LONDON METROPOLITAN UNIVERSITY

informatics college • pokhara

Module Code & Module Title

CS5002NP Software Engineering

Assessment Weightage & Type

35% Individual Coursework

Year and Semester

2nd year, 2nd semester

Student Name: Vamsha Palja Tamu

Group: L2C3

London Met ID: 21050019

College ID: NP04CP4A210106

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

List of Figures

List of Tables

1. Introduction

Software engineering is the branch of engineering that develops software products using well-defined scientific principles, methods and procedures to produce efficient and reliable software products (tutorialspoint, n.d.).

In software development, software engineering is important, as it simplifies the process and reduces the complexity of large software, reducing costs and time, and reducing the complexity of the software (Anand, 2023).

Software engineering principles were prevalent in the late 1970s, as by the late 1960s many software projects were failing because many were over budget, difficult to build and maintain over time, and not meeting the growing needs of the business Customers and Demand for new software grew faster compared to the ability to generate new software (Martin, 2023).

We were hired to develop software for Allgemein, a company that has dominated the entertainment industry for a decade. Its source of income comes from projects related to entertainment industry like film making, documentaries, music and game production etc. Lately they wanted to expand their business to transportation industry as well. They bought a large number of vehicles to support their business. Vehicles range from regular taxis to transport/construction oriented vehicles like trucks and bulldozers.

They wanted some services to be concentrated for this project and they were the first to offer taxi services where the customer can book the taxi services to go from one place to another. The taxi service is not limited to the Kathmandu valley, but also applies to inter-state trips. Second, they want to offer rental car services for vehicles like trucks and bulldozers. After depositing a certain amount of money and identification documents, customers can rent the vehicles for a certain period of time, and finally they want a system

that keeps track of all transaction records, customer information and all vehicle information so that the necessary information can be extracted at any time required.

The assumptions that were made for this projects are:

- To book a taxi or rent a vehicle, a person needs a membership to complete these tasks.
- Members can track their request, whether accepted or not, to book a taxi or rent a vehicle.
- The report must be prepared in a specific time period in terms of customers, business, revenue generated and vehicles.
- To participate in the training course, the customer/member can register for this course only after paying a certain deposit in advance.
- The member/diver can negotiate a price that is suitable for both parties.
- Admin registers a staff as whole and latter changed to driver and other staff

2. Gantt chart

A Gantt diagram is a horizontal bar diagram used in project management, which visually represents a project plan over time (teamgantt, n.d.). Tasks are ordered from the critical components or by their start and end dates. Gantt charts can be used to aid project managers and team members to understand project schedules, identify key routes, effectively allocate resources and track project progress. It is a useful tool for the planning, scheduling and monitoring of projects and for the identification of potential delays and problems that may affect the successful completion of the project.

As the goal is fixed, it was decided to use the waterfall methodology. Hence, we decided to plan the flow of the project according to it and it was made as follows:

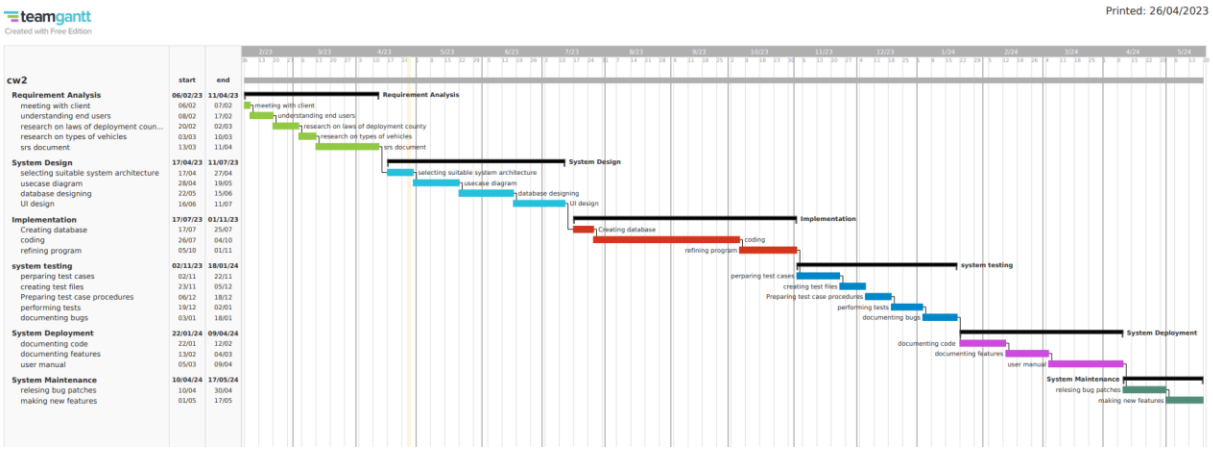
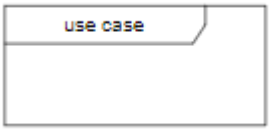


Figure 1: image of Gantt chart

3. Use case diagram

A use case diagram is a diagram which models the behaviour of a system and helps to capture the requirement of the system by describing high-level functions and the scope of the system. This diagram illustrates the interaction between the system and the actor. Use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally (IBM, 2021).

Name	Symbol	Description
System boundaries		All use cases are contained within the system's boundaries.
Use case		A use case is the horizontally shaped oval that represents the different uses of system that a user could use.
Association		it is a line connecting the actors and use cases that helps to identify an association/relation shared between them.
Actor		The actor is a visual representation of a person as a stick figure.

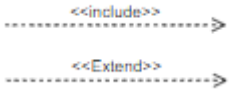
		
Relationship		When a use case is dependent or optionally dependent on another use case, a dotted line is drawn between them as a sign that they have a relationship

Table 1: components of use case diagram

The use case diagram of the software based on the specifications and understandings is presented below:

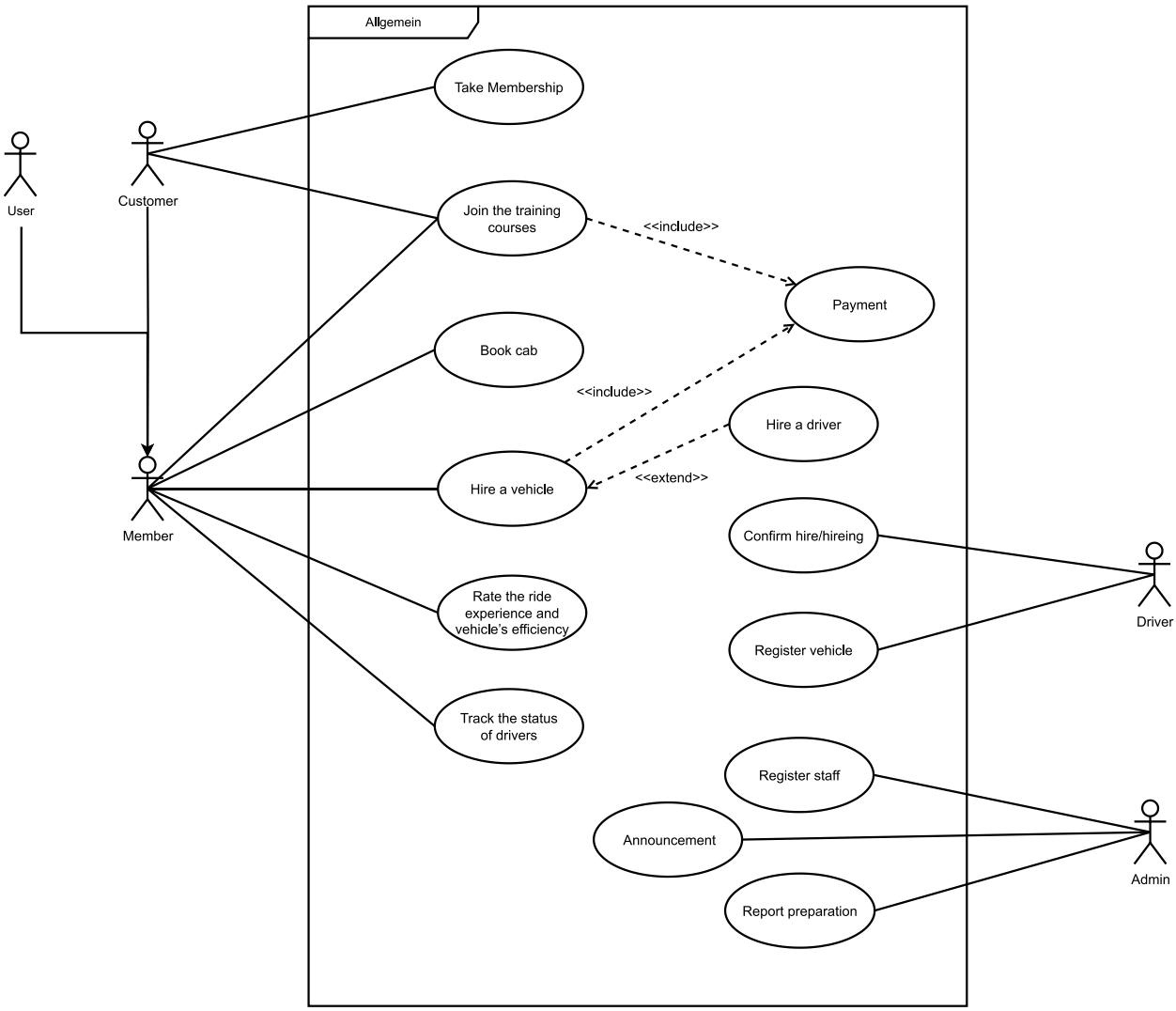


Figure 2: use case diagram for Allgemein

4. High level use case description

The high-level use case is simply a summary description of the task, written as an unstructured text with one or two paragraphs in length. It is intended to provide sufficient detail to give an impression of its complexity and to help group these cases related to the development process in the development phase (PennState College of Earth and Mineral Sciences, n.d.).

Use case	Take Membership
Actor	Customer
Description	As all services that are provided by the software are membership-based, all the customers are required to register if they wish to use any services, in this use case, the customer registers and receives a membership.

Table 2: High level use case description of Take membership

Use case	Join the training courses
Actor	Customer
Description	Members are able to enrol on special paid training that is carried out semi-annually in which they learn how to drive heavy vehicles such as cargo trucks and bulldozers.

Table 3: High level use case description of join the training courses

Use case	Book cab
Actor	Member
Description	the members are able to book cabs by specifying their journey details

Table 4: High level use case description of Book a cab

Use case	Hire a vehicle
Actor	Member
Description	Members are able to hire a vehicle of their choice to suit their requirements and they can also hire a specialist drivers if needed

Table 5: High level use case description of hire a vehicle

Use case	Rate the ride experience and vehicle's efficiency
Actor	Member
Description	Members are able to rate the journey they have undertaken on the quality of its carrying out and efficiency. The members also have the ability to give suggestions.

Table 6: High level use case description of rate the ride experience and vehicle's efficiency

Use case	Track the status of drivers
Actor	Member
Description	Members can track the status of their request being accepted or denied. Members can also visualize their location.

Table 7: High level use case description of track the status of drivers

Use case	Payment
Actor	
Description	As customers and members must pay a certain fee in advance before enrolment. The members also have to pay their fee for travelling in a cab and while hiring a vehicle, they also have to pay in advance and the remaining payment could be done by their choice.

Table 8: High level use case description of payment

Use case	Hire a driver
Actor	
Description	During vehicles hire, the members are able to hire a specialist driver if they wish.

Table 9: High level use case description of hire a driver

Use case	Confirm hire
Actor	Driver
Description	Upon request, the driver can choose whether to accept or deny any request from cab booking or vehicle hire.

Table 10: High level use case description of confirm hire

Use case	Register vehicle
Actor	Driver
Description	The driver could register their vehicle by fulfilling the required details.

Table 11: High level use case description of register vehicle

Use case	Register staff
Actor	Admin
Description	After verifying the necessary details and submitting proof, the admin registers the staff.

Table 12: High level use case description of register staff

Use case	Report preparation
Actor	Admin
Description	Admin is given a generated report related to customers, business, revenue generated, and vehicles.

Table 13: High level use case description of report preparation

Use case	Announcement
Actor	Admin
Description	The Admin announces the announcement about the start of a specific course.

Table 14: High level use case description of announcement

5. Expanded use case diagram

5.1. Expanded use case diagram of join training course

Use case: join training course

Actor: customer

Description: Members are able to enrol on special paid training that is carried out semi-annually in which they learn how to drive heavy vehicles such as cargo trucks and bulldozers.

Actor	System
	1. System shows all the available courses
2. Actor selects the course they want	
	3. System asks for actors' personal detail by giving a form
4. Actor fill the form with their personal details	
	5. System shows the price of that course
6. Actor pays with any type of card they prefer	
	7. Confirm Enrollment

Table 15: Expanded use case diagram of join training course

End cases:

- Use case ends on 2 when the system does not have the course that customer wanted.
- Use case ends on 6 when actor does not have a card for payment.

5.2. Expanded use case diagram of book cab

Use case: Book Cab

Actor: Member

Description: the members are able to book cabs by specifying their journey details

Actor	System
1. Actor fills journey detail in form	
	2. System sends a confirmation request to proceed
	3. System shows verities of payment method
4. Actor select their preferred method of payment	
	5. System asks address to send the billing
	6. System sends the billing

Table 16: Expanded use case diagram of book cab

End cases:

- Use case ends on 4, if the actor does not possess the payment methods that were displayed.

6. Communication/Collaboration Diagram

Communication diagrams could be defined as diagrams that can be used to explore the dynamic behaviour of a system or software application by showing the interactions between the objects or roles associated with lifelines and the messages that pass between lifelines (IBM, 2021). The Communication/Collaboration Diagram has 4 components and they are:

- Objects

The objects represent a class surrounded by a rectangle with underline.

- Actors

The actors represent consumers that use the system by interacting with it.

- Messages pathways

A message pathway is a connector between the roles and objects represented by connecting a simple line.

- Messages

A message is a communication between the roles and objects by conveying information from one instance, which is represented by a lifeline, to another instance in an interaction.

6.1. Steps involved in Communication/Collaboration diagram

a) Identifying the domain classes and creating object of it

Through the Expanded use case diagram of the join training course, we identified the domain classes and we found payment, courses and trainee.

Figure 3: finding and adding domain classes of Communication/Collaboration diagram

b) Identifying and adding the controller object

Controller object is a reference to those objects which sit between and controls the communication between UI, objects and represents the transfer/flow of information. The controller object is named the same as the name of the use case.

Figure 4: adding controller object to Communication/Collaboration diagram

c) Identifying and adding the boundary object

Boundary objects are those that manage the interaction/ flow of information between the system and actors. The boundary object is named the same as the name of the use case with UI attached at last.

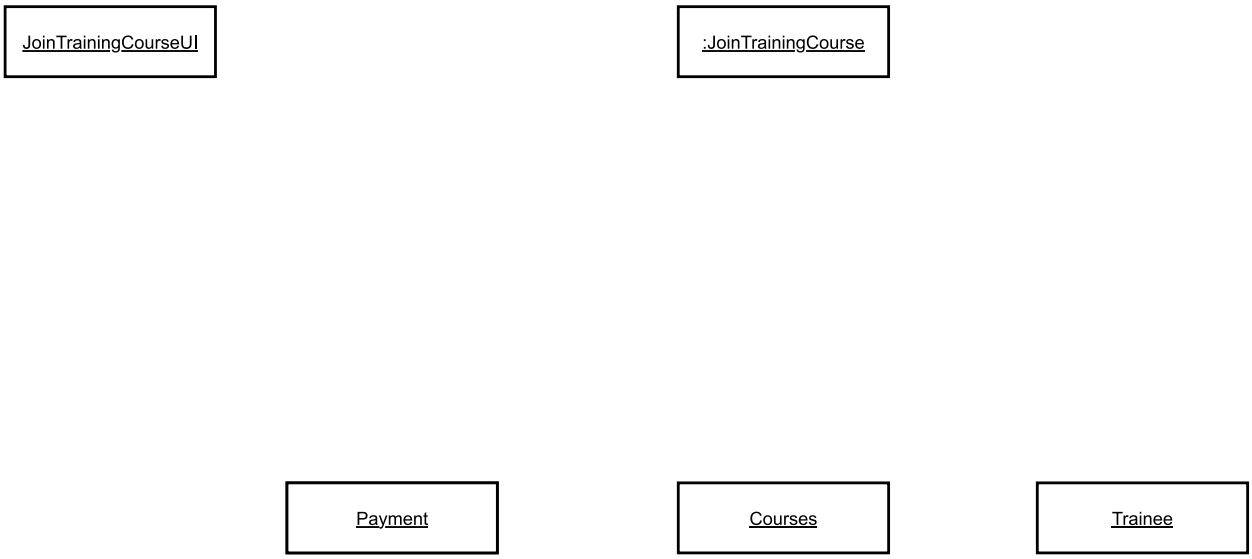


Figure 5: adding boundary class for Communication/Collaboration diagram

d) Adding Actor

Actors are the entities that interact with the system.

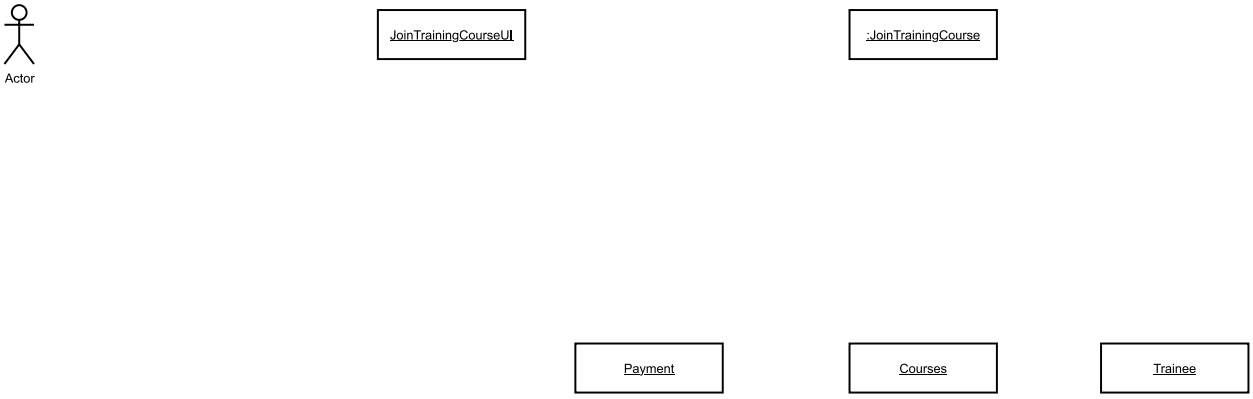


Figure 6: adding actor to Communication/Collaboration diagram

e) Adding Association

Now, we associate the actor, boundary object, control object and the object of the domain classes as follow:

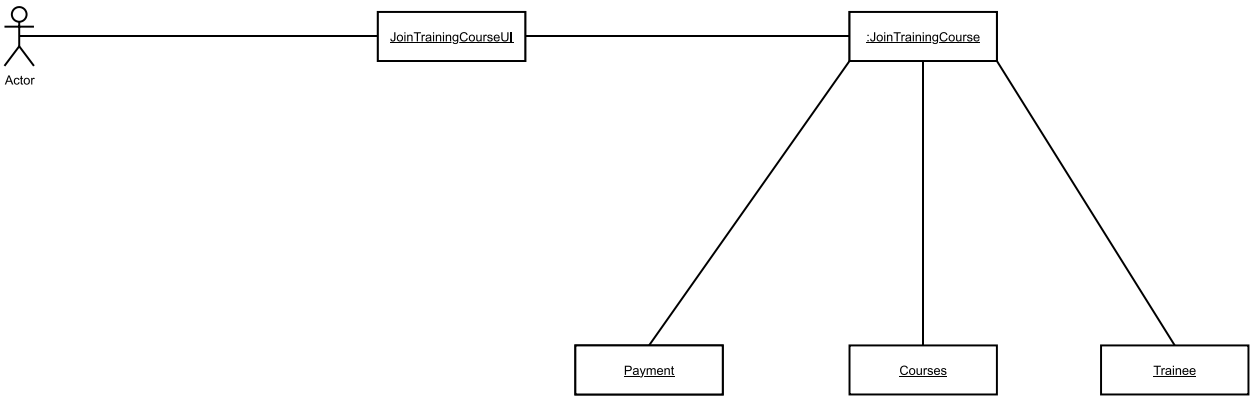


Figure 7: joining association of objects in Communication/Collaboration diagram

f) Adding messages to diagram

After associating all the factors in the diagram, it is time for adding how the data is flown between them by an arrow and message beside it.

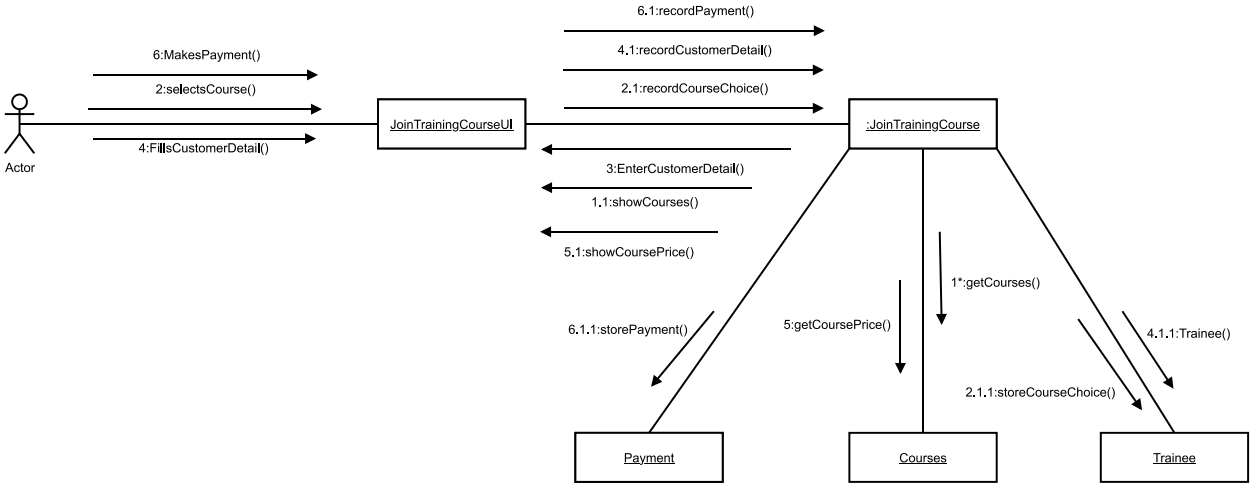


Figure 8: adding messages for Communication/Collaboration diagram

6.2. Final collaboration diagram

After completing all steps successfully, the final collaboration diagram of the use case "Join training course" is as follow:

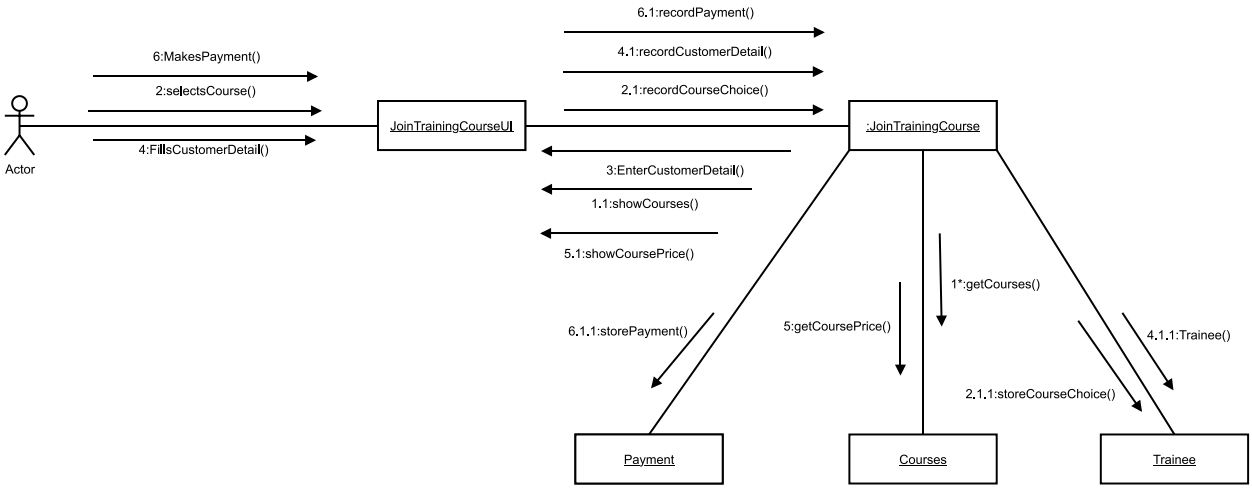


Figure 9: final Communication/Collaboration diagram

7. Sequence Diagram

Sequence diagrams are diagrams that illustrate the sequence of messages between a group of objects represented by lifelines, and the messages that they exchange over time during the interaction (IBM, 2021). Some of components are:

- Lifeline
- Message
- Flow of control
- Objects

7.1. Steps involved in sequence diagram

a) Identifying the domain classes and creating object of it

Through the Expanded use case diagram of the join training course, we identified the domain classes and we found card, courses and trainee. Its domain classes are identical to those found on the sequence diagram.

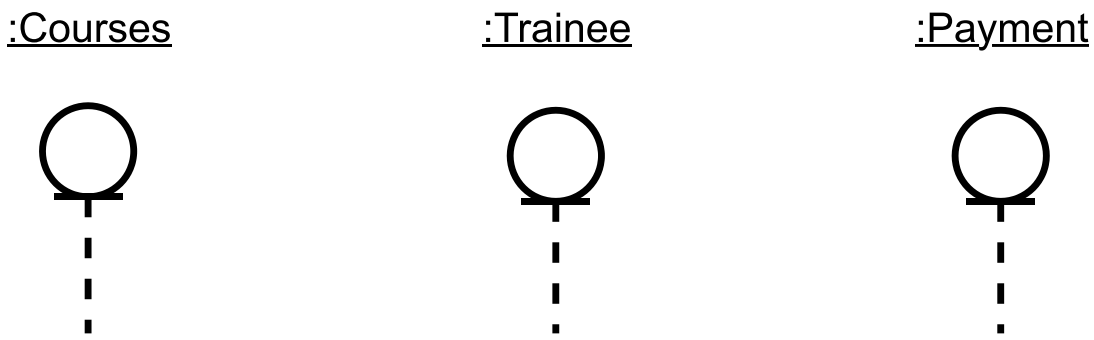


Figure 10: identifying and adding domain class in sequence diagram

b) Identifying and adding controller object lifeline

Controller lifeline is a reference to those objects which sit between and controls the communication between UI, objects and represents the transfer/flow of information. The lifeline object is named the same as the name of the use case.

Figure 11: adding controller object to sequence diagram

c) Identifying and adding boundary object lifeline

Boundary lifeline are those that manage the interaction/ flow of information between the system and actors. The boundary lifeline is named the same as the name of the use case with UI attached at last.

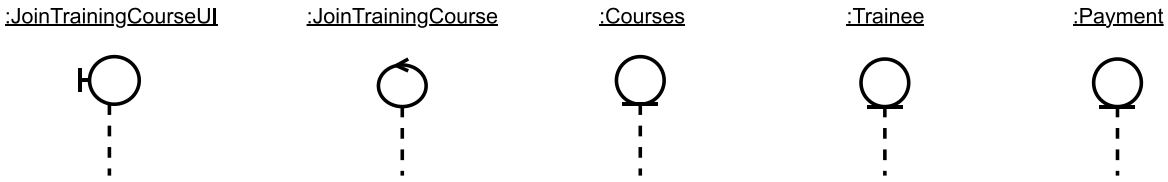


Figure 12: adding boundary object to sequence diagram

d) Drawing actor lifeline

Actors are the entities that interact with the system. The lifeline of the actor represents the interaction between the actor and the system.

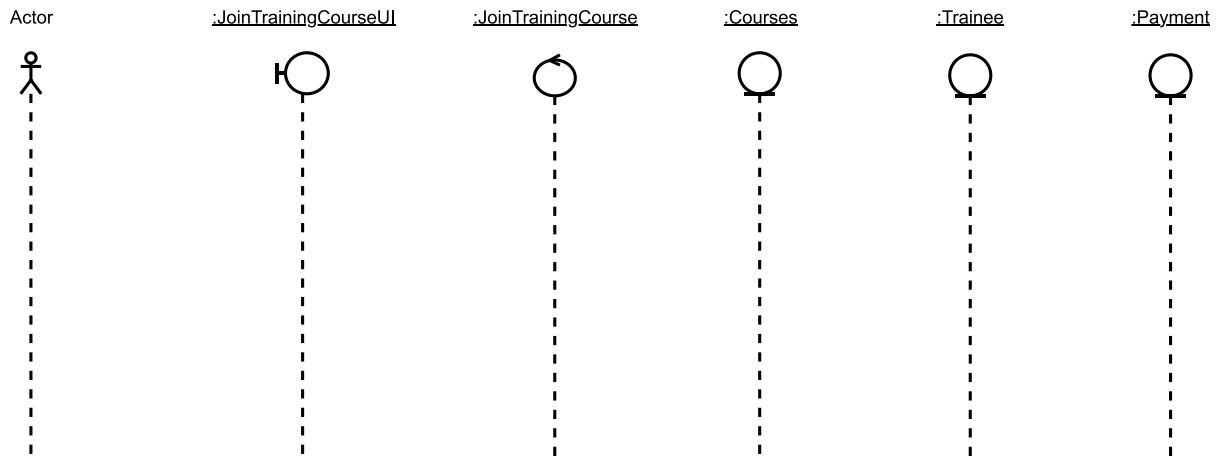


Figure 13: drawing actor lifeline

e) Determining activation period

Drawing the rectangular black shaded bars serially as how they are activated and how the interaction between objects is represented.

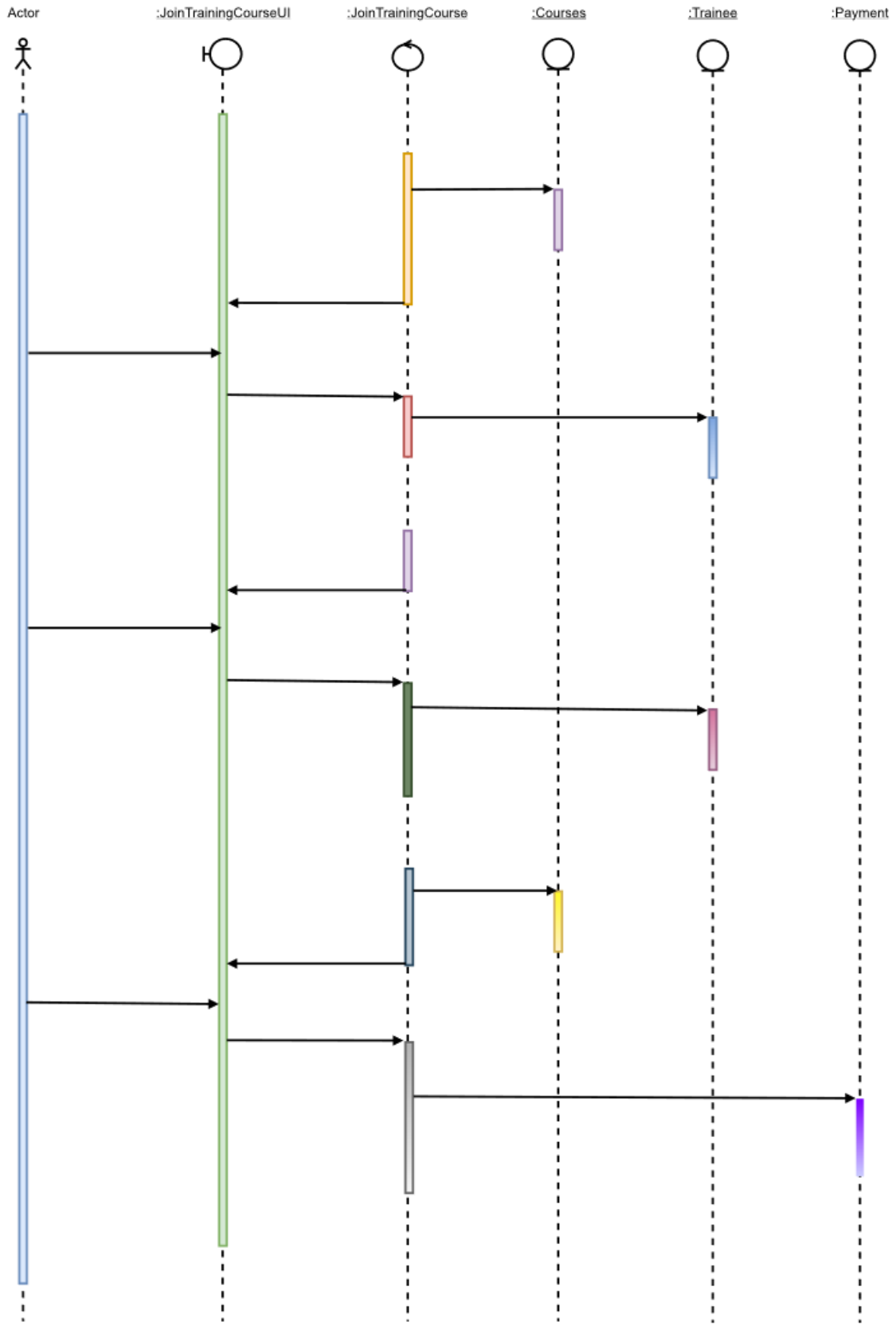

Actor
:JoinTrainingCourseUI
:JoinTrainingCourse
:Courses
:Trainee
:Payment

Figure 14: determining activation bar in lifeline

f) Adding messages to the diagram

Add messages to the diagram on the behaviour of calling one another for a specific operation.

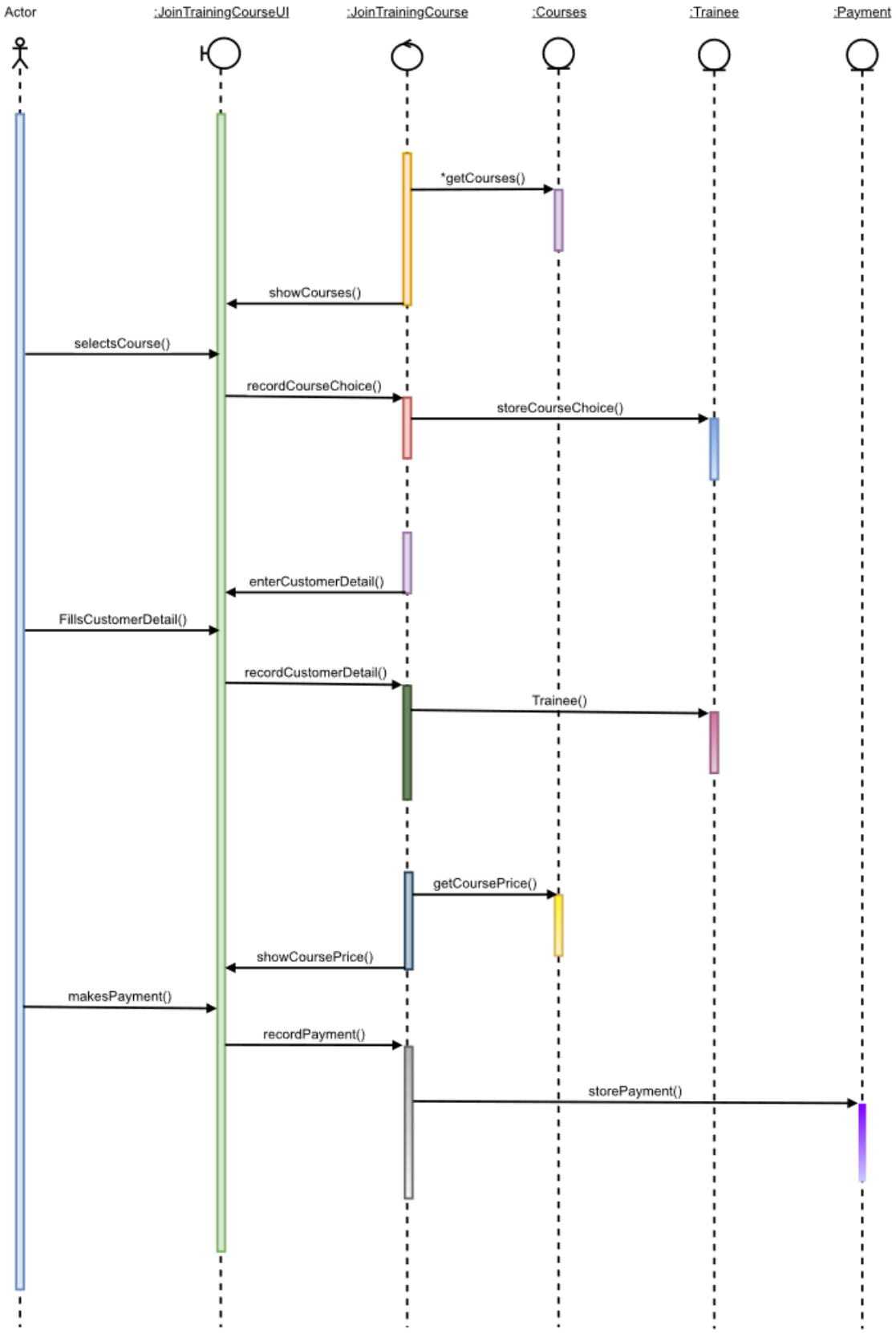

Actor
:JoinTrainingCourseUI
:JoinTrainingCourse
:Courses
:Trainee
:Payment
*getCourses()
showCourses()
selectsCourse()
recordCourseChoice()
storeCourseChoice()
enterCustomerDetail()
FillsCustomerDetail()
recordCustomerDetail()
Trainee()
getCoursePrice()
showCoursePrice()
makesPayment()
recordPayment()
storePayment()

Figure 15: adding messages on how objects are called

g) Adding functionality

Finally, any special functionality such as looping or other and representation of it by a rectangle covering the whole activation process.

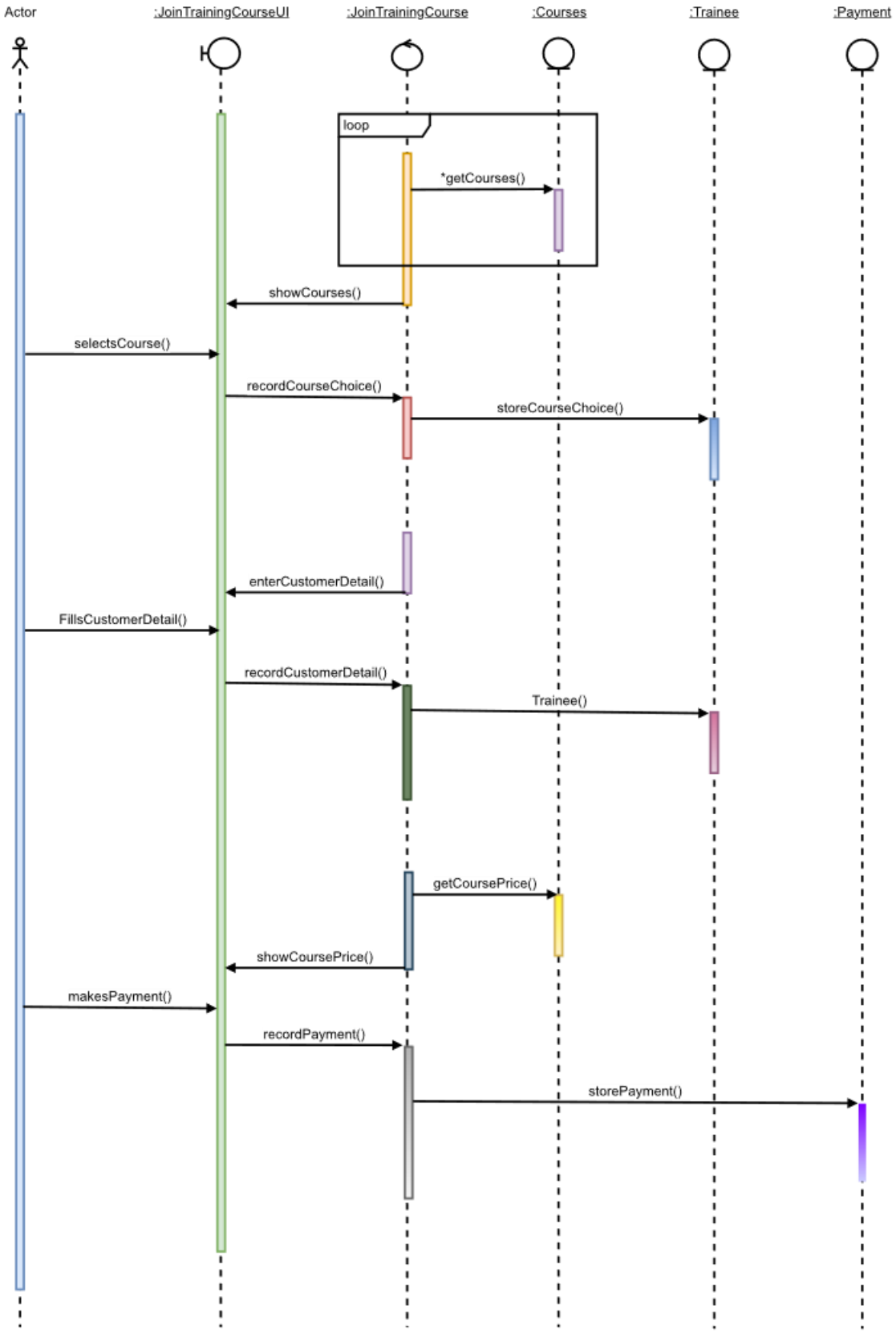

Actor
:JoinTrainingCourseUI
:JoinTrainingCourse
:Courses
:Trainee
:Payment
loop
*getCourses()
showCourses()
selectsCourse()
recordCourseChoice()
storeCourseChoice()
enterCustomerDetail()
FillsCustomerDetail()
recordCustomerDetail()
Trainee()
getCoursePrice()
showCoursePrice()
makesPayment()
recordPayment()
storePayment()

Figure 16: adding functionality to sequence diagram

7.2. Final sequence diagram

After completing all steps successfully, the final sequence diagram of the use case “Join training course” is as follow:

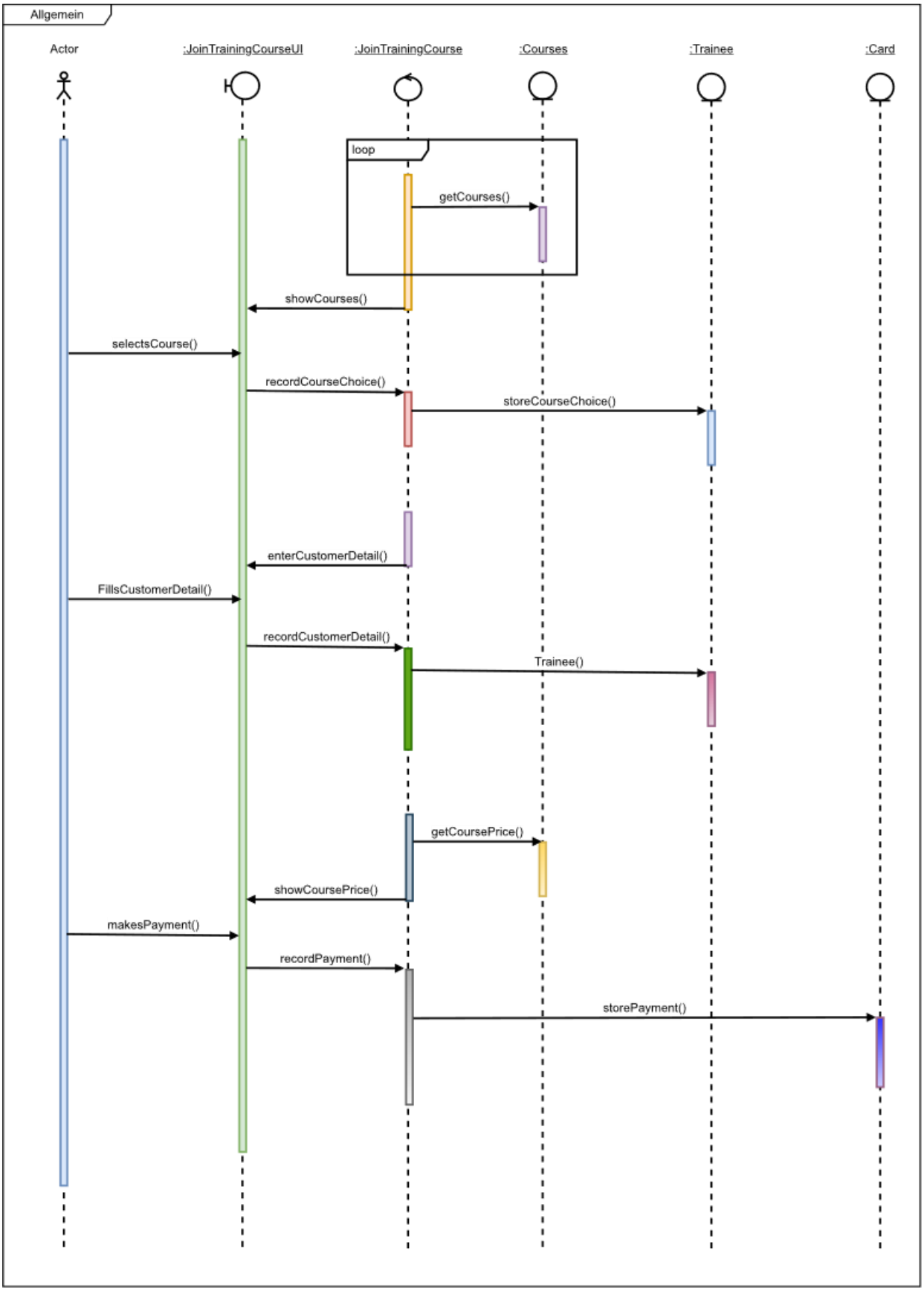

Allgemein
Actor
:JoinTrainingCourseUI
:JoinTrainingCourse
:Courses
:Trainee
:Card
loop
getCourses()
showCourses()
selectsCourse()
recordCourseChoice()
storeCourseChoice()
enterCustomerDetail()
FillsCustomerDetail()
recordCustomerDetail()
Trainee()
getCoursePrice()
showCoursePrice()
makesPayment()
recordPayment()
storePayment()

Figure 17: final sequence diagram

8. Class diagram

Class diagrams are a fundamental form of the object modelling process, which model the static structure of a system. Class diagrams represent the blueprints of the system and sub-system. Classes diagrams can be used to model the object-level composition of the system, to display the relationship between the objects and to describe the function of the objects and the services provided by them (IBM, 2021). What class diagram shows:

- Classes
- Attributes
- Methods/operations
- relation

8.1. Steps involved in class diagram

I. Finding domain classes

Listing out all the domain classes:

Use cases	Domains
Take member	Register, member, customer
Join training course	payment, courses, trainee, member, customer
Book cab	Cab, payment, journey, member
Hire a vehicle	Vehicle, payment, journey, member
Hire a driver	Driver, member
Rate the ride experience and vehicle's efficiency	Rate, member
Track the status of drivers	Driver, member
Payment	Payment, member
Confirm hire	Driver, staff, vehicle
Register vehicle	Vehicle, driver
Register staff	Register, staff, admin
Report preparation	Report, rate, payment, vehicles, journey
Announcement	Courses

Table 17: finding domain classes for class diagram

The domain classes were:

- Admin
- Cab
- Courses
- Customer
- Driver
- Journey
- Member
- Payment
- Rate
- Register
- Report
- Staff
- Trainee
- Vehicle

II. Drawing classes

Combining and displaying all the domain classes.

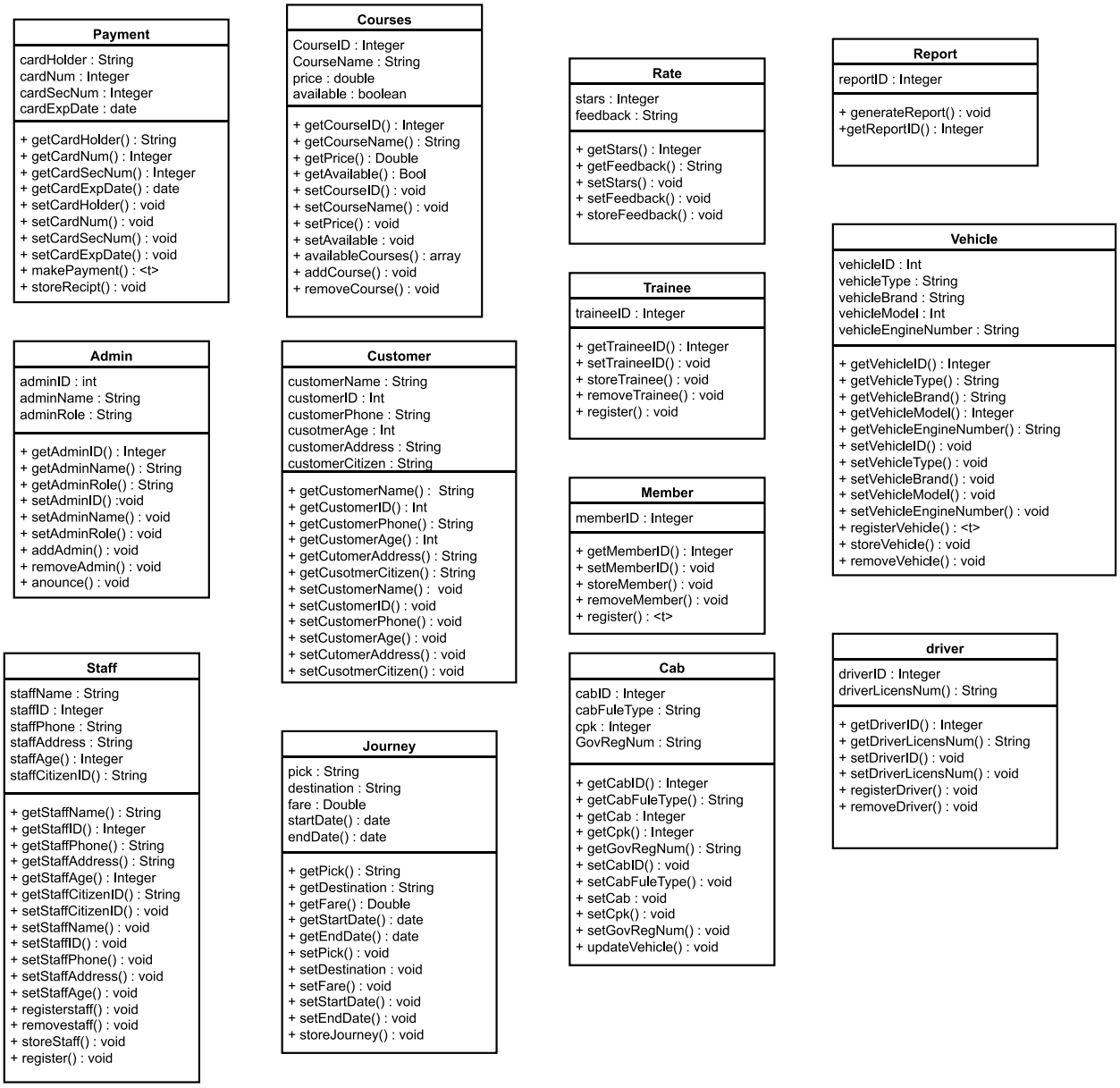


Figure 18: combining and displaying domain classes

III. Adding relationship

We can observe that the relationships of all domain classes can be multiple as described below:

a) Association

In the domain classes, the association occurs as domain classes communicate with each other. The association is represented by a simple line or a symboled line pointing to another.

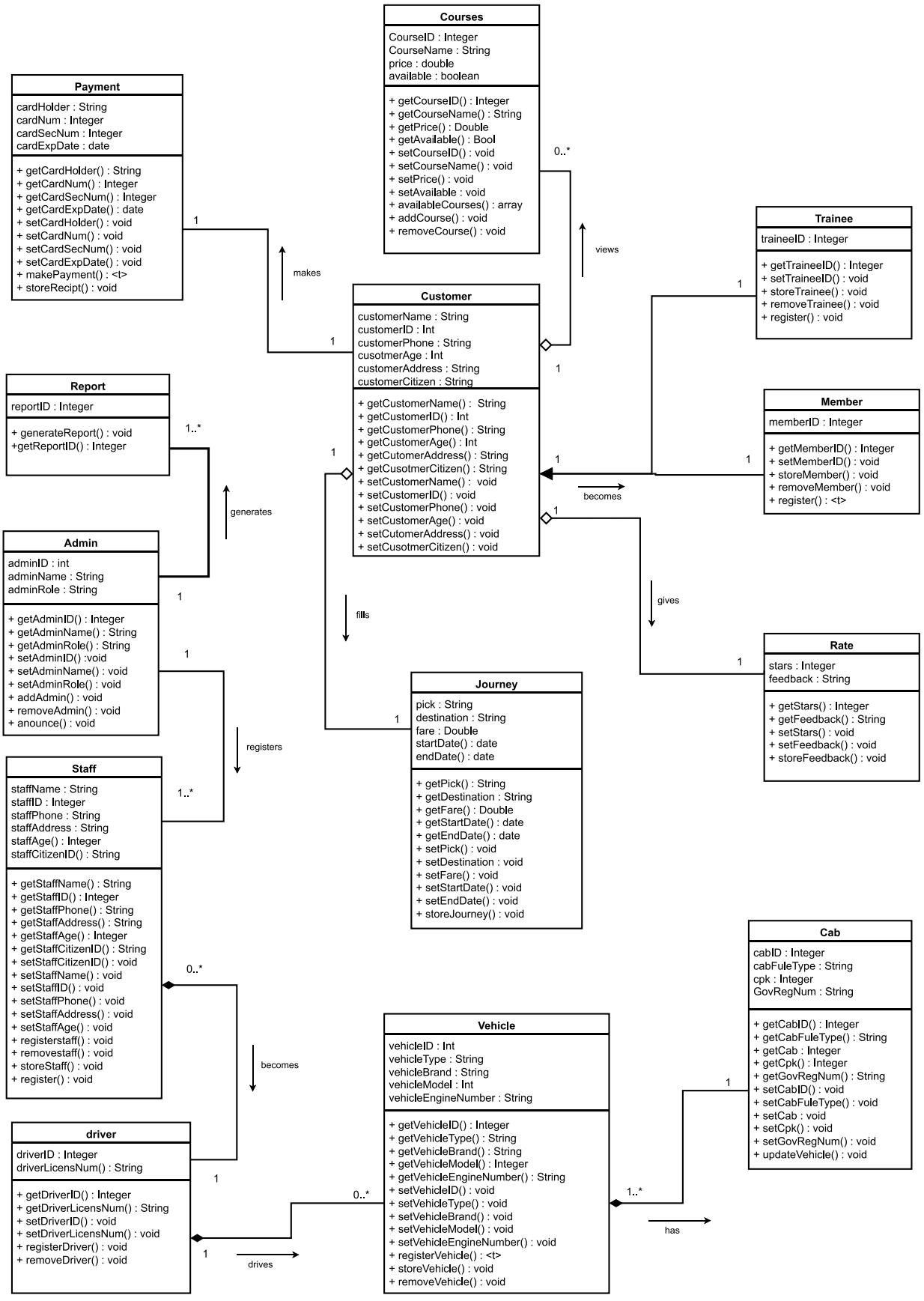

Courses
CourseID : Integer
CourseName : String
price : double
available : boolean
+ getCourseID() : Integer
+ getCourseName() : String
+ getPrice() : Double
+ getAvailable() : Bool
+ setCourseID() : void
+ setCourseName() : void
+ setPrice() : void
+ setAvailable : void
+ availableCourses() : array
+ addCourse() : void
+ removeCourse() : void
Payment
cardHolder : String
cardNum : Integer
cardSecNum : Integer
cardExpDate : date
+ getCardHolder() : String
+ getCardNum() : Integer
+ getCardSecNum() : Integer
+ getCardExpDate() : date
+ setCardHolder() : void
+ setCardNum() : void
+ setCardSecNum() : void
+ setCardExpDate() : void
+ makePayment() : <t>
+ storeRecipt() : void
Trainee
traineeID : Integer
+ getTraineeID() : Integer
+ setTraineeID() : void
+ storeTrainee() : void
+ removeTrainee() : void
+ register() : void
Customer
customerName : String
customerID : Int
customerPhone : String
cusotmerAge : Int
customerAddress : String
customerCitizen : String
+ getCustomerName() : String
+ getCustomerID() : Int
+ getCustomerPhone() : String
+ getCustomerAge() : Int
+ getCutomerAddress() : String
+ getCusotmerCitizen() : String
+ setCustomerName() : void
+ setCustomerID() : void
+ setCustomerPhone() : void
+ setCustomerAge() : void
+ setCutomerAddress() : void
+ setCusotmerCitizen() : void
Report
reportID : Integer
+ generateReport() : void
+getReportID() : Integer
Member
memberID : Integer
+ getMemberID() : Integer
+ setMemberID() : void
+ storeMember() : void
+ removeMember() : void
+ register() : <t>
Admin
adminID : int
adminName : String
adminRole : String
+ getAdminID() : Integer
+ getAdminName() : String
+ getAdminRole() : String
+ setAdminID() :void
+ setAdminName() : void
+ setAdminRole() : void
+ addAdmin() : void
+ removeAdmin() : void
+ anounce() : void
Rate
stars : Integer
feedback : String
+ getStars() : Integer
+ getFeedback() : String
+ setStars() : void
+ setFeedback() : void
+ storeFeedback() : void
Journey
pick : String
destination : String
fare : Double
startDate() : date
endDate() : date
+ getPick() : String
+ getDestination : String
+ getFare() : Double
+ getStartDate() : date
+ getEndDate() : date
+ setPick() : void
+ setDestination : void
+ setFare() : void
+ setStartDate() : void
+ setEndDate() : void
+ storeJourney() : void
Staff
staffName : String
staffID : Integer
staffPhone : String
staffAddress : String
staffAge() : Integer
staffCitizenID() : String
+ getStaffName() : String
+ getStaffID() : Integer
+ getStaffPhone() : String
+ getStaffAddress() : String
+ getStaffAge() : Integer
+ getStaffCitizenID() : String
+ setStaffCitizenID() : void
+ setStaffName() : void
+ setStaffID() : void
+ setStaffPhone() : void
+ setStaffAddress() : void
+ setStaffAge() : void
+ registerstaff() : void
+ removestaff() : void
+ storeStaff() : void
+ register() : void
Cab
cabID : Integer
cabFuleType : String
cpk : Integer
GovRegNum : String
+ getCabID() : Integer
+ getCabFuleType() : String
+ getCab : Integer
+ getCpk() : Integer
+ getGovRegNum() : String
+ setCabID() : void
+ setCabFuleType() : void
+ setCab : void
+ setCpk() : void
+ setGovRegNum() : void
+ updateVehicle() : void
Vehicle
vehicleID : Int
vehicleType : String
vehicleBrand : String
vehicleModel : Int
vehicleEngineNumber : String
+ getVehicleID() : Integer
+ getVehicleType() : String
+ getVehicleBrand() : String
+ getVehicleModel() : Integer
+ getVehicleEngineNumber() : String
+ setVehicleID() : void
+ setVehicleType() : void
+ setVehicleBrand() : void
+ setVehicleModel() : void
+ setVehicleEngineNumber() : void
+ registerVehicle() : <t>
+ storeVehicle() : void
+ removeVehicle() : void
driver
driverID : Integer
driverLicensNum() : String
+ getDriverID() : Integer
+ getDriverLicensNum() : String
+ setDriverID() : void
+ setDriverLicensNum() : void
+ registerDriver() : void
+ removeDriver() : void
makes
views
becomes
generates
fills
gives
registers
becomes
drives
has
1
0..*
1..*

Figure 19: drawing association in class diagram

- ***Aggregation***

A aggregation is a special type of association in which a group of objects are assembled or configured together to create a more complex object by defining a single point of control.

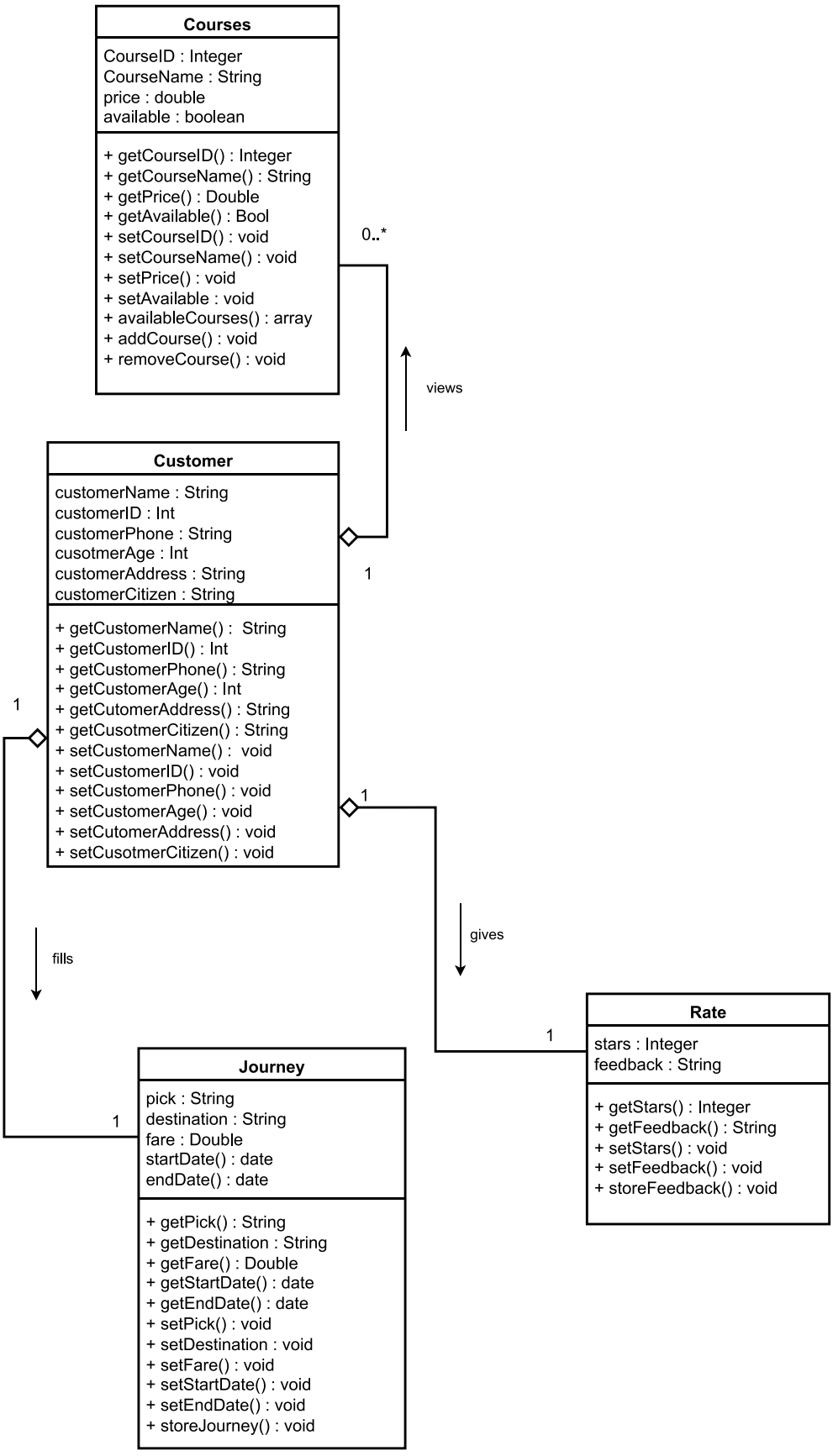

Courses
CourseID : Integer
CourseName : String
price : double
available : boolean
+ getCourseID() : Integer
+ getCourseName() : String
+ getPrice() : Double
+ getAvailable() : Bool
+ setCourseID() : void
+ setCourseName() : void
+ setPrice() : void
+ setAvailable : void
+ availableCourses() : array
+ addCourse() : void
+ removeCourse() : void
0..*
views
Customer
customerName : String
customerID : Int
customerPhone : String
cusotmerAge : Int
customerAddress : String
customerCitizen : String
+ getCustomerName() : String
+ getCustomerID() : Int
+ getCustomerPhone() : String
+ getCustomerAge() : Int
+ getCutomerAddress() : String
+ getCusotmerCitizen() : String
+ setCustomerName() : void
+ setCustomerID() : void
+ setCustomerPhone() : void
+ setCustomerAge() : void
+ setCutomerAddress() : void
+ setCusotmerCitizen() : void
1
1
1
1
fills
gives
Journey
pick : String
destination : String
fare : Double
startDate() : date
endDate() : date
+ getPick() : String
+ getDestination : String
+ getFare() : Double
+ getStartDate() : date
+ getEndDate() : date
+ setPick() : void
+ setDestination : void
+ setFare() : void
+ setStartDate() : void
+ setEndDate() : void
+ storeJourney() : void
Rate
stars : Integer
feedback : String
+ getStars() : Integer
+ getFeedback() : String
+ setStars() : void
+ setFeedback() : void
+ storeFeedback() : void

Figure 20: aggregation in class diagram

- ***Composition***

It is a part of the association that depicts dependency between a composite and its part, and if a composite is deleted then its part is made useless as they share a relation between them as if they are 2 sides of a coin.

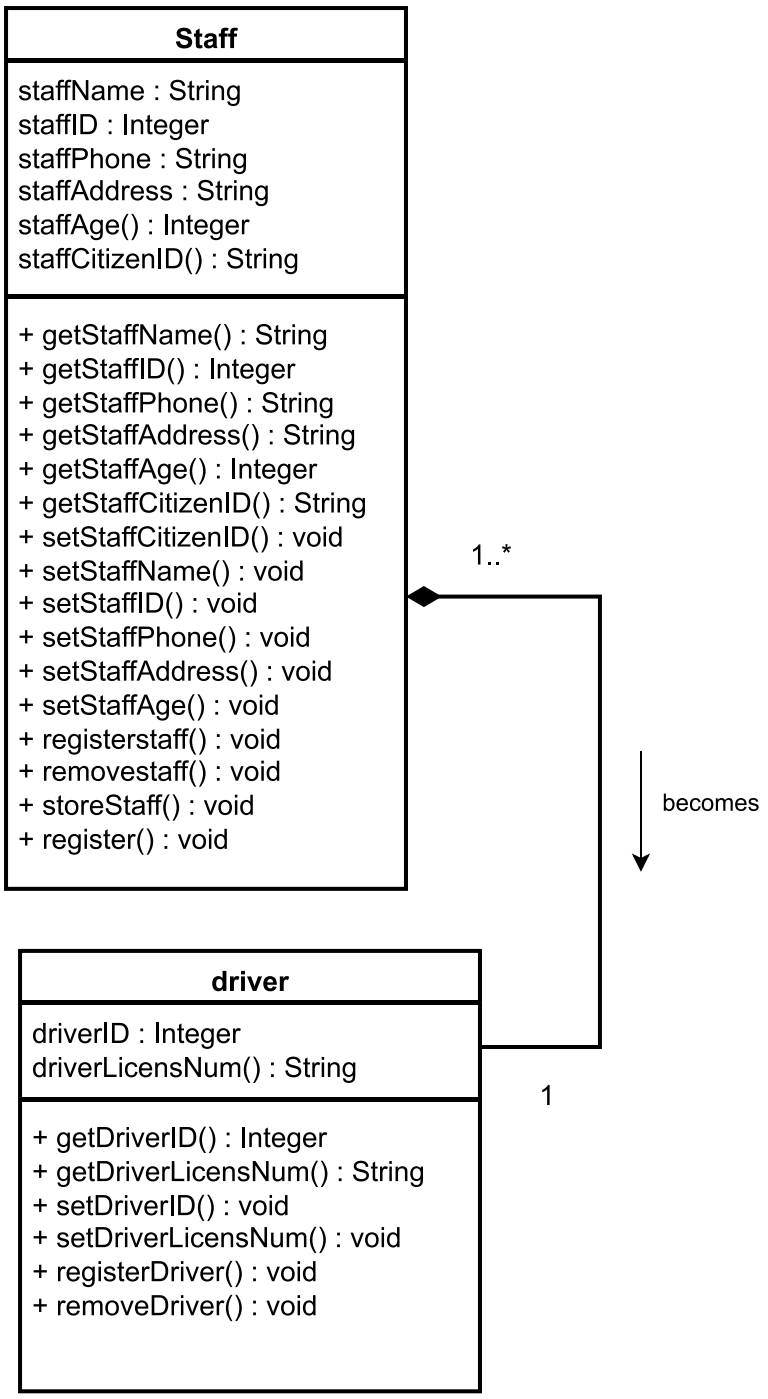


Figure 21: composition in class diagram

b) Inheritance

Inheritance is a concept where A class consists of a collection of attributes and methods that determine the state and behaviour of its instances and is passed onto B class and it is able to use all methods that are in A class as if inheriting.

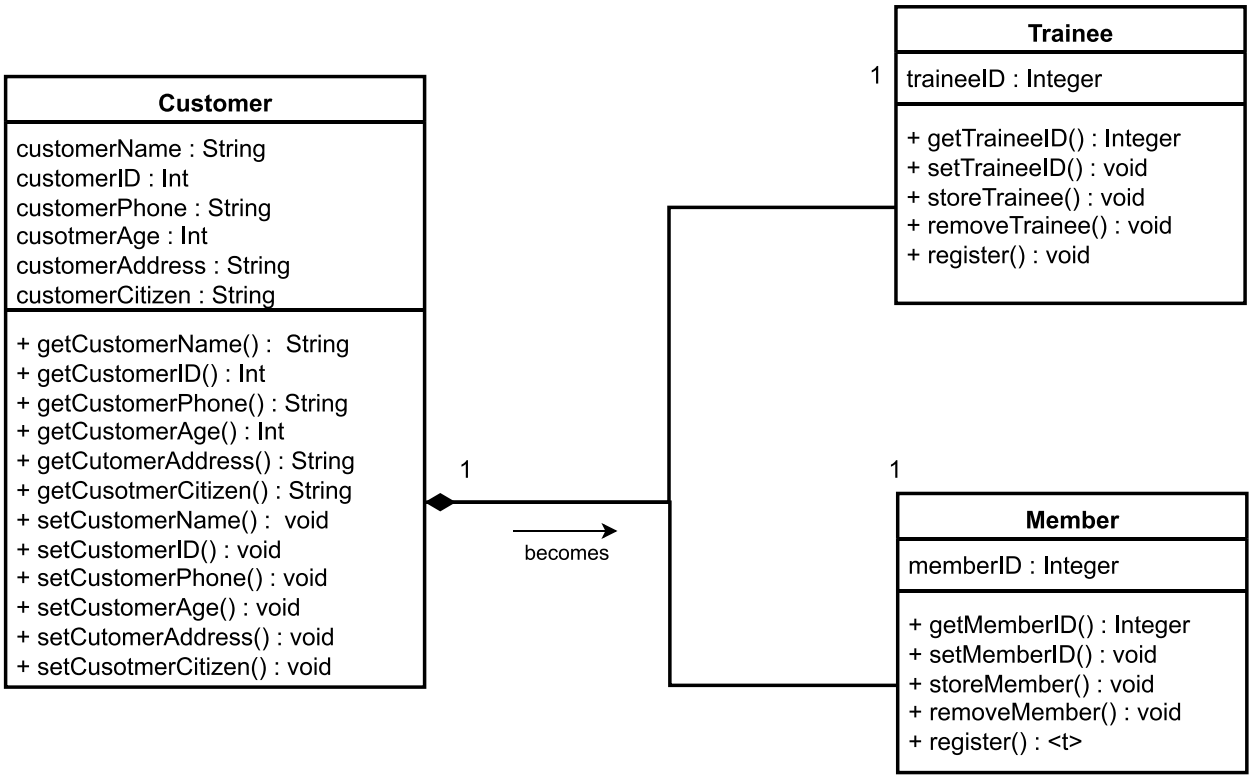


Figure 22: example 1 for inheritance in class diagram

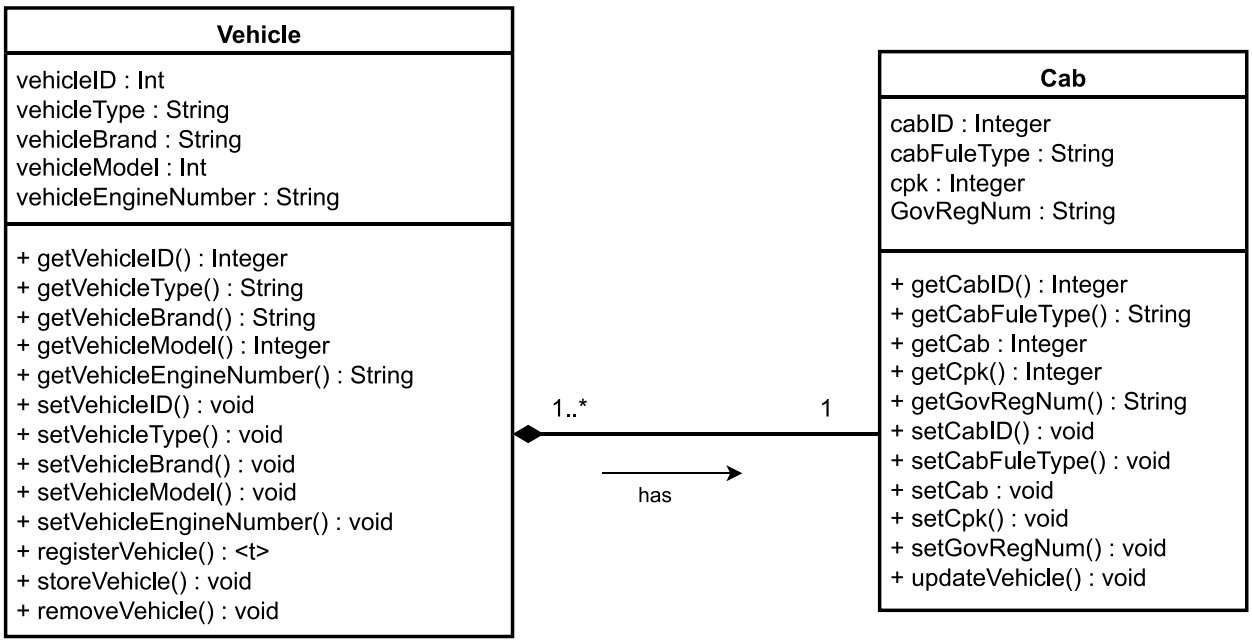


Figure 23: example 2 for inheritance in class diagram

8.2. Final class diagram

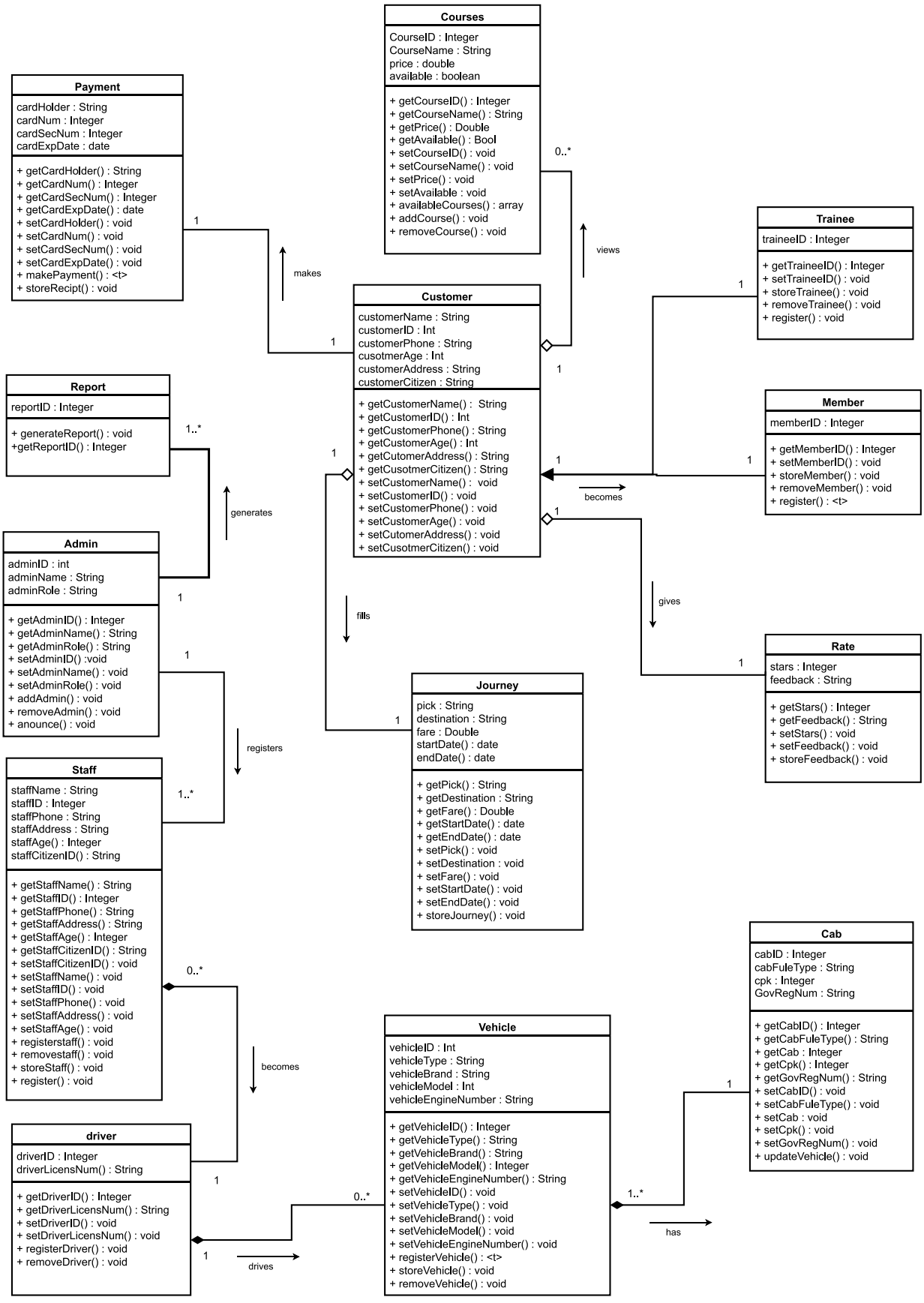

Courses
CourseID : Integer
CourseName : String
price : double
available : boolean
+ getCourseID() : Integer
+ getCourseName() : String
+ getPrice() : Double
+ getAvailable() : Bool
+ setCourseID() : void
+ setCourseName() : void
+ setPrice() : void
+ setAvailable : void
+ availableCourses() : array
+ addCourse() : void
+ removeCourse() : void
Payment
cardHolder : String
cardNum : Integer
cardSecNum : Integer
cardExpDate : date
+ getCardHolder() : String
+ getCardNum() : Integer
+ getCardSecNum() : Integer
+ getCardExpDate() : date
+ setCardHolder() : void
+ setCardNum() : void
+ setCardSecNum() : void
+ setCardExpDate() : void
+ makePayment() : <t>
+ storeRecipt() : void
Trainee
traineeID : Integer
+ getTraineeID() : Integer
+ setTraineeID() : void
+ storeTrainee() : void
+ removeTrainee() : void
+ register() : void
Customer
customerName : String
customerID : Int
customerPhone : String
cusotmerAge : Int
customerAddress : String
customerCitizen : String
+ getCustomerName() : String
+ getCustomerID() : Int
+ getCustomerPhone() : String
+ getCustomerAge() : Int
+ getCutomerAddress() : String
+ getCusotmerCitizen() : String
+ setCustomerName() : void
+ setCustomerID() : void
+ setCustomerPhone() : void
+ setCustomerAge() : void
+ setCutomerAddress() : void
+ setCusotmerCitizen() : void
Report
reportID : Integer
+ generateReport() : void
+getReportID() : Integer
Member
memberID : Integer
+ getMemberID() : Integer
+ setMemberID() : void
+ storeMember() : void
+ removeMember() : void
+ register() : <t>
Admin
adminID : int
adminName : String
adminRole : String
+ getAdminID() : Integer
+ getAdminName() : String
+ getAdminRole() : String
+ setAdminID() :void
+ setAdminName() : void
+ setAdminRole() : void
+ addAdmin() : void
+ removeAdmin() : void
+ anounce() : void
Rate
stars : Integer
feedback : String
+ getStars() : Integer
+ getFeedback() : String
+ setStars() : void
+ setFeedback() : void
+ storeFeedback() : void
Journey
pick : String
destination : String
fare : Double
startDate() : date
endDate() : date
+ getPick() : String
+ getDestination : String
+ getFare() : Double
+ getStartDate() : date
+ getEndDate() : date
+ setPick() : void
+ setDestination : void
+ setFare() : void
+ setStartDate() : void
+ setEndDate() : void
+ storeJourney() : void
Staff
staffName : String
staffID : Integer
staffPhone : String
staffAddress : String
staffAge() : Integer
staffCitizenID() : String
+ getStaffName() : String
+ getStaffID() : Integer
+ getStaffPhone() : String
+ getStaffAddress() : String
+ getStaffAge() : Integer
+ getStaffCitizenID() : String
+ setStaffCitizenID() : void
+ setStaffName() : void
+ setStaffID() : void
+ setStaffPhone() : void
+ setStaffAddress() : void
+ setStaffAge() : void
+ registerstaff() : void
+ removestaff() : void
+ storeStaff() : void
+ register() : void
Cab
cabID : Integer
cabFuleType : String
cpk : Integer
GovRegNum : String
+ getCabID() : Integer
+ getCabFuleType() : String
+ getCab : Integer
+ getCpk() : Integer
+ getGovRegNum() : String
+ setCabID() : void
+ setCabFuleType() : void
+ setCab : void
+ setCpk() : void
+ setGovRegNum() : void
+ updateVehicle() : void
Vehicle
vehicleID : Int
vehicleType : String
vehicleBrand : String
vehicleModel : Int
vehicleEngineNumber : String
+ getVehicleID() : Integer
+ getVehicleType() : String
+ getVehicleBrand() : String
+ getVehicleModel() : Integer
+ getVehicleEngineNumber() : String
+ setVehicleID() : void
+ setVehicleType() : void
+ setVehicleBrand() : void
+ setVehicleModel() : void
+ setVehicleEngineNumber() : void
+ registerVehicle() : <t>
+ storeVehicle() : void
+ removeVehicle() : void
driver
driverID : Integer
driverLicensNum() : String
+ getDriverID() : Integer
+ getDriverLicensNum() : String
+ setDriverID() : void
+ setDriverLicensNum() : void
+ registerDriver() : void
+ removeDriver() : void
views
makes
generates
becomes
fills
gives
registers
becomes
drives
has
1
0..*
1..*

Figure 24: final class diagram

9. Further development

As previously stated, the methodology used to develop the software is the water flow methodology. Waterfall methodology is a traditional software development methodology, which approaches the development of software from a linear, sequential perspective. The main advantage of using the waterfall methodology is the clear and structured approach to software development, which is crucial to the management of complex projects. The reason for choosing this methodology was that the project has well-defined requirements and a clear understanding of the project scope.

At this point, the decision to use the MYSQL database to store, manage and manipulate necessary data such as personal details of customers, drivers and staff, details about vehicles and financial transactions. It uses Structured Query Language (SQL) for the data management and manipulation. MySql is a scalable, secure, and reliable data management platform that allows to manage large volumes of data. MySql is based on relational database management system (RDBMS) and it stores and provides access to data points that are related to each other. In order to assess the relationship between customer, service, driver, vehicle and transit as which customer had taken which type of service and which driver had fulfilled the service with which vehicle and what was the transit detail, we needed the relation between customer, service, driver, vehicle and transit. For GUI design of the software, the user interfaces (design of the user interface and vice versa) and prototypes (concepts of how the system is going to be triggered to another and how it will run behind the system) are designed and developed using Figma. Figma is an innovative design tool used by developers for the development of UI and prototypes. It has a robust collaboration system in which multiple developers can simultaneously design for a productivity boost while remaining open-source for the wider community. Java was chosen as the core programming language for the reasons of robustness in the domain of strong static typing, multi-threaded performance and support if a problem arises during the development of the software. Java is a platform independent, object-oriented and statically typed language. Because the Java code could be compiled to suit multiple platforms making the development and deployment of the

software much easier. Java has a vast library and framework suite which makes it very easy to develop any spectrum of software and applications. The focus was selected for the front-end and UI of software HTML, CSS and JavaScript were chosen the focus on how the front-end is going to look when accessed from the web on a mobile and the same interface is going to be used for the mobile application. In order for that to be achieved, we are going to use Java's external library, JxBrowser to achieve it. Docker was chosen for the virtual development environment because of its portability(it provides an environment that is consistent across different machines and platforms, ensuring that the Java application will run in the same way across all the machines), isolation(an isolated environment for the Java application, thereby reducing the risk of conflicts with other applications or libraries on the host machine) and dependency management(it eases the management of dependencies for the Java application as they can package all the required libraries and dependencies inside the container image, making it easier to deploy and maintain the application). Docker is a containerization platform that makes it easy to package applications and dependencies into portable containers that can be deployed and run on any system without any additional setup or configuration. Finally, Cause Effect Graphing and Path Coverage were selected for the testing methodology. Under Cause Effect Graphing, the relationship between logical input and corresponding actions is checked to ensure that it is properly executed in the expected sequence. For Path Coverage, the tester writes the unit tests to execute as many paths as possible through the program's control flow to identify broken, redundant or inefficient paths for bug detection. After the testing is complete and the positive results are obtained, we then proceeded further to make a user manual and document the features and code of the system in detail. after that, the software will be finished and ready to be shipped to the market for customers.

10. Prototype for application

A prototype is an early version of the implementation of concepts that are going to be developed and deployed on software. It also provides many advantages including being flexible enough to allow for the developers/designers to change the structure if they feel a bit dissatisfied with it. The prototype for the software was compleated as image below:

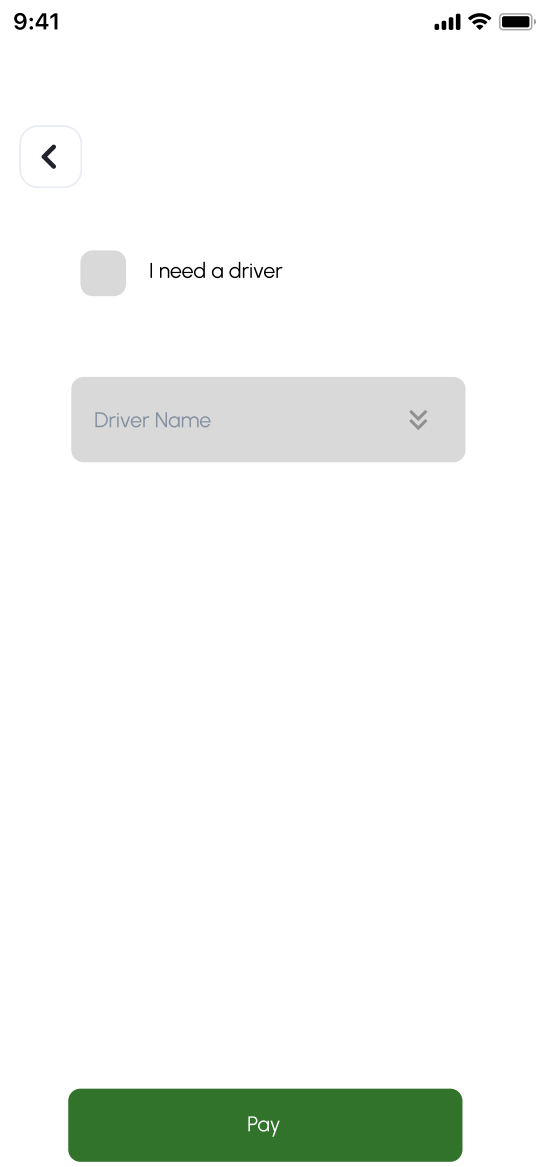


Figure 25: frame asking if customer needs a driver

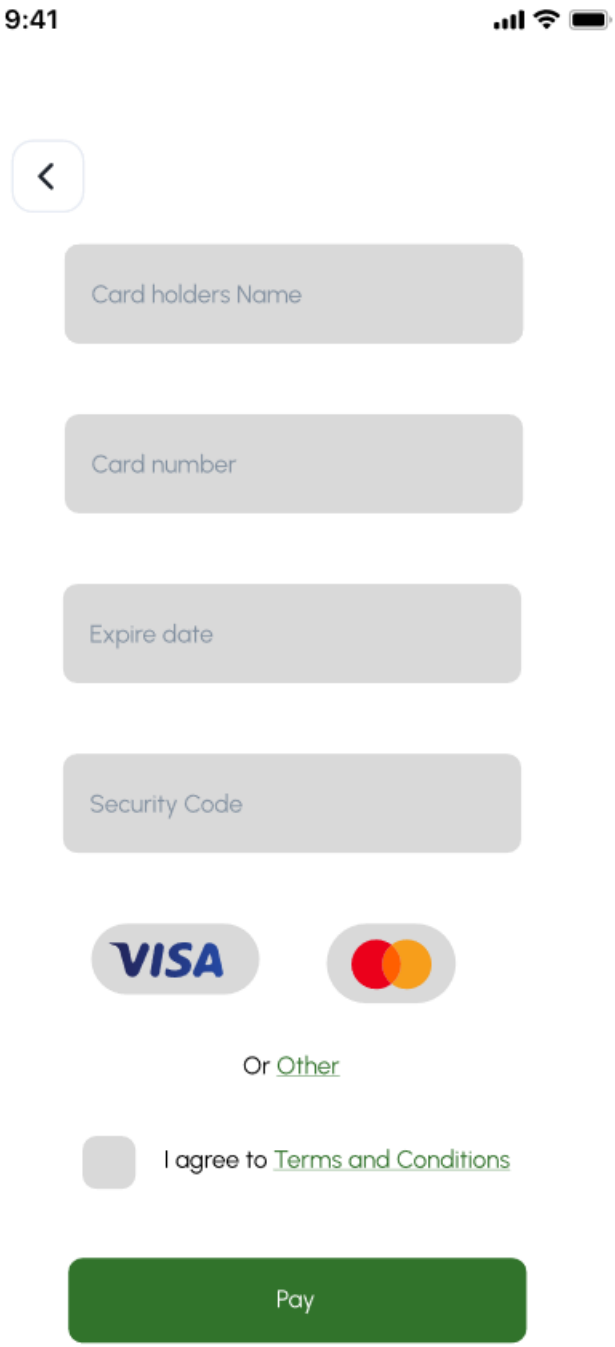


Figure 26: frame of asking for card details to customer

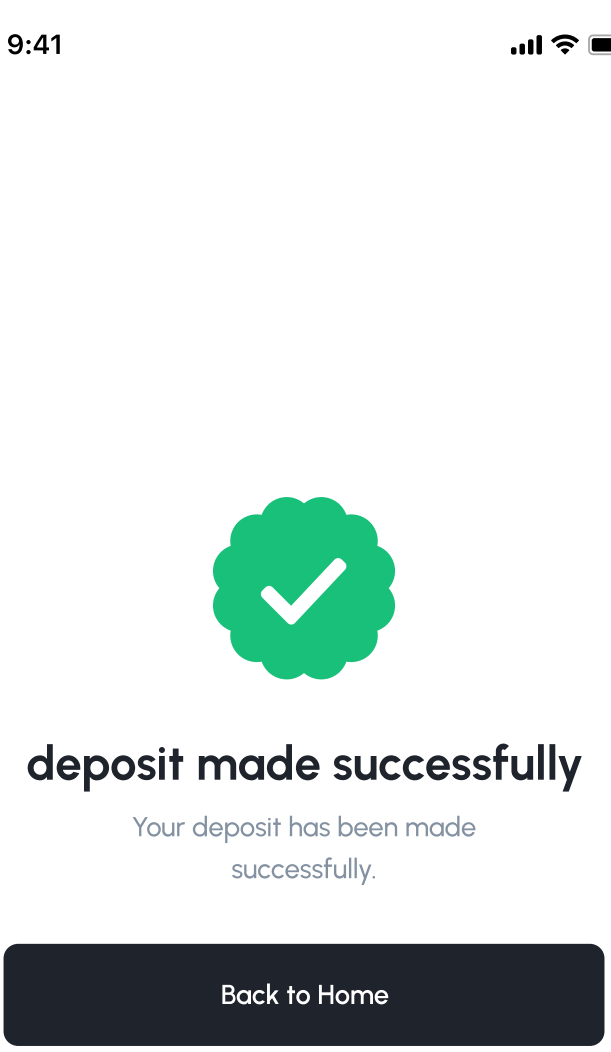


Figure 27: frame showing deposit has been made

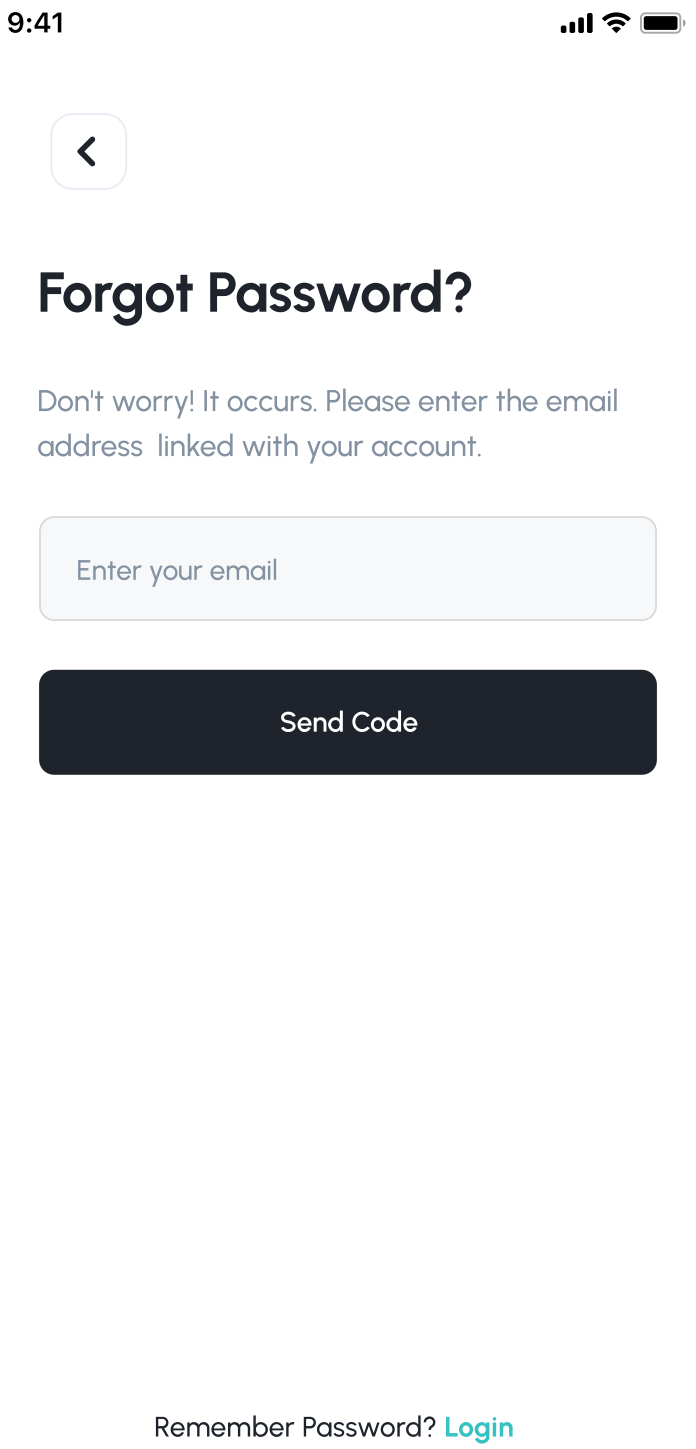


Figure 28: frame asking password when customer forgot password

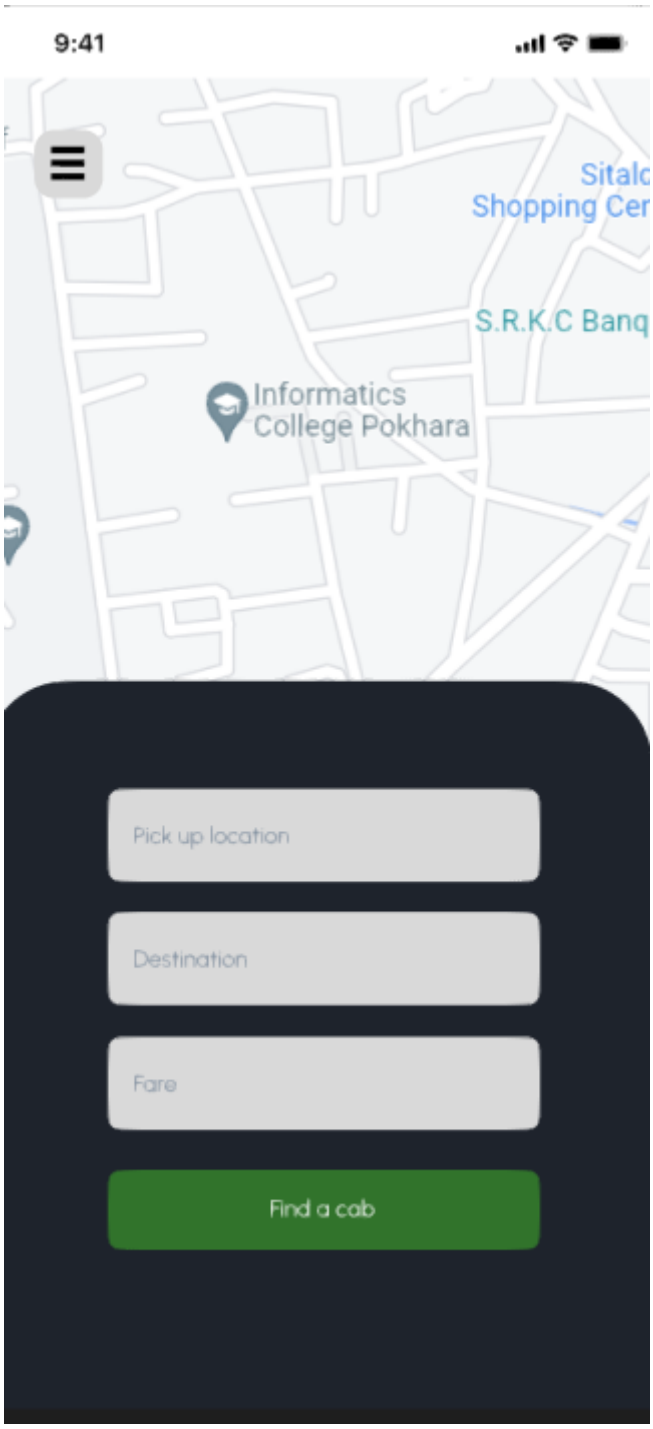


Figure 29: landing page of software

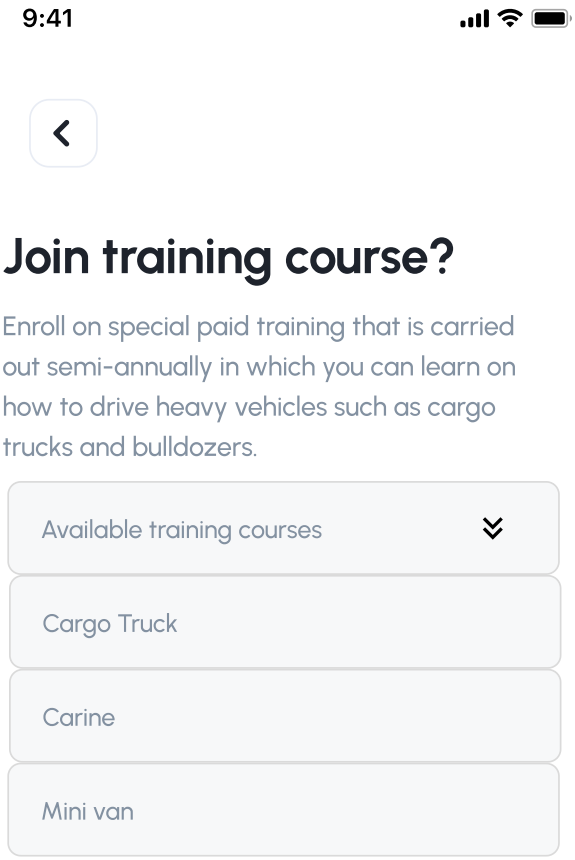


Figure 30: view available courses

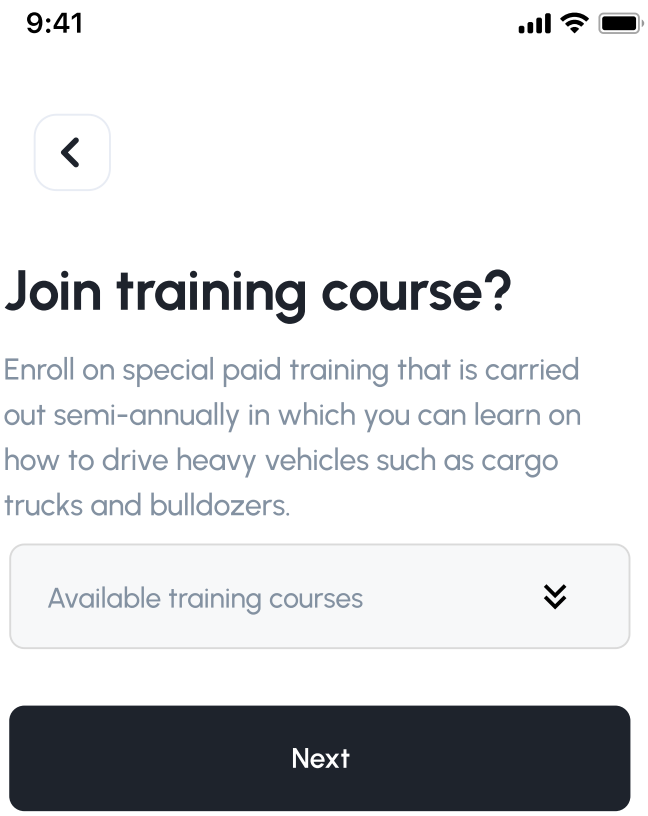


Figure 31: select a course

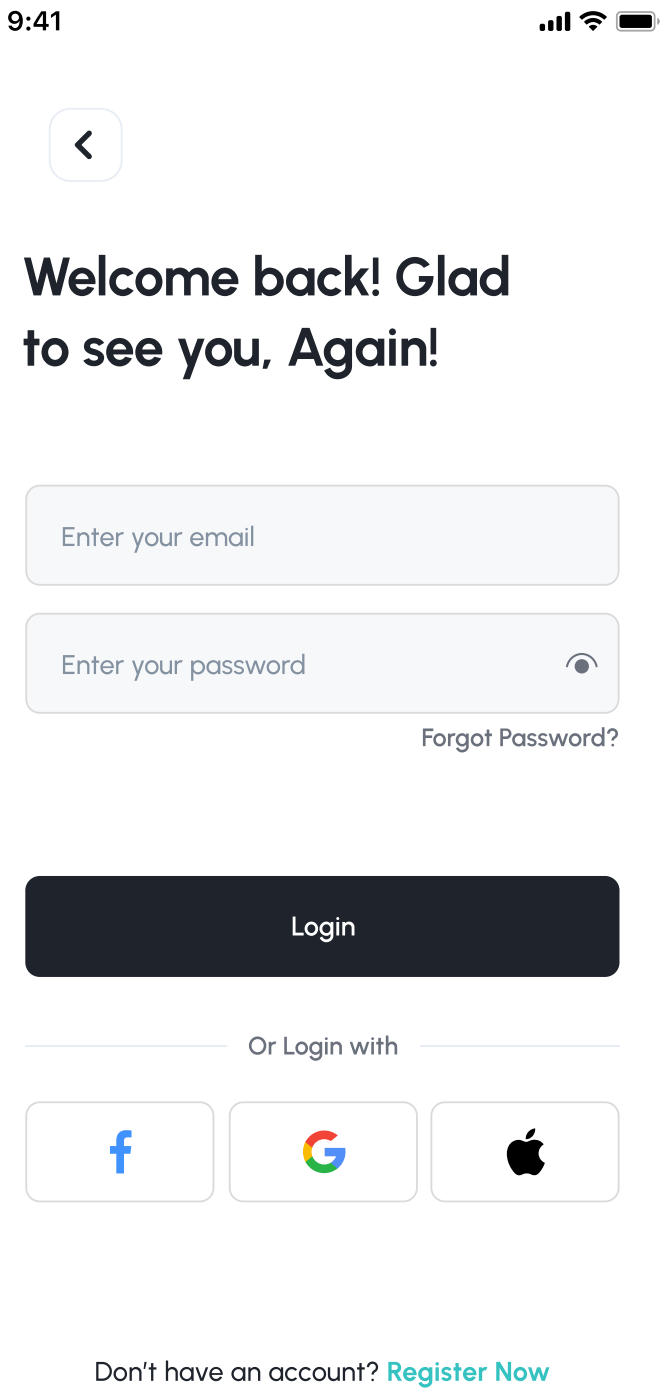


Figure 32: login frame

Figure 33: admin dashboard

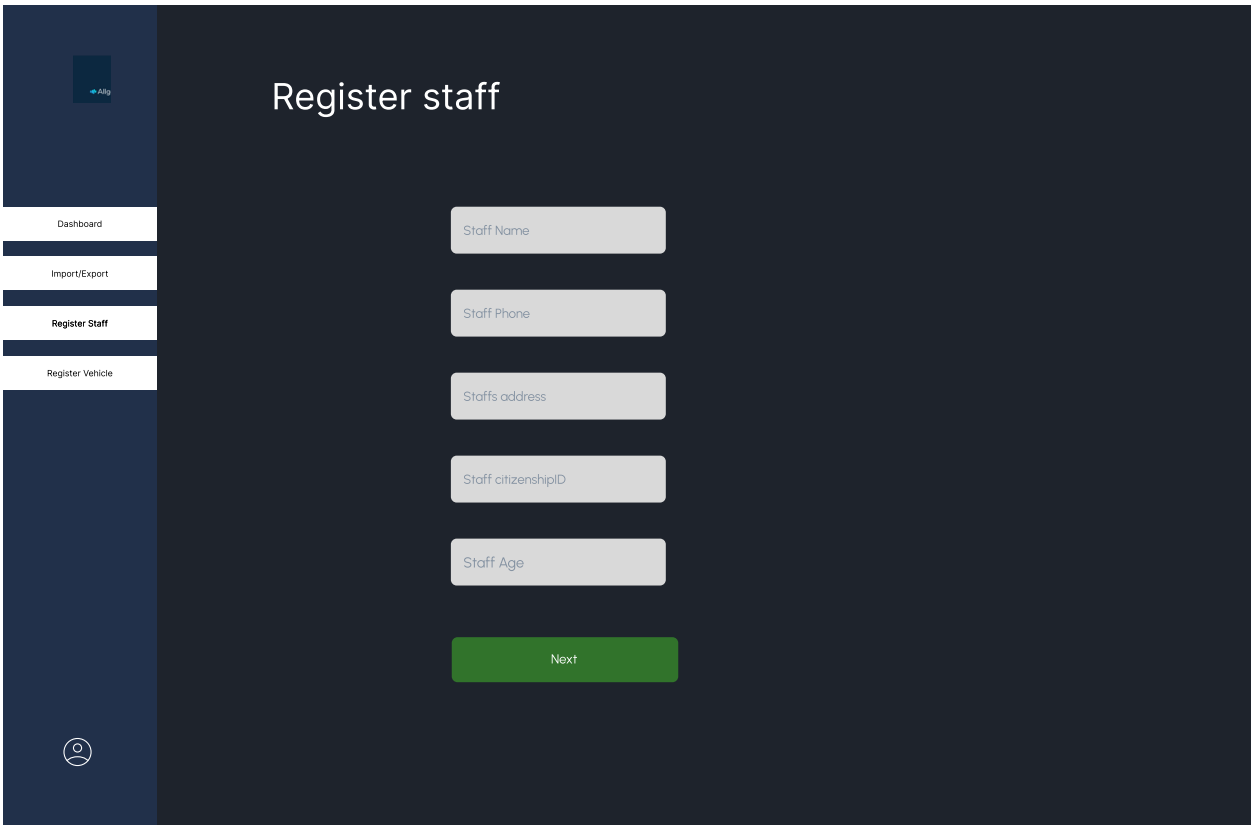


Figure 34: register staff webpage

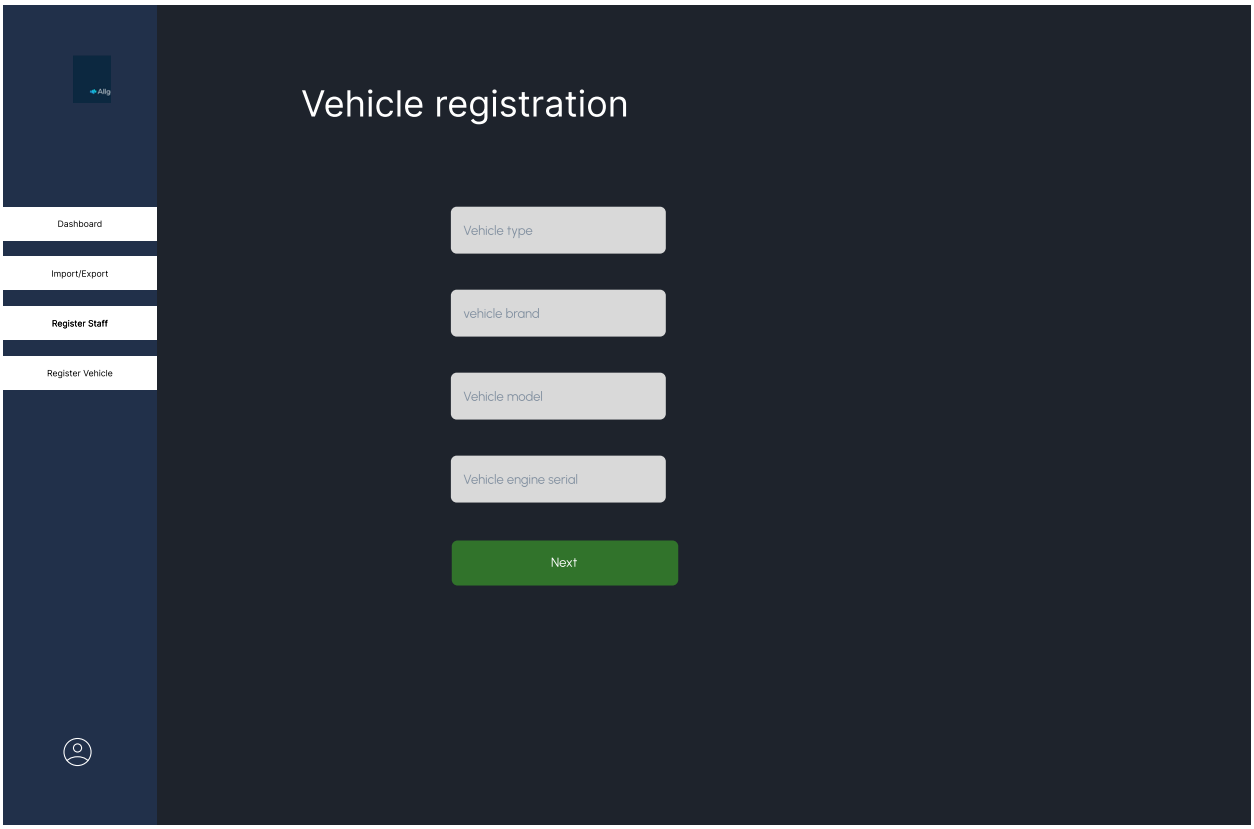


Figure 35: register vehicle page

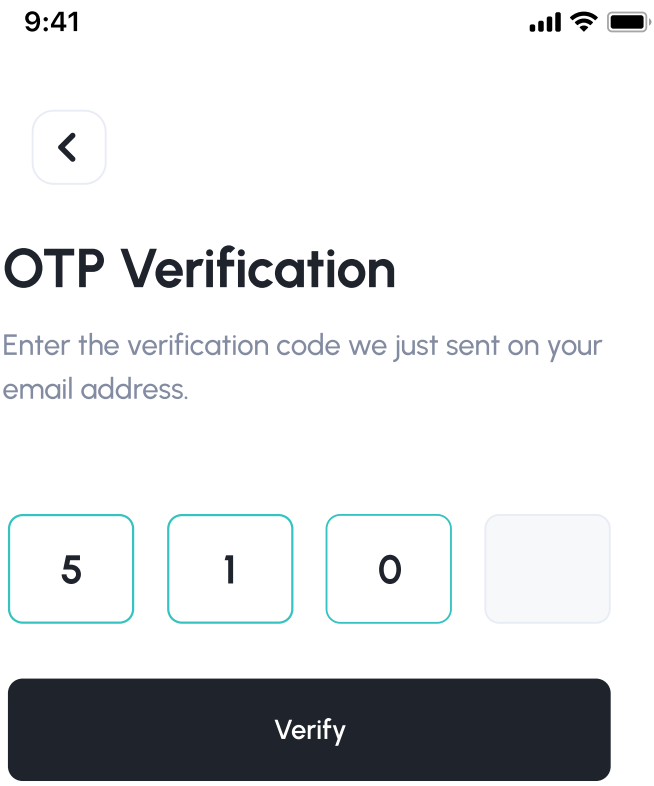


Figure 36: OTP verification frame

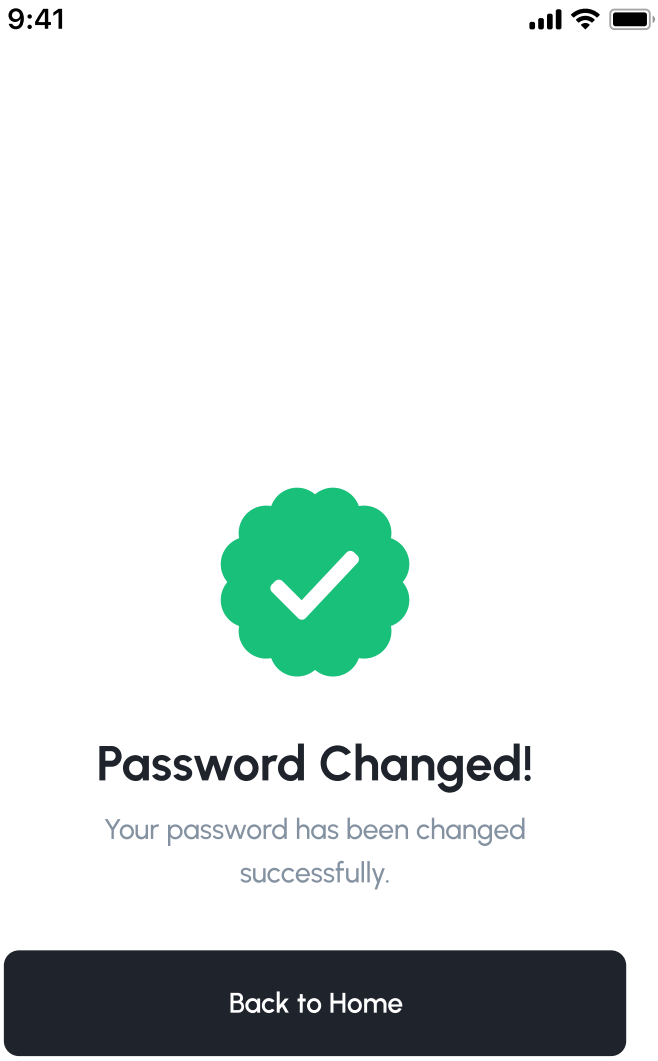


Figure 37: password changed successfully

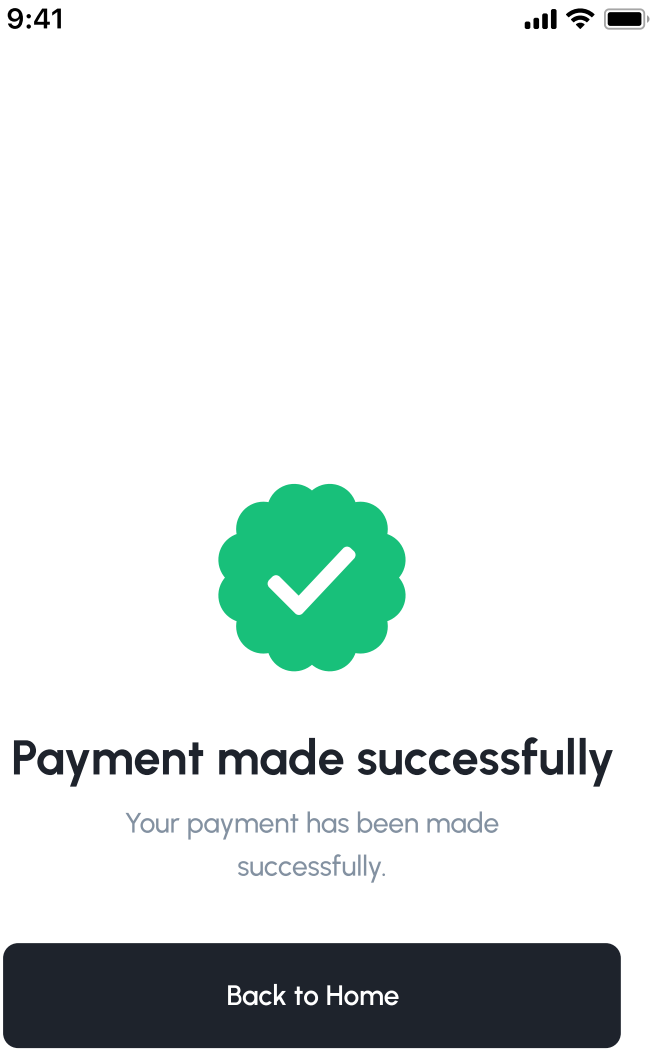


Figure 38: payment made successfully

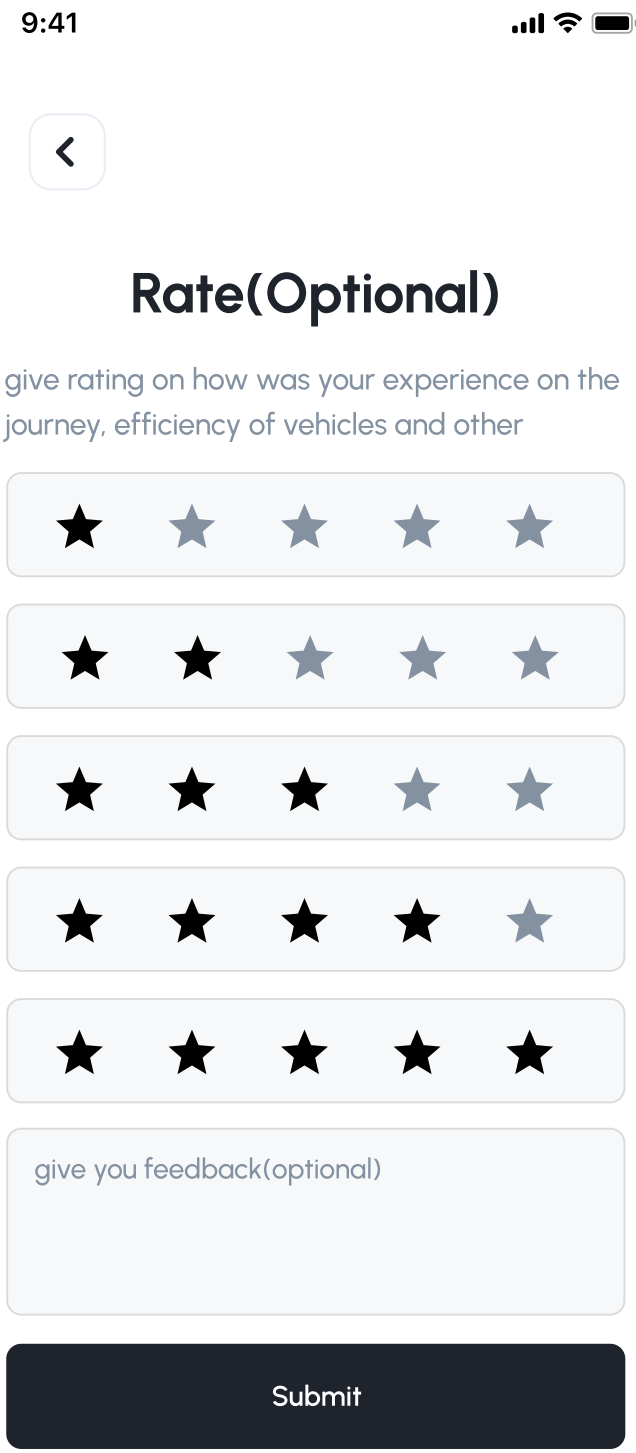


Figure 39: optional rating asking frame

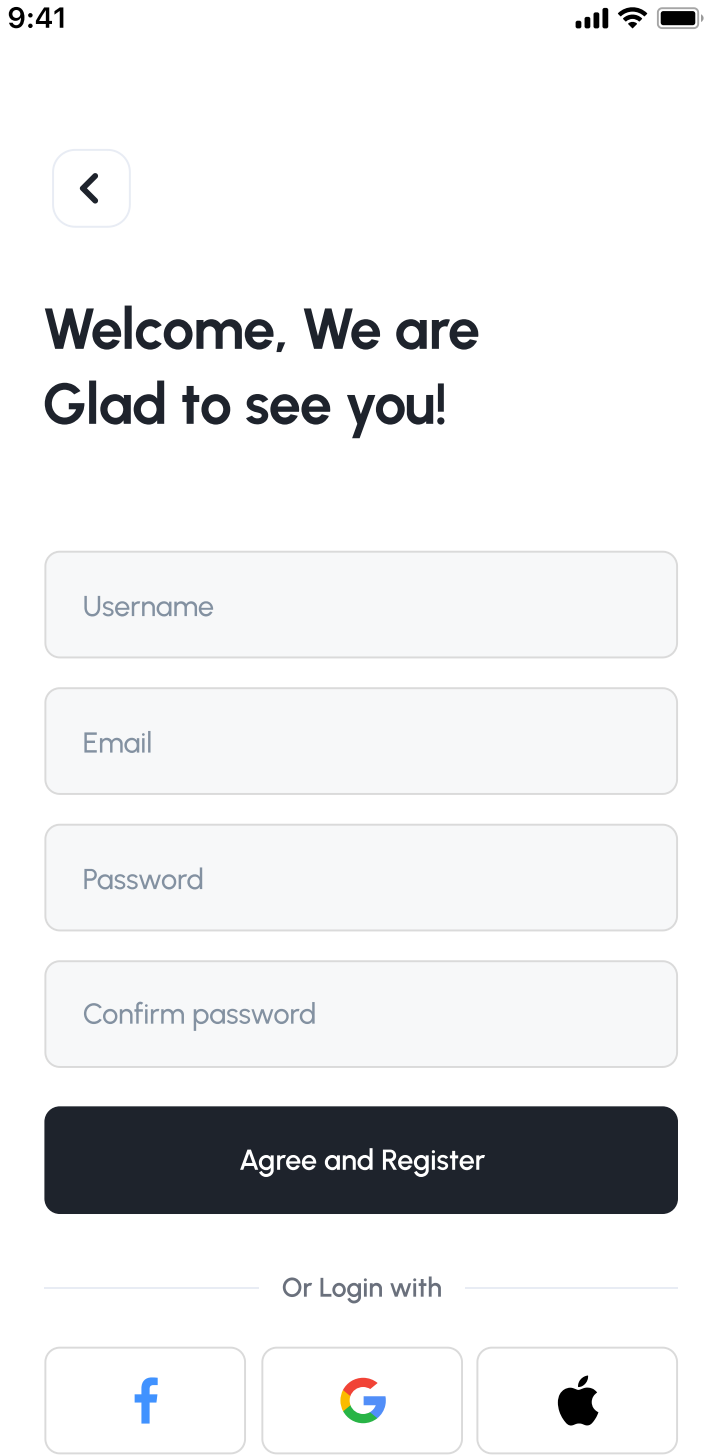


Figure 40: register frame

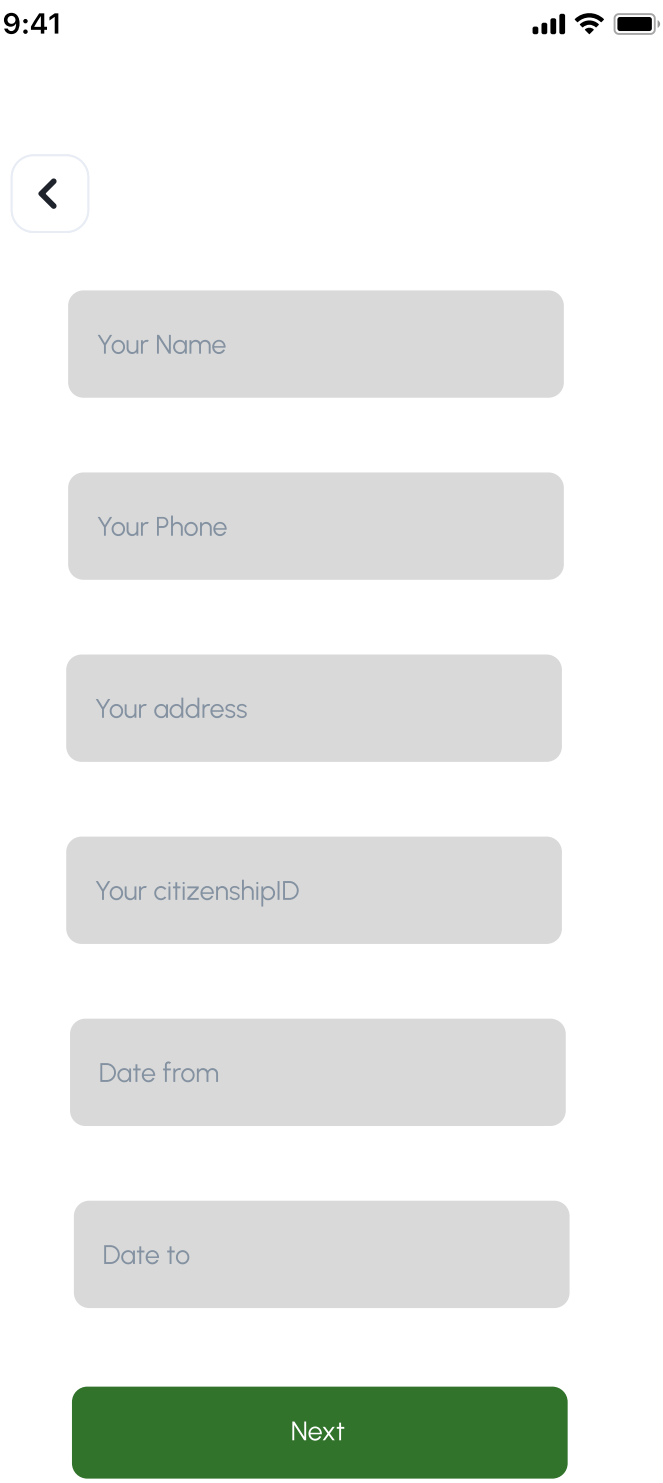


Figure 41: customer detail asking frame for hiring vehicle

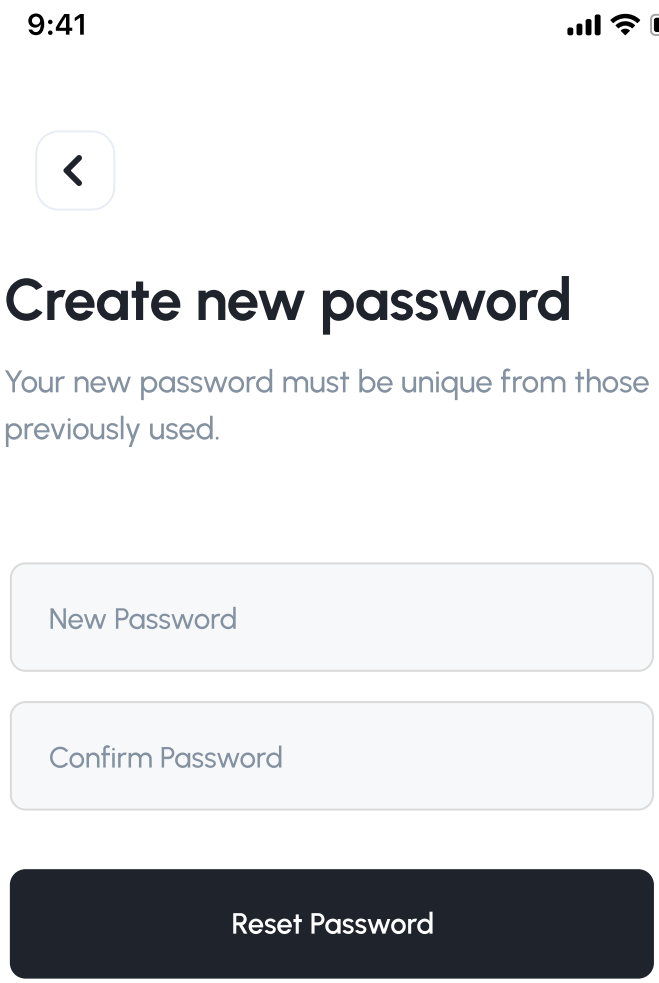


Figure 42: create new password frame

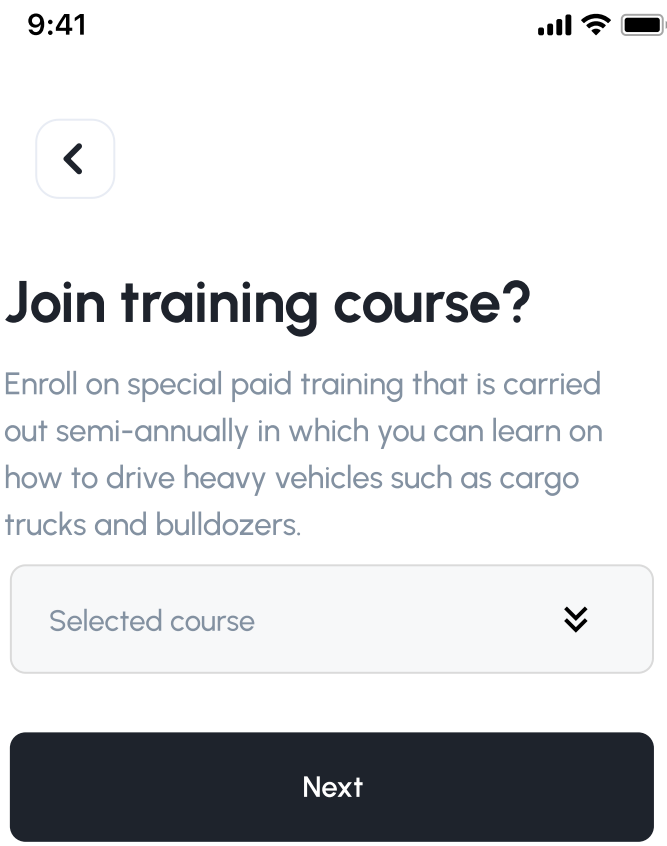


Figure 43: selected training course frame

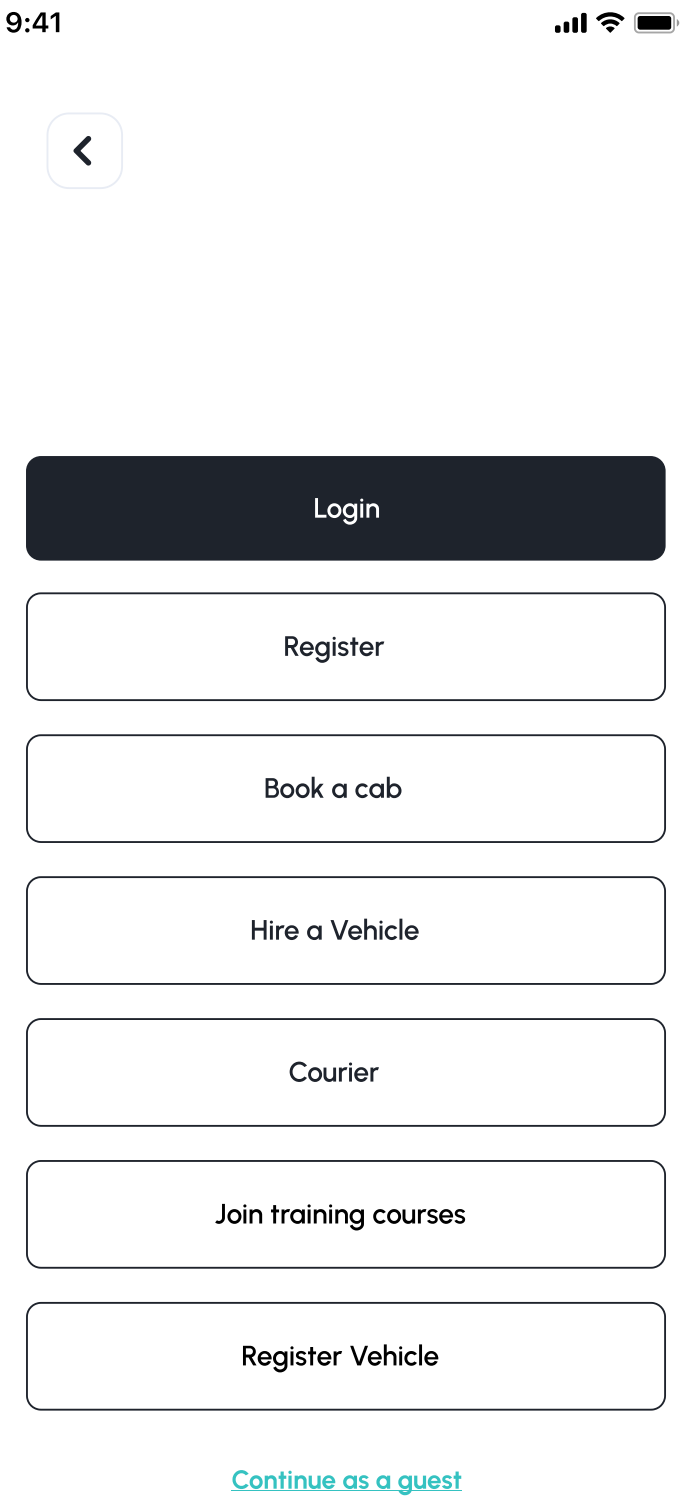
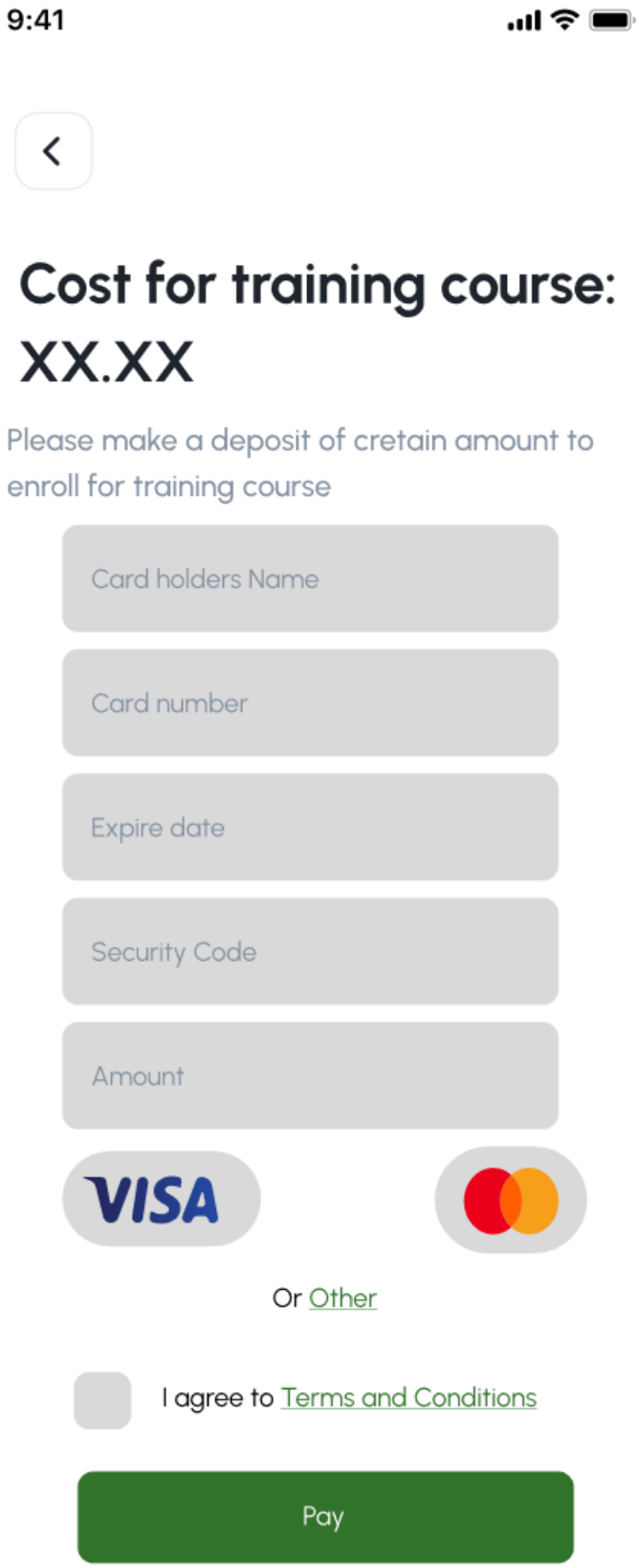


Figure 44: sidebar of software


9:41
Cost for training course:
XX.XX
Please make a deposit of cretain amount to enroll for training course
Card holders Name
Card number
Expire date
Security Code
Amount
VISA
Or Other
I agree to Terms and Conditions
Pay

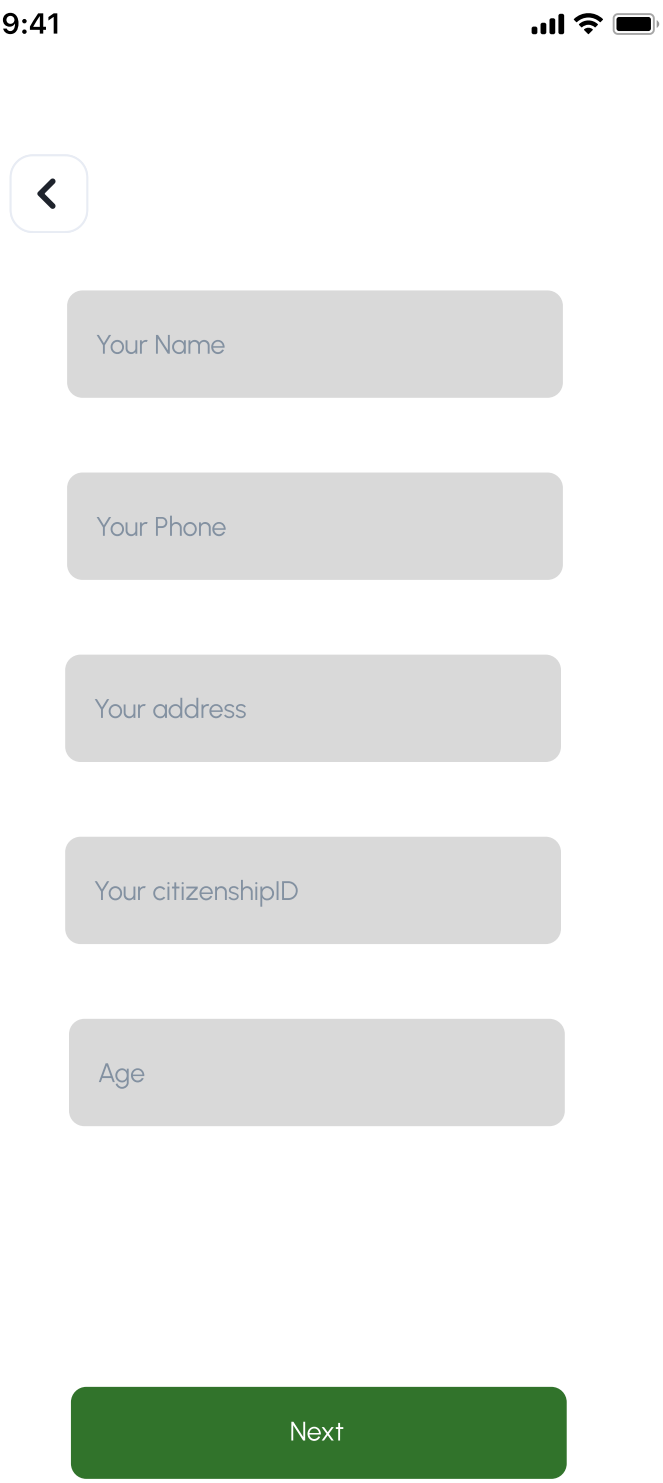


Figure 45: customer detail asker for joining training course

11. Conclusion

As per the specification, we were hired to develop software for Allgemein, in which they offer taxi services where the customer can book taxi services to go from one place to another. The taxi service is not limited to the Kathmandu Valley but also applies to inter-state trips. Second, they want to offer rental car services for vehicles like trucks and bulldozers. After depositing a certain amount of money and identification documents, customers can rent the vehicles for a certain period, and finally, they want a system that keeps track of all transaction records, customer information and all vehicle information so that the necessary information can be extracted at any time required.

During the process of development of the software, we decided to implement the waterfall methodology to develop the software. I had previously designed a use-case diagram for how the system consists of different components and sequences on the flow. Then, the high-level use-case description for each use case was written and the expanded use-case description for two of the use cases was written. Then, I produced step by step process with some descriptions and pictures present to create the collaboration diagram and sequence diagram. Finally, I created a class diagram for the system. The structure of the class diagram was based on domain classes and the methods contained in them turned out to be simple but at first, I had thought that it was complicated and the design was difficult. The waterfall methodology was relatively new to me, but with research, it was a straightforward methodology to work on to make development easy. The research and consulting with others in the same field, friends and the module teacher were able to clear out some confusion that had arisen during the process of development.

While doing this coursework, I learned the importance of a software development methodology to develop better software. Moreover, the knowledge acquired through this project boosted my knowledge of software development, developed problem-solving and research skills, and broadened my horizon on the extent of the IT field, and had an

enriching effect on me as I dipped my fresh hands-on software development. Despite these multiple challenges, they were overcome through research, consultations and problem-solving.

12. References

Anand, B., 2023. *Importance of Software Engineering: Key Reasons.* [Online] Available at: https://www.knowledgehut.com/blog/web-development/importance-of-software-engineering
[Accessed 24 03 2023].

IBM, 2021. *Class diagrams.* [Online] Available at: https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams
[Accessed 27 03 2023].

IBM, 2021. *Communication diagrams.* [Online] Available at: https://www.ibm.com/docs/en/rsm/7.5.0?topic=uml-communication-diagrams
[Accessed 25 03 2023].

IBM, 2021. *Sequence diagrams.* [Online] Available at: https://www.ibm.com/docs/en/rsm/7.5.0?topic=uml-sequence-diagrams
[Accessed 27 03 2023].

IBM, 2021. *Use-case diagrams.* [Online] Available at: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case
[Accessed 25 03 2023].

Martin, M., 2023. *What is Software Engineering? Definition, Basics, Characteristics.* [Online]
Available at: https://www.guru99.com/what-is-software-engineering.html
[Accessed 24 03 2023].

PennState College of Earth and Mineral Sciences, n.d. *Use Cases in a Nutshell.* [Online] Available at: https://www.e-education.psu.edu/geog468/l8_p3.html#:~:text=The%20high%20level%20use%20case,development%20in%20the%20Elaboration%20phase.
[Accessed 25 03 2023].

teamgantt, n.d. *What Is a Gantt Chart? How to Use Gantt Charts in Project Management.* [Online]

Available at: https://www.teamgantt.com/what-is-a-gantt-chart [Accessed 25 03 2023].

tutorialspoint, n.d. *Software Engineering Overview.* [Online] Available at: https://www.tutorialspoint.com/software_engineering/software_engineering_overview.htm# [Accessed 19 4 2023].