CHAPTER ONE

INTRODUCTION

Cargo tracking management system is a java UI application project. In this project, we will simulate the operations of cargo companies. Cargo tracking management system is a system that allows users to see and organize their cargo. This system has both a user interface and an administrative interface side. As addresses and dates are very important for our cargo tracking system, this information will also be provided in this system. Because for cargo tracking, information about where the cargo was on what date should be found in the system. The main purpose of our system is to enable users to easily track their cargo and to create a platform where cargo companies can intervene in these cargoes when necessary. In this system, the information of cargoes, customers, couriers and companies will be found.

Users will be able to see the status of their current cargo in the system after logging into the system with their registered information in the database. The system will have the following information about the cargo: the address of the sender, whether the sender is an institution or an individual (information that should be according to this situation), the weight and size of the cargo, the address of the recipient, the dates of the transactions and the information of the courier will also be included. Customers can also perform other operations within the system if they want, such as creating a new cargo or canceling an existing cargo.

Couriers will also be able to access the system by entering their information. In this way, couriers will be able to see which cargo they will deliver, where they will deliver it, and the information of the cargo from the system. Keeping these records makes a significant contribution to the fast, efficient and timely delivery of cargo.

In the Admin section, each manager can list all the cargoes of his company and view their status. You can cancel the cargo you want. In addition to these, the company can also list its employees. It will provide better in-house tracking of which employee with information about which cargo they are carrying. It may dismiss its employees who do not meet certain conditions, or it may hire new employees. In other words, the administrator will be able to access most processes, which will lead to easier and more effective management.

The reason we chose this project is that cargo companies have a considerable place in today's world. Every company needs this system to ensure that product shipments are delivered quickly, efficiently, and also on time. We chose this system to achieve this situation. It will be ensured that the above-mentioned operations can be done easily with an interface. The system will have a user-friendly interface design that can be understood by everyone. The information in the system will be kept in a database. All information will be stored in the database. The data will be easily updated. Cargo tracking is a very important issue in terms of the image of customers and cargo companies. Smoothly executed cargo management and delivery processes increase the satisfaction of customers who purchase products. This situation is also of great benefit to the cargo company. As a result, we will have turned this important issue into a practical application through our cargo tracking system, which is our project.

CHAPTER TWO

REQUIREMENTS

Admin - Admin will have name, surname, phone number, address, email address, company name. Additional managers will reach their customers and employees.

- **Login-** A method that allows the admin to enter the system in order operate on the system.
  Parameters: String(email,password) **Return:**No return.

- **Change Password-** A method that allows the administrator to change her/his password at login to the system at any time.
  Parameters: String(oldPassword, newPassword, confirmPassword)
  Return: No return.

- **Add employee** – The process of adding new employees by the administrator. The employee's information will be requested. Database information will be added and added information will be returned.
  Parameters: String(name,surname,phone_number,email)
  Return: Courier

- **Delete Employee** – The admin will be able to fire employees as long as they do not meet certain conditions. For the dismissal process, information such as id, name, surname will be requested from the admin.
  Parameters: String(name,surname) int(employee_id)
  Return: Courier

- **List Employees** – A method where the manager can list all employees belonging to his company.

  Parameters: No parameter. **Return:** No return.

- **Cargo search** – A cargo will be found with a specific id and its information will be returned. Contrary to the above-mentioned function, a cargo's information will be provided.

  Parameters: String(receipient_name, recipient_surname) int(cargo_id)

  Return: Cargo

- **List Cargo**– A method where the administrator can list all the cargoes in the company.

  Parameters: No parameter. **Return:** No return.

Customer - Customers' specific id, name, surname, address, phone number information will be kept. Customers have the right to make changes to this information at any time.

- **Login-** A method that allows the customer to enter the system in order operate on the system.

  Parameters: String(email,password)

  Return: No return.

- **Change Password-** A method that allows the customer to change her/his password at login to the system at any time.

  Parameters: String(oldPassword, newPassword, confirmPassword)

  Return: No return.

- **Add cargo** – They can have the new cargo added. After customers create a new shipment, they will be given a tracking number. This function will return the tracking number of the cargo to the user and the customer will track the cargo with this number.
  Parameters: String(receipient_name, recipient_surname,note) Cargo(cargo_info) Address(deliveryAddress)
  Return: int(tracking number).

- **Cargo search** – It is a method that allows the customer to track their cargo with the number given to the customer after the cargo has been added. In this method, information about the cargo location and the cargo carrier will be returned to the customer.
  Parameters: : String(receipient_name, recipient_surname) int(cargo_id)
  Return: Cargo

- **Cargo Cancellation** – It is the function that allows the cancellation of the undesired cargo to be forwarded. This transaction will be provided with the cargo number. The customer will be provided with a warning message that the cancellation has been made.
  Parameter: String(name,surname) int(cargo_id)
  Return: Cargo

Cargo – Each cargo will have an id, cargo tracking number, weight, recipient and sender address.

Courier - Couriers' name, surname, phone number, salary information will be found.

- **Cargo search -** The method that will return the information of the cargo that the couriers will transport. After logging in to the system, the courier will be informed by the cargo search according to the current order. In the order information, there will be the address to which the cargo should be delivered, the name and surname of the customer who will receive the delivery, and the telephone number.
  Parameter: int(cargo_id)
  Return: Cargo

- **Change Password-** A method that allows the courier to change her/his password at login to the system at any time.
  Parameters: String(oldPassword, newPassword, confirmPassword)
  Return: No return.

- **Login-** A method that allows the courier to enter the system in order operate on the system.
  Parameters: String(email,password)
  Return: No return.

Database - In the database section, the attributes user name, password database name, host and port are defined as private. We used database instead of text file for our data. In this class, there are methods that contain changes that we make to the data from the database.

- **Add People**- A method that adds people with some features to the database.
  Parameters: String(name,surname,phone,email,pass) int(type)
  Return:int(a)

- **Add Cargo**- It is a method that allows adding the information of the cargo to the database according to the given parameters.
  Parameters:
  String(cargo_user_id,delivery_address,sent_address,recipient_name,recipient_surname,sender_name,shipping_date,cargo_content)
  Return: int(a)

- **List Cargo-** It is the part that allows us to list our cargoes by pulling them from the database.
  Parameters: : String(ID,type)
  Return: String[][](a)

- **Length Query-** This is the method that returns how many rows of data the query contains according to the parameter sent.
  Parameters: String(type)
  Return: int(rowCount)

- **Length Query Cargo-** It is a method that returns the number of how many cargoes the customer has.

  Parameters: String(ID,type)

  Return: int(rowCount)

- **List Employees-** It is a method that allows us to access a list of all employees in the database.

  Parameters: String(type)

  Return: Queue<People> (info)

- **Cancel Cargo-** It is a method that returns the number of how many cargoes the customer has.

  Parameters: String(cargo_id,senderName)

  Return: int(a)

- **Change Password-** It is a method that allows the user to change the password that they want to change in the database with a new one.

  Parameters: String(newPassword,oldPassword,name,surname,id)

  Return: int(a)

- **Get Name and Surname-** It is a method that returns the name, surname, type and id of a user whose email and password are given from the database.

  Parameters : String(email,password)

  Return : int(a)

- **Get Cargo-** It is a method that allows you to fetch the information of the desired cargo from the database.

  Parameters: String(name,surname,cargo_id)

  Return : String[] (a)

- **Delete-** It is a method that allows you to delete a person's information provided in the attribute section from the database section.

  Parameters: String(name,surname,id,type)

  Return: int(a)

- **Check-** It is a method that allows users using the system to check their information from the database and log in successfully or unsuccessfully**.**

  Parameters: String(email,password)

  Return: Boolean(flag)

- **Check Email-** It is a method that checks whether the e-mail is correct over the database.

  Parameters: String(email)

  Return: Boolean(flag)

- **Change Address Id-** It is a method that allows you to set the address ID.

  Parameters: String(user_ID,addressID)

  Return: int(a)

- **Get Address Id-** According to the information provided from the database, this method allows you to withdraw the address id.

  Parameters: String(street,town,city,apa,door)

  Return: int(address_id)

- **Add Address-** It is a method that allows you to add the desired address to the database.

  Parameters: String(street,town,city,apa,door)

  Return: int(a)

- **Get Address-** The getAddress method takes the user's id, name and surname and makes a query in the database. The return value is kept in a string.

  Parameters: String(ID,name,surName)

  Return: String(address)

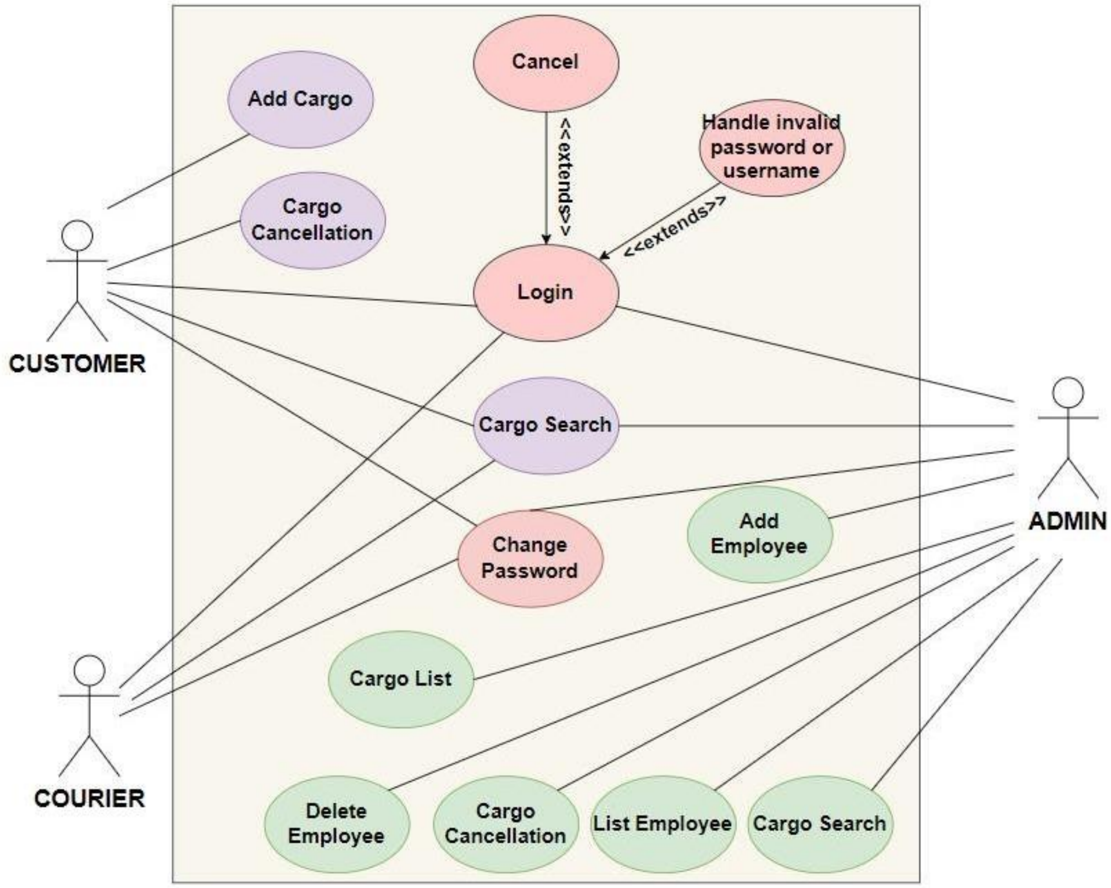
- **Get Address2-** In this method, information is sent to the address method and a new address type object is created.

  Parameters: Address(address)

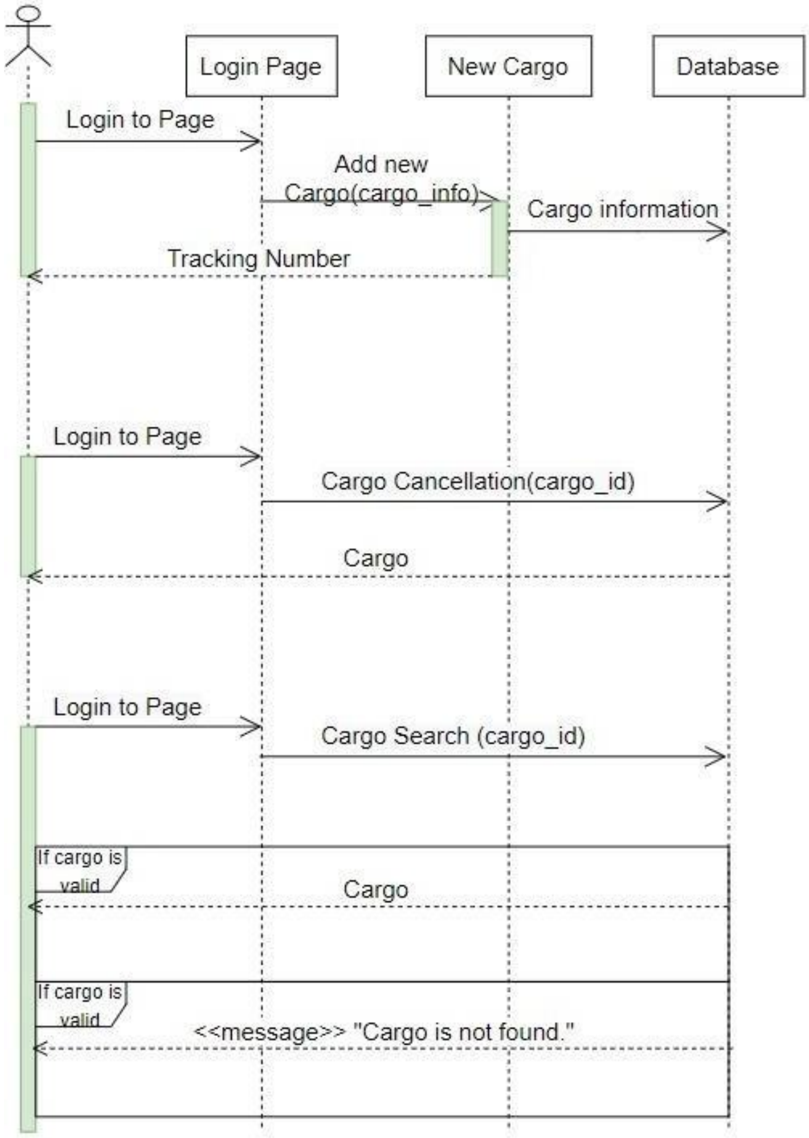
  Return: Address(a)

CHAPTER THREE

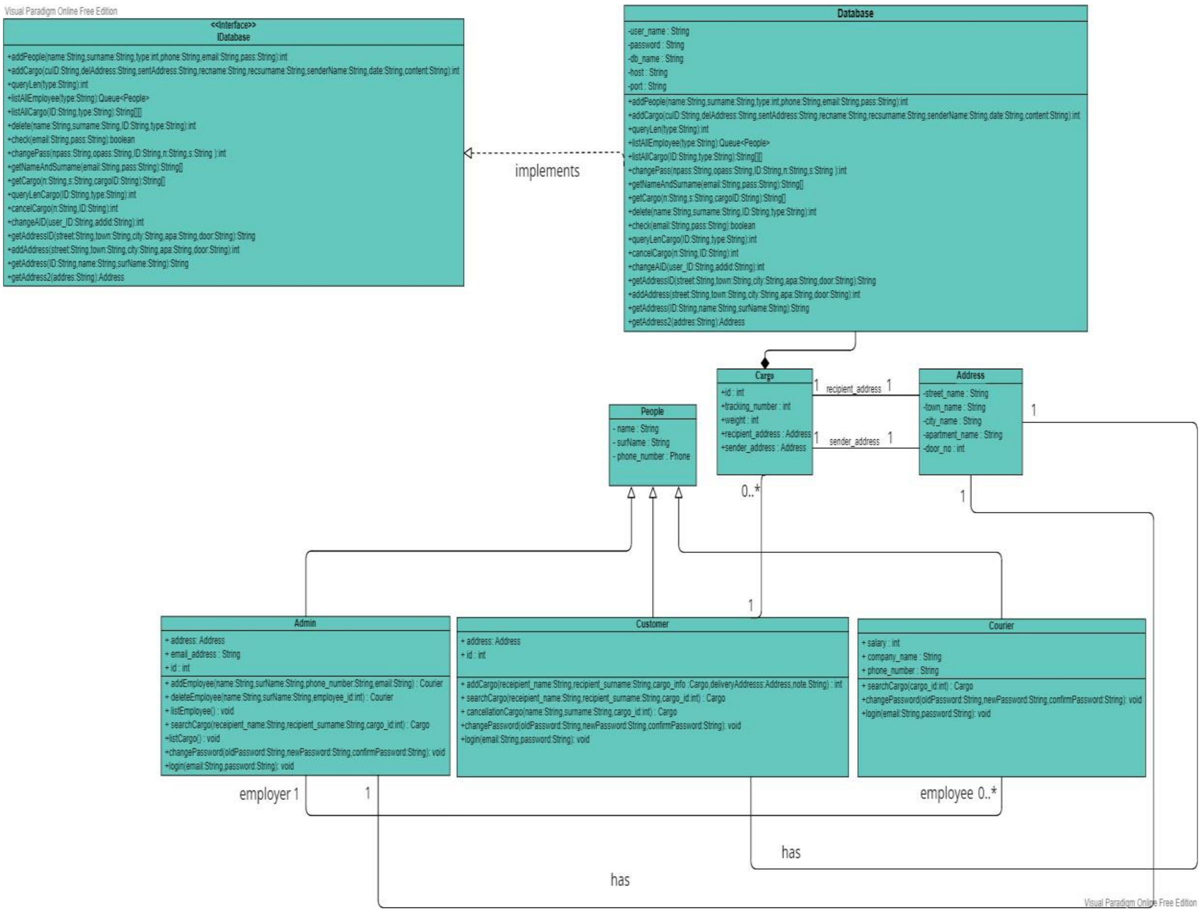
UML DIAGRAMS



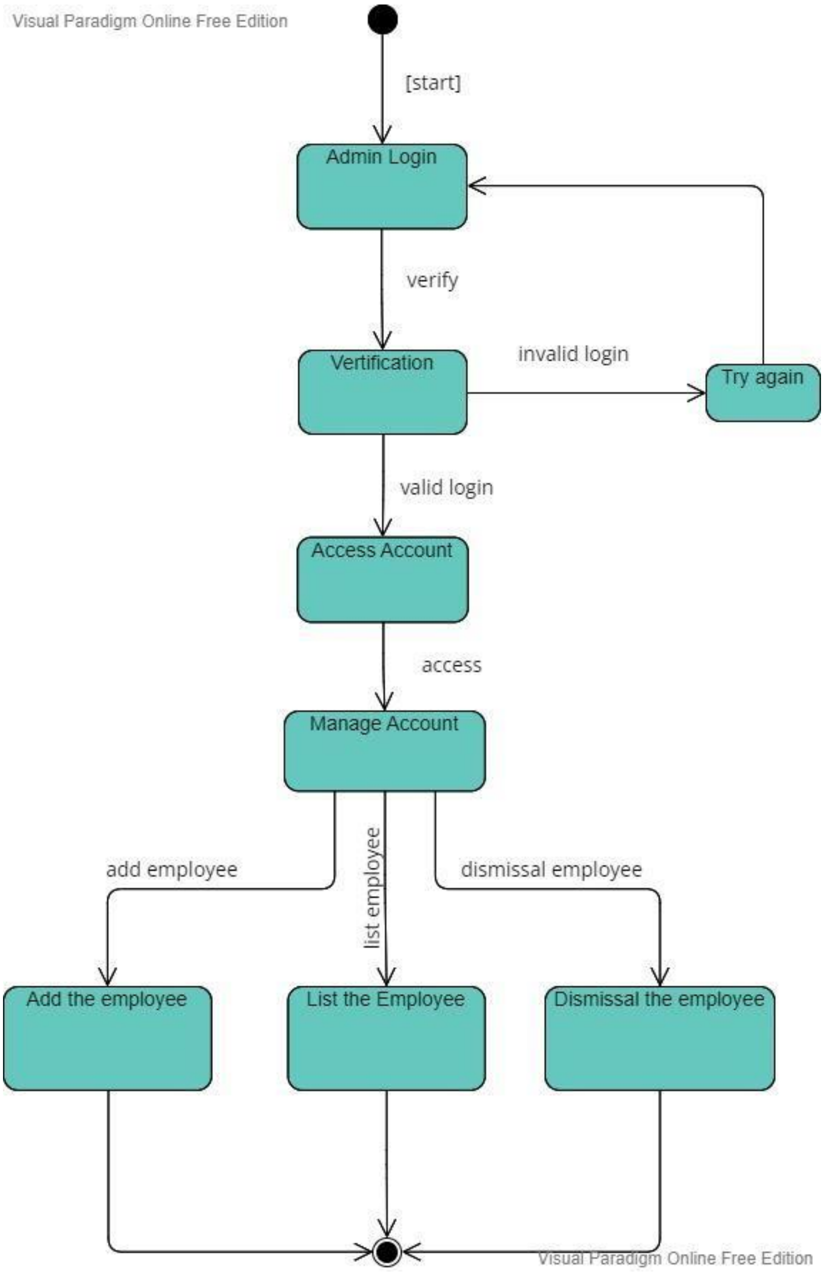
Above is the usecase of the Cargo management system. You can be admin, customer, courier and guest as an actor. Except for guests, other actors can log in from the login page. The login page contains the information section. For example, as shown above, admin can access employee processes and cargo serach process. The transactions that each actor can perform are limited and they cannot perform every transaction.



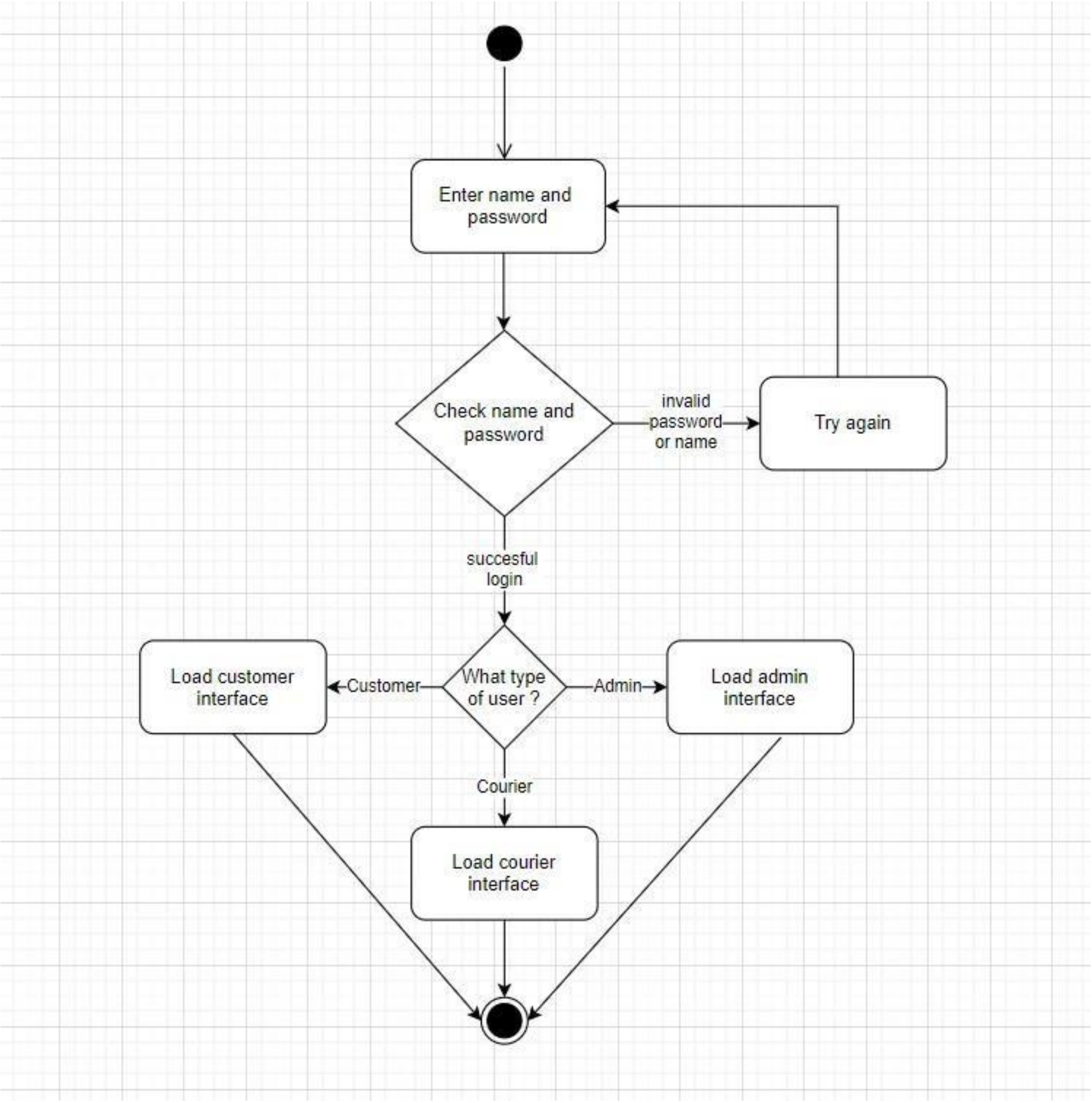
Sequence diagram shows the processes of a customer creating a cargo, canceling the cargo and searching for a cargo in the system, respectively. As seen above, the customer must log in from the login page before performing any transaction. Afterwards, you should decide on the process and continue. When a new cargo is created, the information of the new cargo is added to the database, or when a cargo is cancelled, if it is still registered in the database, the cargo is returned. In the search for the cargo, a warning message or information is returned depending on whether the cargo exists or not.


Visual Paradigm Online Free Edition
<<Interface>>
IDatabase
+addPeople(name:String,surname:String,type:int,phone:String,email:String,pass:String):int
+addCargo(culD:String,delAddress:String,sentAddress:String,recname:String,recsurname:String,senderName:String,date:String,content:String):int
+queryLen(type:String):int
+listAllEmployee(type:String):Queue<People>
+listAllCargo(ID:String,type:String):String[][]
+delete(name:String,surname:String,ID:String,type:String):int
+check(email:String,pass:String):boolean
+changePass(npass:String,opass:String,ID:String,n:String,s:String):int
+getNameAndSurname(email:String,pass:String):String[]
+getCargo(n:String,s:String,cargoID:String):String[]
+queryLenCargo(ID:String,type:String):int
+cancelCargo(n:String,ID:String):int
+changeAID(user_ID:String,addid:String):int
+getAddressID(street:String,town:String,city:String,apa:String,door:String):String
+addAddress(street:String,town:String,city:String,apa:String,door:String):int
+getAddress(ID:String,name:String,surName:String):String
+getAddress2(addres:String):Address
implements
Database
-user_name : String
-password : String
-db_name : String
-host : String
-port : String
+addPeople(name:String,surname:String,type:int,phone:String,email:String,pass:String):int
+addCargo(culD:String,delAddress:String,sentAddress:String,recname:String,recsurname:String,senderName:String,date:String,content:String):int
+queryLen(type:String):int
+listAllEmployee(type:String):Queue<People>
+listAllCargo(ID:String,type:String):String[][]
+changePass(npass:String,opass:String,ID:String,n:String,s:String):int
+getNameAndSurname(email:String,pass:String):String[]
+getCargo(n:String,s:String,cargoID:String):String[]
+delete(name:String,surname:String,ID:String,type:String):int
+check(email:String,pass:String):boolean
+queryLenCargo(ID:String,type:String):int
+cancelCargo(n:String,ID:String):int
+changeAID(user_ID:String,addid:String):int
+getAddressID(street:String,town:String,city:String,apa:String,door:String):String
+addAddress(street:String,town:String,city:String,apa:String,door:String):int
+getAddress(ID:String,name:String,surName:String):String
+getAddress2(addres:String):Address
Cargo
+id : int
+tracking_number : int
+weight : int
+recipient_address : Address
+sender_address : Address
1
recipient_address
1
1
sender_address
1
Address
-street_name : String
-town_name : String
-city_name : String
-apartment_name : String
-door_no : int
1
1
People
- name : String
- surName : String
- phone_number : Phone
0..*
1
Admin
+ address: Address
+ email_address : String
+ id : int
+ addEmployee(name:String,surName:String,phone_number:String,email:String) : Courier
+ deleteEmployee(name:String,surName:String,employee_id:int) : Courier
+ listEmployee() : void
+ searchCargo(receipient_name:String,recipient_surname:String,cargo_id:int) : Cargo
+listCargo() : void
+changePassword(oldPassword:String,newPassword:String,confirmPassword:String): void
+login(email:String,password:String): void
Customer
+ address: Address
+ id : int
+ addCargo(receipient_name:String,recipient_surname:String,cargo_info :Cargo,deliveryAddresss:Address,note:String) : int
+ searchCargo(receipient_name:String,recipient_surname:String,cargo_id:int) : Cargo
+ cancellationCargo(name:String,surname:String,cargo_id:int) : Cargo
+changePassword(oldPassword:String,newPassword:String,confirmPassword:String): void
+login(email:String,password:String): void
Courier
+ salary : int
+ company_name : String
+ phone_number : String
+ searchCargo(cargo_id:int) : Cargo
+changePassword(oldPassword:String,newPassword:String,confirmPassword:String): void
+login(email:String,password:String): void
employer 1
1
employee 0..*
has
has
Visual Paradigm Online Free Edition

We have defined classes and their properties and the behaviors that they can express, and we have drawn a class diagram for them.There may be various relationships between classes.For example, there is association and multiplicity between admin and employee. An employer has more than one employee and we defined it with multiplicity. There is also a realization relationship between the Database class and the IDatabase class.This relationship is indicated by a dashed line. The Database class implements the other class.There are also composition relationship.



The structure that shows all the states of an object or system as a diagram is called a state diagram. We also drew a state diagram for the admin. Admin tries to log in to his account successfully by entering his information into the system. If the verification of information by the system occurs, it continues, if not, it should try again. If it succeeds, the system will now also have a successful account. Thus, by managing the account, you can perform some operations and, as a result, reach some situations. For example, we can access the list of employees automatically by using the "listemployee" process.

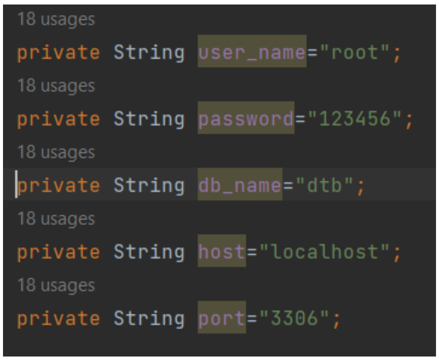


Above is an activity diagram of the login process. After a name and password are entered, it is questioned whether this name and password are correct. If false, this process is repeated. If it is correct, the type of the logged in user is determined and an interface is loaded on the screen accordingly.

CHAPTER FOUR

IMPLEMENTATION

We chose singletation from the design patterns requested from us in our project and used it in the database class. For this, we created an instance called getInstance. The purpose of using it is to create an object for the whole application. Thus, we have provided controlled access in our project. Also, our database class depends on the database interface. This is because there is clarity in determining which methods will be inside the database class. The contents of the methods in the interface are filled according to the required operations. First of all, certain attributes have been created so that the database can be used (*Attributes*). These values were used to create the url for each method. A connection to the database has been established with the created url. With this established connection, it provided access to both data. These data, together with the methods, were used for both insertion, deletion and control.



```
18 usages
private String user_name="root";
18 usages
private String password="123456";
18 usages
private String db_name="dtb";
18 usages
private String host="localhost";
18 usages
private String port="3306";
```

Attributes

The database class is the class in which the transactions are made. This class includes addPeople(), addCargo(), queryLen(), queryLenCargo(), listAllEmployee(), listAllCargo(), cancelCargo(), changePass(), getNameAndSurname(), getCargo(), delete(), check() and checkEmail() methods. How the methods work is described below.

First, in the check method, it is checked according to the email and password received from the user via the interface. If a match is achieved, the user is redirected to the other page according to the type. The user's type can be admin, customer or courier. Select is used in the query part of the method. If the correct input is made, the boolean held flag becomes true.

The AddPeople method is also used during the admin's employee addition. For this, the e-mail address is controlled. The method that provides this is the checkmail method. The entered variables are checked in the database, and if they are not pre-attached, a warning message is returned. In addition, if there are deficiencies in some of the information requested to be added, it is not added and a warning message is given.

The addcargo method is like the addpeople method, but it is used for adding cargo. It has similar controls in it. It is provided to add according to the absence of missing information.

The queryLen method is used to provide information about how many people are of the requested type. For example, it allows to reach the number of people with type 3 (courier). The queryLenCargo method asks for the user's id and returns the number of registered cargo belonging to that id.

The listAllEmployee method allows the database employees to be listed by giving their type. Retrieves Id names, surnames and phone numbers. It allows to be listed with this information. Here queue structure is used to hold the employes and their information. In this queue created in People type, the name, surname, id and phone information of the employees are kept.

The listAllCargo method ensures that all cargoes are pulled from the database. Id and type information are sent to this method. Depending on whether it is an admin or a customer, it is possible to access the cargo information. 1 type belongs to admins and 3 types belong to customers. Here, a 2-dimensional array is used to hold information about the cargoes.

It is ensured that a user named in cancelCargo method is deleted from the database with the cargo id information.

The changePass method allows to update the name, surname, id, old password and new password information and the password of the user registered in the database.

The getNameAndSurname method provides access to the user's information according to the entered email and password. The user's type , id , name and surname are kept in a string type array so that they can be used.

The getCargo method provides access to the cargo information of the user who entered the name surname and cargo id. Like the getNameAndSurname method, information is kept in an array of string type. This array is returned.

The delete method works when the admin employee needs to delete it. After entering the name, surname, id and type in the data, the relevant data is deleted from the database.

The checkEmail method is used for checking like the check method. If the sent email address is already registered in the database, the boolean flag will be false and it will be prevented from being added.

The changeAID method gets the user's id and the address id. Allows you to change the id of the address.

The getAddressID method takes the content of the address and ensures that the address of this information is found in the database and its id is returned.

The addAddress method gets the content of the address. It allows the address with this data to be added to the address database.

The getAddress method takes the user's id, name and surname and makes a query in the database. Returns the id of the person it finds and stores it in a string. In the getAddress2 method, it is sent to the address method. A new address object is created for this address. This newly created address object is returned.

In the actionPerformed method in the Launchpage class, we first created an object of the database type. Then we had to check the login button and the register button. We returned a boolean value by sending the email and password that we received from the user to the database's check method to check the login button, if the value is true, we showed the user the message "Login Successful", if the value is false, we also showed the message "Login Unsuccessful". If there was a successful entry, we sent the mail and password to the getNameAndSurname method of the database and threw the returned value into the string list. The contents of this string list include the user's first name, last name, user type, and ID. If the type returns a value of zero or a value, we have enabled the admin type to create an object.Another situation is that if it returns two, as a type, then we have created an object of the customer type.Finally, if it returns the value of three, we create a courier-type object. If the user presses the register button, we have opened the registerWindow page.

In the actionPerformed method in the registerWindow class, we first created an object of the database type. Then we provide the register button and the control of the back button.If the user has pressed the back button, he is redirected to the launch page again. In the content of the register button, we check some situations. For example, it prevents the user from registering when the user does not enter a value in any of the two password fields.Or when you leave the name,surname,email sections blank, it does not allow registration in the same way.Finally, if the user enters the digits of the phone number more or less than it should be, the user is also prevented from registering in the system in the same way as the user. In addition, the address and address id of the person who will register in the database are added to the register window class.

In addition, if the user's email was previously in the database, the error message is also displayed. As a final check, the two passwords entered by the user must match each other if they are not the same, the user is told "Register Unsuccessful.\nPasswords are not the same" message is shown. After all these cases have been provided, the user can now register. Registration occurs successfully by sending parameters to the addPeople method of the database class. After successful registration, the user is redirected to the login page so that she/he can log in to the system.

There are also similar things in the addEmployee class to those in the register process. For example, in this section, if any of the employee's information is left blank, it prevents us from adding a successful employee. If the e-mail of the employee to be added is an e-mail previously saved in the database, "This e-mail already exists." displays the message.

In the Cargo Cancel class, we first created an object of the database type. Then, the data received from the user is sent to the cancelCargo method of the database class. According to what this method returns to us, the user is shown information about whether it was successful or unsuccessful. Of course, all this happens if the user has not left the cargo id part blank.

The actionPerformed method in the CargoSearch class also performs a cargo search. Firstly, the necessary information is sent to the getCargo method of the database class. Then the returned value is stored in a string list.If the first value of this string list is not empty, this means that the cargo has been successfully added to the database. Finally, data is added to the table to display the returned information in the interface.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

Our project was to create a cargo tracking system by creating an interface so that users could use it conveniently. We used swing to create the interface within the scope of our project.In this way, we have learned how to use swing in detail. For example, we learned how to use buttons using swing, or how to create tables, and many other topics.We can also say that we have learned how to connect the interface pages to each other. In other words, we have developed our project considerably in terms of design.

In the background of our project, we used the object oriented programming approach.In other words, the features required by object oriented programming are provided in our project. Since it is a big project, it has allowed us to learn very well how to use many classes in a way that is compatible with each other. In our project, we chose to use a database instead of using a text file to shoot videos. The fact that we chose this caused us to learn quite a lot about database, even if it forced us in some places. How to pull data from the database or how to update the data there, how to add data or how to delete data from the database.Using a database has helped us learn a lot of things like this.

In addition, we learned to work in a planned way because we did our project by systematically creating uml diagrams. We found that working using uml diagrams is very useful for an object oriented project. We found that working using uml diagrams is very useful for an object oriented project.

Our project works only for VIZZ kargo. But in the future, our project may expand to the use of all other cargo companies. We think that our project can reach a larger area by adding some additional studies in this direction. in other words, we are considering developing our project in this direction in the future.