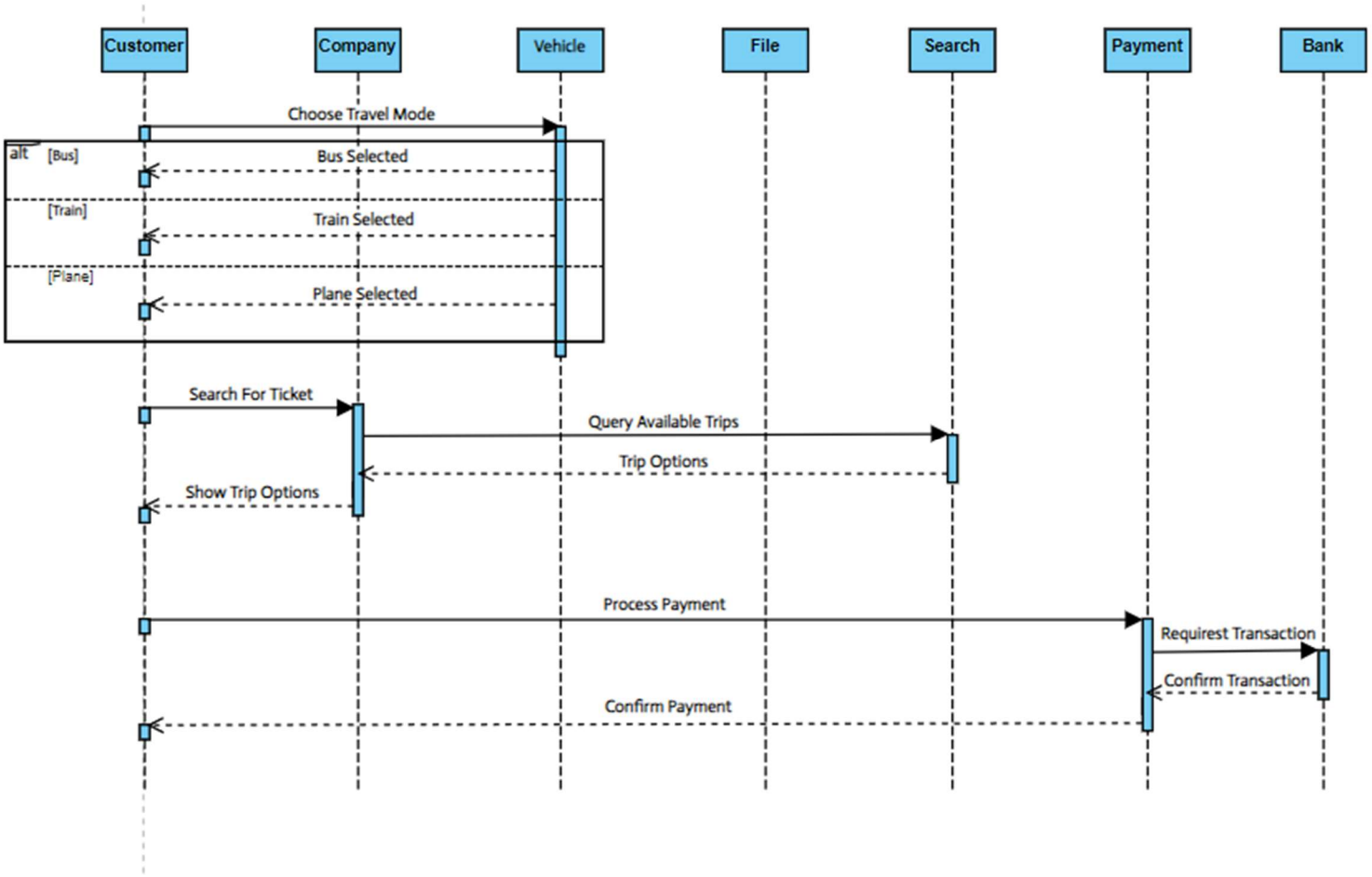
   For companies:

   Existing companies enter their name and password for verification, resulting in a "Successful Login" response. Companies that do not exist go through an "Add Company" process, enter information and receive confirmation with a "Successful Registration" message.

   The "Frame" and "File" components deal with the validation and storage processes.



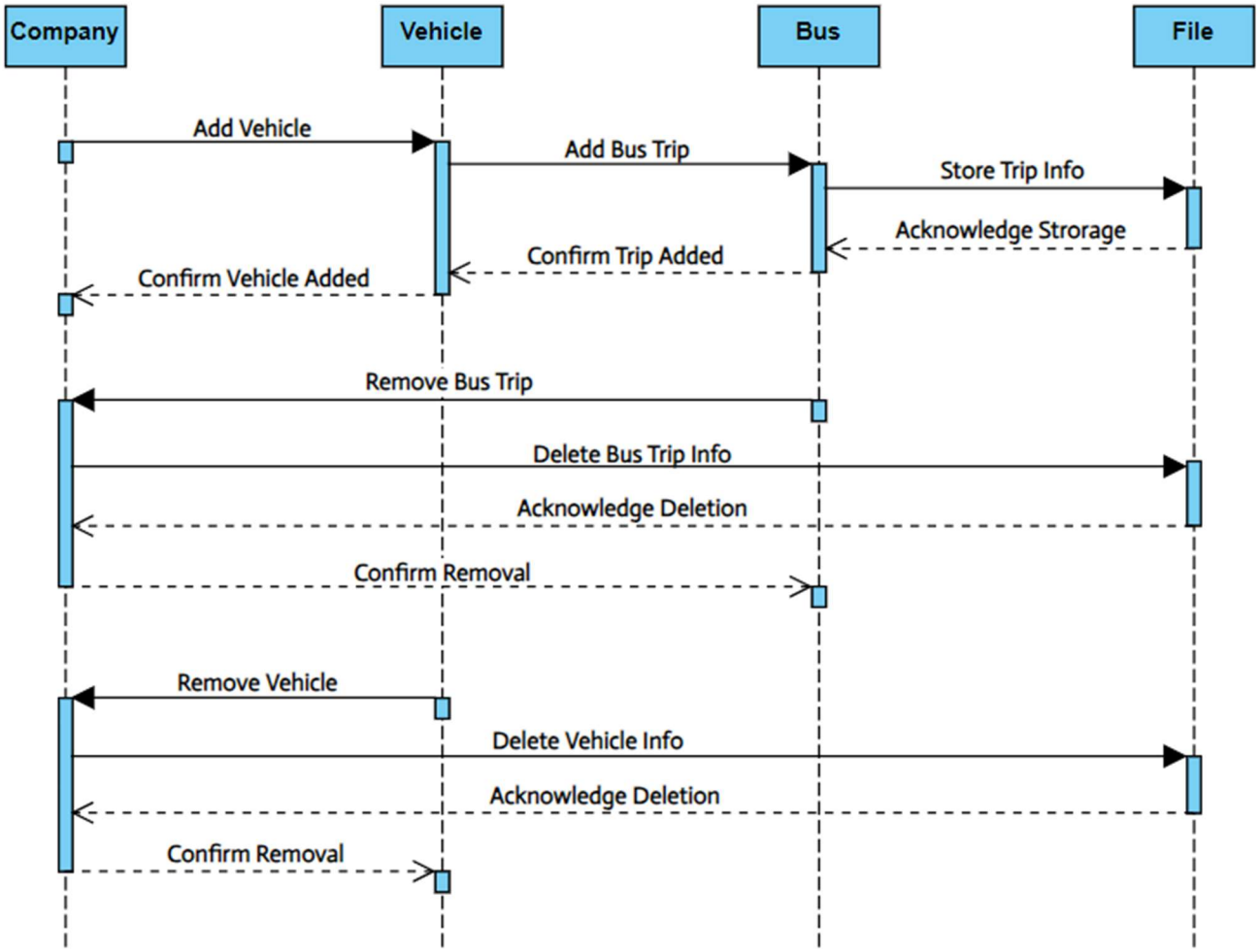
2. This sequence diagram describes a transaction flow where a customer selects the travel mode and then searches for tickets, queries available sailings, shows travel options, and makes payment. This transaction flow also includes alternative routes depending on the customer's choice of travel mode (bus, train or plane) and interactions with the bank during the payment process.



3. This sequence diagram visually represents a company's vehicle and bus schedule management system. The company can operate on this system by adding or removing vehicles and bus routes. While operations related to vehicles occur in interaction with the 'Vehicle' component, operations related to bus services occur through the 'Bus' component. Additionally, trip information is stored in the file and this file is managed by the 'File' component.

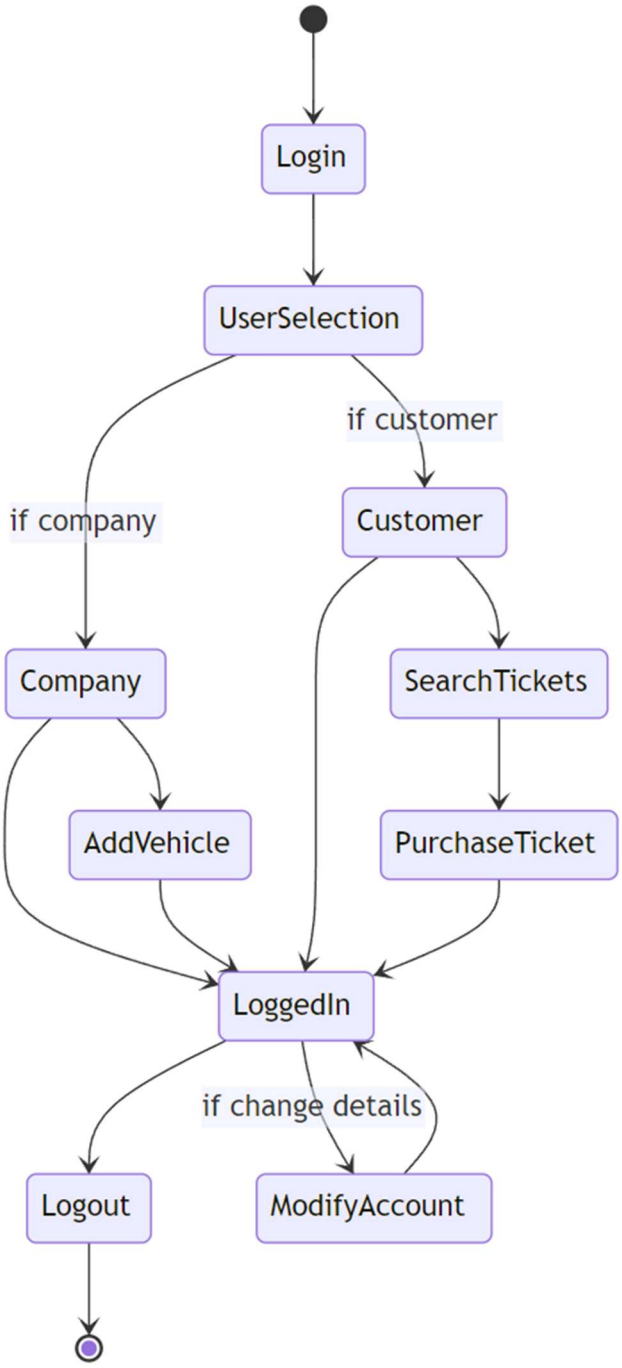
   When the company wants to add a vehicle, the 'Add Vehicle' process is initiated and the 'Vehicle' component confirms that this addition process has been completed successfully with the 'Confirm Vehicle Added' message. To add a bus trip, the 'Add Bus Trip' process is started and the 'Bus' component notifies you that this addition is successful with the 'Confirm Trip Added' message.

   When the company wants to cancel a bus trip, the 'Remove Bus Trip' process is initiated and the 'Bus' component responds to this request with the 'Delete Bus Trip Info' process. Similarly, to remove a vehicle from the system, the 'Remove Vehicle' operation is initiated and the 'Vehicle' component performs this deletion through the 'Delete Vehicle Info' operation.

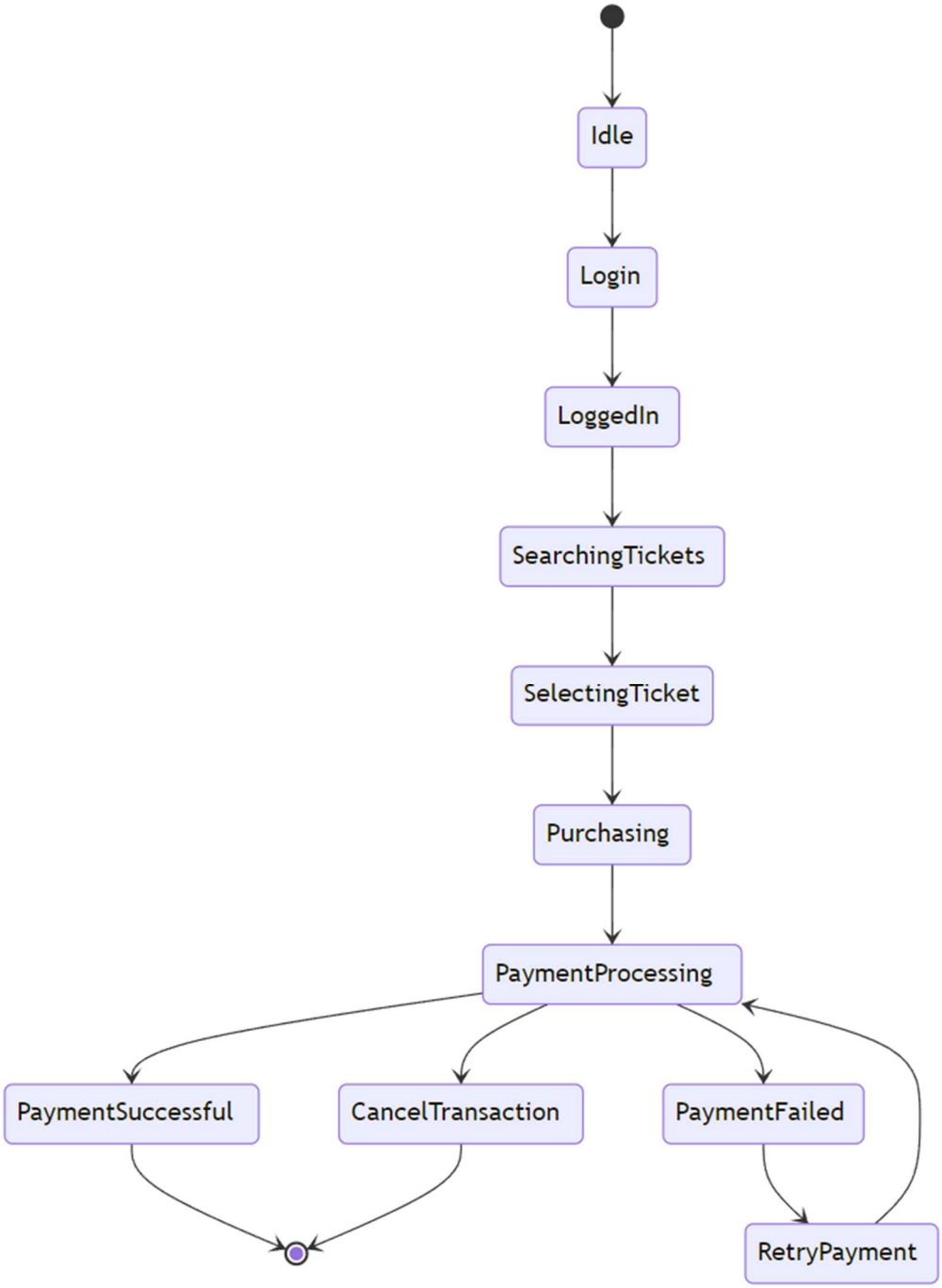


State Diagrams:

1. This state diagram represents the flow of user interaction within a system after logging in. The process starts at the "Login" state and transitions to "UserSelection" where there are two conditional paths: one for a company and one for a customer. If "company" is selected, the state transitions to "Company," which has an option to "AddVehicle" before reaching the "LoggedIn" state. If "customer" is chosen, the state moves to "Customer," from where the user can either "SearchTickets" or "PurchaseTicket," eventually leading to the "LoggedIn" state. Once logged in, the user has the option to "ModifyAccount" if there is a need to change details. Otherwise, they can directly "Logout" of the system.

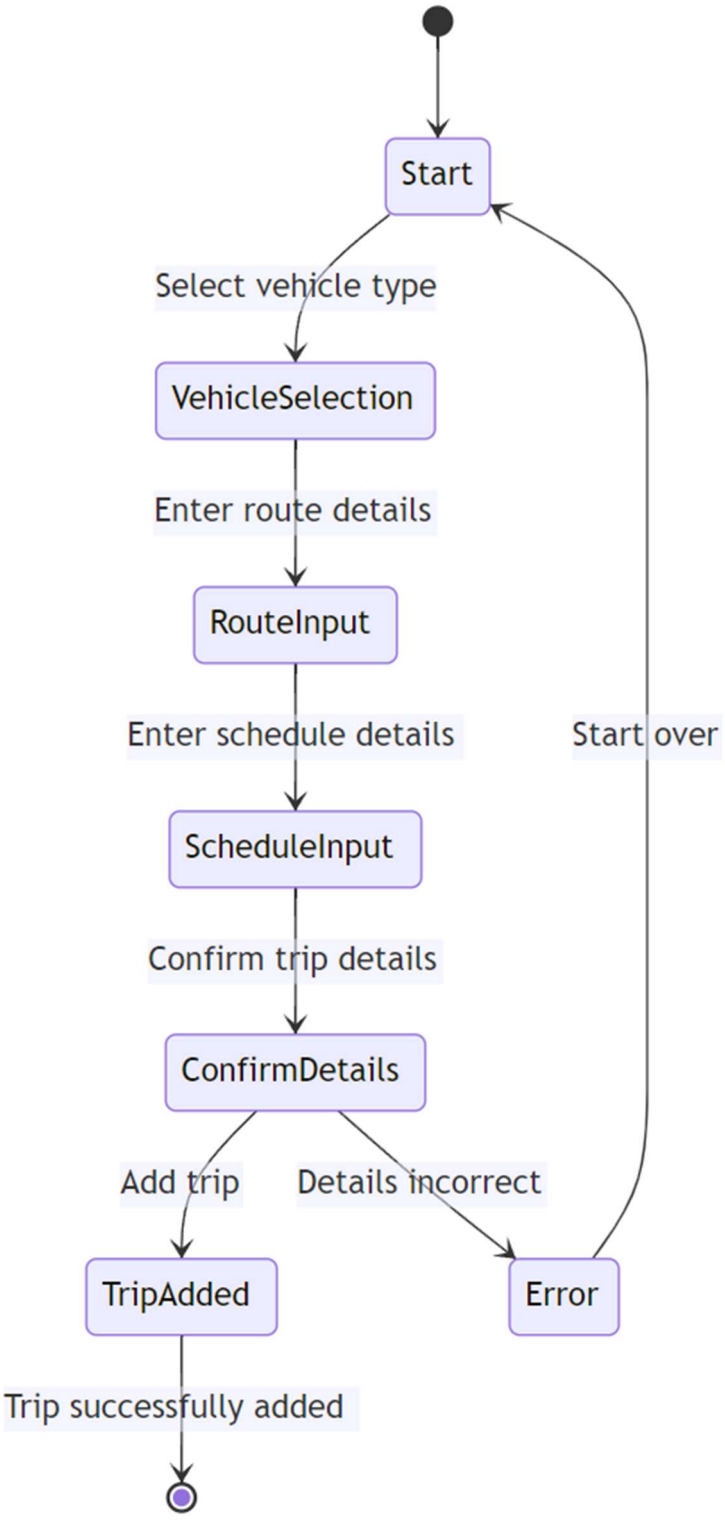


2. This state diagram shows the steps users go through during the ticket purchasing process and the transitions between these steps. First, the user logs into the system ("Login") and logs in ("LoggedIn"). It then starts the ticket search process ("SearchingTickets") and selects a ticket ("SelectingTicket"). Then, he/she takes action to purchase the ticket ("Purchasing"). Three different outcomes are possible during the payment process: "PaymentSuccessful" status indicates that the payment was successful and the transaction was completed; The "CancelTransaction" status indicates that the transaction has been canceled; The "PaymentFailed" status indicates that the payment has failed and the user is given the chance to retry the payment ("RetryPayment").



3. This state diagram explains a company's step-by-step process for adding trips. The first step is for the company to start the process. Next, the company selects the vehicle type and enters details about the route. He then enters the details about the voyage and finally reviews and confirms all the information.

   If the information entered is complete, the trip is added to the system and the process is completed successfully. However, if there is an error in the information entered, the system sends an error message to the company. To fix the error, the company can go back to the starting point and start the process again.



CHAPTER FOUR
IMPLEMENTATION

To thoroughly explain the interfaces and background code of the TicketWise application, I'll provide a detailed breakdown of the key components as outlined in the document.

Interfaces

1. User Interface

The TicketWise application will utilize Java Swing for its graphical user interface (GUI). This will involve creating various frames and panels to handle different functionalities for both customer and company users.

2. Login Interface

- **Customer Login Panel**: A form for customers to input their email and password, with options to register if they don't have an account.
- **Company Login Panel**: Similar to the customer login panel but tailored for companies, allowing them to input their company name and password.

3. Main Menu

- **Customer Menu**: Options for searching and purchasing tickets, viewing ticket status, and account management.
- **Company Menu:** Options for managing vehicles, adding or removing trips, and viewing trip details.

4. Search and Purchase Panels

- **Search Panel:** Allows customers to search for available tickets by entering criteria such as origin, destination, and date.
- **Purchase Panel:** Handles the ticket purchasing process, including payment details and confirmation.

5. **Company Interface**

This interface includes functionalities specific to company users, such as managing vehicles and trips:

6. **Vehicle Management Panel**

- **Add Vehicle**: Form for inputting vehicle details such as capacity, number plate, and type (bus, train, plane).
- **Remove Vehicle**: Allows the removal of vehicles by entering the vehicle's number plate.

7. **Trip Management Panel**

- **Add Trip:** Form to add trip details like origin, destination, departure time, and additional services (wireless connection, catering).
- **Remove Trip:** Allows companies to delete trips by specifying the trip details.

BACKGROUND CODES

1. **Customer Class:**

- **General Purpose**

The Customer class is designed to manage customer information, including storing customer details, adding new customers, reading and writing customer data to and from a file, handling customer login, and querying customer information.

- **Methods**

Add Customer:

Creates a new customer and adds it to the list of customers. Additionally, it writes the customer details to a file.

Login:

Reads customer information from a file and checks if there is a customer with the provided email and password. If a matching customer is found, it returns true; otherwise, it returns false.

Write to File:

Writes customer information to the specified file. This method appends the customer details to the end of the file.

Read from File:

Reads customer information from the specified file and adds each customer to the list of customers. This is used to persist customer information across sessions.

Print Customers:

Prints the details of all customers to the console. This is useful for viewing all the customer information stored in the system.

- **Additional Methods**

Find Existing Customer by Identity:

Searches for a customer with the specified identity number and prints the customer details if found. If no customer is found, it notifies that no customer was found.

Find Customer Name by Email:

Returns the first name of the customer with the specified email address.

Find Customer Surname by Email:

Returns the last name of the customer with the specified email address.

The Customer class provides essential functionalities for customer management, including adding new customers, reading and writing customer data to a file, handling login, and querying customer information. This class is fundamental for managing customer data efficiently in a customer management system.

2. **Company Class:**

- **General Purpose**

The Company class is designed to manage company information, including storing company details, adding new companies, reading and writing company data to and from a file, handling company login, and querying company information.

- **Methods**

Add Company:

Creates a new company and adds it to the list of companies. Additionally, it writes the company details to a file.

Login Company:

Reads company information from a file and checks if there is a company with the provided name and password. If a matching company is found, it returns true; otherwise, it returns false.

Write to File:

Writes company information to the specified file. This method appends the company details to the end of the file.

Read from File:

Reads company information from the specified file and adds each company to the list of companies. This is used to persist company information across sessions.

Display Company Info:

Prints the details of the company with the specified name to the console. If no company is found, it notifies that no company was found.

Find Company Type:

Returns the type of the company with the specified name by reading from the file.

The Company class provides essential functionalities for managing company data, including adding new companies, reading and writing company data to a file, handling login, and querying company information. This class is fundamental for managing company data efficiently in a company management system.

3. **Bus-Train-Plane Class**

- **General Purpose**

Bus-Train-Plane classes extends the Vehicle class and is designed to manage bus/train/plane information within the ticket management system. This class includes functionalities for adding, removing, and verifying the existence of buses, as well as reading and writing bus/train/plane data to and from a file.

- **Methods**

Add Vehicle:

Adds a new bus/train/plane to the list of vehicles and writes its details to a file.

Overrides the addVehicle method from the Vehicle class.

Remove Vehicle:

Reads the vehicle data from a file, removes bus/train/plane with the specified license plate from the list, and updates the file with the remaining vehicles.

Overrides the removeVehicle method from the Vehicle class.

Read from File:

Clears the current list of vehicles and reads vehicle data from the specified file, adding each bus/train/plane to the list.

Overrides the readFromFile method from the Vehicle class.

Write to File:

Writes the details of a specific vehicle to a specified file, appending the data to the end of the file.

Overrides the writeToFile method from the Vehicle class.

Write to File Updated:

Clears the content of the specified file and writes the updated list of vehicles to it.

Overrides the writeToFileUpdated method from the Vehicle class.

Exist Vehicle:

Checks if a bus with the specified license plate exists in the list of vehicles and if it belongs to the currently logged-in company.

Overrides the existVehicle method from the Vehicle class.

Clean Text File:

Clears the content of the specified file.

Overrides the cleanTxt method from the Vehicle class.

The Bus/Train/Plane class provides essential functionalities for managing bus data, including adding new bus/train/plane, removing existing bus/train/plane, and verifying the existence of bus/train/plane. This class is fundamental for managing bus/train/plane information efficiently in a vehicle management system.

4. Vehicle Class:

- **General Purpose**

The Vehicle class is an abstract class that provides a blueprint for managing vehicle-related information within the ticket management system. This class includes abstract methods that need to be implemented by subclasses, such as Bus. It also contains common fields and methods that are shared among all vehicle types.

- **Abstract Methods**

addVehicle:

Adds a new vehicle to the list of vehicles and writes its details to a file.

removeVehicle:

Removes a vehicle with the specified license plate from the list and updates the file.

readFromFile:

Reads vehicle data from the specified file and populates the list of vehicles.

writeToFile:

Writes the details of a specific vehicle to a specified file.

toString:

Returns a string representation of the vehicle details.

writeToFileUpdated:

Writes the updated list of vehicles to a specified file.

existVehicle:

Checks if a vehicle with the specified license plate exists in the list.

cleanTxt:

Clears the content of the specified file.

The Vehicle class serves as a foundation for managing vehicle data, providing a common structure and defining essential methods that must be implemented by specific vehicle types such as buses, traines, etc. This ensures consistency and reusability in handling different types of vehicles within the system.

5. **Trip Class:**

- **General Purpose**

The Trip class is designed to manage trip-related information within the ticket management system. This class includes functionalities for adding, removing, checking the existence of trips, and reading and writing trip data to and from a file.

- **Methods**

Add Trip:

Adds a new trip to the map of trips and writes its details to a file.

Check if Trip Exists:

Checks if a trip with the specified details exists in the map of trips.

Check if Vehicle Exists:

Checks if a vehicle with the specified company name and license plate exists in the vehicles.txt file.

Find Trip Number:

Finds the trip number based on the specified trip details.

Write Trip to File:

Writes the details of a specific trip to the specified file.

Remove Trip:

Removes a trip with the specified details from the map and updates the file.

Read Trips from File:

Reads trip data from the specified file and populates the map of trips.

Write Trips to File Updated:

Writes the updated list of trips to the specified file.

Clean Text File:

Clears the content of the specified file.

The Trip class provides essential functionalities for managing trip data, including adding new trips, removing existing trips, and verifying the existence of trips. This class is fundamental for managing trip information efficiently in a trip management system.

6. **Seat Class:**

- **General Purpose**

The Seat class is designed to manage seat availability for trips within the ticket management system. This class includes functionalities for adding trips with seats, updating seat availability, removing trips and their associated seats, and reading and writing seat data to and from a file.

- **Methods:**

Write to File:

Writes the seat availability data to the seats.txt file.

Update Seat:

Updates the availability status of a specific seat for a specific trip and writes the updated data to the file.

Remove Trip:

Removes all seat data for a specific trip and updates the file.

Add Trip:

Adds a new trip with default seat availability (all seats available) and writes the updated data to the file.

Read Seats from File:

Reads seat availability data from the seats.txt file and populates the seats map.

Check Seat Availability:

Checks if a specific seat is available for a specific trip.

The Seat class provides essential functionalities for managing seat availability, including adding new trips with seats, updating seat availability, and removing trips with their seats. This class is fundamental for managing seat information efficiently in a seat management system.

7. Payment Class:

- **General Purpose**

The Payment class is designed to handle payment information within the ticket management system. This class includes functionalities for validating credit card details such as the cardholder's name, card number, and CVV. It also provides getter and setter methods for these fields.

- **Methods:**

Validate Cardholder Name:

Validates the cardholder's name to ensure it has at least a first name and a last name.

Validate Card Number:

Validates the card number to ensure it is exactly 16 digits long and contains only numeric characters.

Validate CVV:

Validates the CVV to ensure it is exactly 3 digits long.

The Payment class provides essential functionalities for managing and validating payment information, including the cardholder's name, card number, expiry date, and CVV. This class is fundamental for handling payment data efficiently in a payment management system.

CHAPTER FIVE
CONCLUSION AND FUTURE WORKS

In this project, we developed a comprehensive ticket sales application, TicketWise, utilizing Java Swing for the user interface. Our application caters to two types of users: customers and companies, each with specific functionalities to enhance the user experience and streamline ticket sales operations.

For customers, the application offers features such as searching for tickets based on vehicle type, date, and route preferences, purchasing tickets for future travel dates, and viewing vehicle occupancy status. The user-friendly interface ensures an intuitive and seamless experience for customers to make informed travel decisions.

For companies, the application provides functionalities to manage their services efficiently. Companies can add and manage vehicles, modify fare routes and dates, and access ticket fields. These features enable companies to reach a broader audience, optimize their service offerings, and plan more effectively based on customer data.

Key technical aspects of our project include:

Java Swing for UI: The desktop interface was created using Java Swing, ensuring portability across different operating systems and providing a flexible, interactive user experience.

Object-Oriented Design: The application was designed following object-oriented principles, promoting modularity, reusability, and maintainability of the codebase.

API Integration: We integrated external APIs to enhance the functionality of TicketWise, such as retrieving real-time vehicle occupancy status and dynamic route updates. The APIs are accessed using OkHttp for HTTP requests, ensuring reliable and efficient data communication.

Security Measures: We implemented necessary security measures, such as password hashing and secure session management, to protect user information.