GTU Department of Computer Engineering CSE 222/505 - Spring 2022
GROUP 2
2nd Project Report

Instructor
Prof. Dr. Fatih Erdoğan SEVİLGEN

PROJECT
Online Food Service System: HoldON

Table of Contents

Note: Updated parts from proposal are highligted with blue if an adding is performed, ~~scratch off~~ is performed if it is removed from project and explained why.

1. GROUP MEMBERS

STUDENT	ID
AZİZ CAN AKKAYA	1801042250
YASİN AKAR	215008003085
FEYZA NUR KÜÇÜK	1801042618
SERHAT SARI	200104004028
ABDULSAMED ASLAN	200104004098
MUHAMMED SİNAN PEHLİVANOĞLU	1901042664
ATACAN BAŞARAN	200104004008
ASUMAN SARE ERGÜT	1901042657

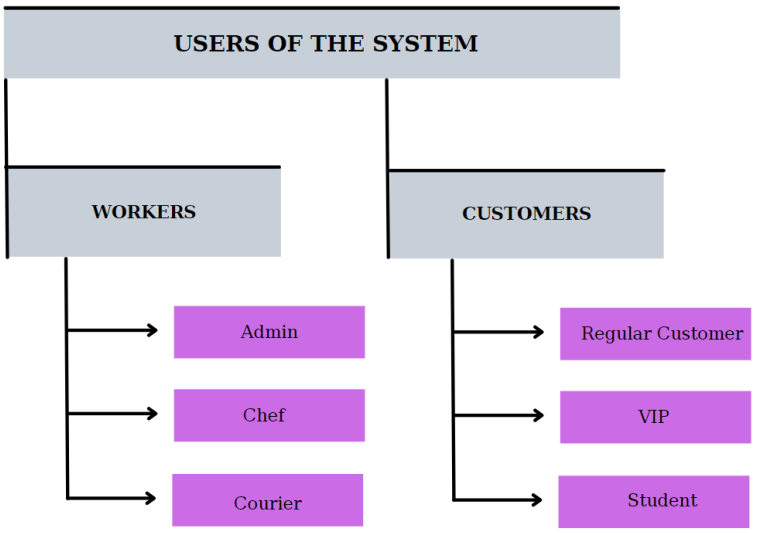
2. PROBLEM DEFINITION

Time is a determinant property nowadays that everyone considers while making decisions. In a short lunch break, since going to your favorite restaurant takes time, you may be obliged to prefer nearby restaurants. Or, in the group you will go to lunch with, there may be people who have different tastes. Giving different menus in groups and eating together still is possible in giving remote orders.

This time, let's have a look at from restaurant's perspective. Accommodate in a big place to serve more people is necessary. And bigger places costs higher bill and rent. This leads decreases in the profit of restaurant. To hold service quality maximum while given effort and time per customer is stable, is possible with online order system. Moreover, keeping customer data increases customer satisfaction due to it provides personalized menus.

Considering all those, an online food ordering system has been developed, named HoldON, referring “You don’t need to go anywhere to eat meal, just use your phone”.

3. USERS OF THE SYSTEM



3.a) Workers

3.a.1) Admin

Admin role is for maintaining and managing the whole restaurant system, it is designed to be owner of a restaurant. Hence, has capability of see and edit all information about restaurant itself, it's users and other features like orders, income/outcome.

He/she can see both type of user's - workers and customers- personal information, which are name, age, username and password and non-personal information of workers, such as certificate number. Salary of the worker is being determined by admin, according to customer's votes. In case of evaluated worker performance is decreased over some point, admin can fire that worker and can hiring someone new.

Managing the menu of restaurant -create, change, delete whole menu or some foods inside menu - is also under the responsibility of admin.

3.a.2) Chef

Chef has same user information with additional professional qualification indicators like certificate number and experience year. It's responsibility is cooking the order in the waiting order queue (~~create food with ingredients by customer orders and chefs can change the menu according to requests or ingredients.~~[1]). The status (beginner, junior, mid-level, senior) and salary of the Chef are updated according to professional competence, experience and customer votes.

3.a.3) Courier

Couriers deliver food orders that are prepared by Chef to customers and couriers can see the list of orders cooked to be delivered to customers. As a requirement of direct communication with the customer, unlike the Chef, Courirer has phone number information. The status (beginner, junior, mid-level, senior) and salary of the Courier are updated according to experience year and average score, rated by customers.

Delivering process will be prioritezed in the queue according to order's price, the most expensive order will be delivered first.

3.b) Customers

For registration and verification process, a typical user is asked for name, age, job, unique username, password, phone number, balance information. Customer can see it's previous orders and can give new order from menu anytime. After order is taken, Customer gives votes to both Courirer and Chef.

3.b.1) Regular Customer

Regular customer has no privilage like students or VIP customers. They are just typical users. Account is calculated without any reducer coefficent.

3.b.2) VIP

When the given order amount reaches the 5, regular customer turns into VIP customer and gains some privilages. Those are: Getting %15 discount for following orders and if order's price is high, gaining priority in the order queue.

3.b.3) Student

If the customer's job is selected as student during registration, without waiting to be VIP, customer's account will be evaluated with %25 discount.

1- This feature was removed because it was wrong to change the customer's order for any reason and send a different product than the one requested.

4. REQUIREMENTS IN DETAILS

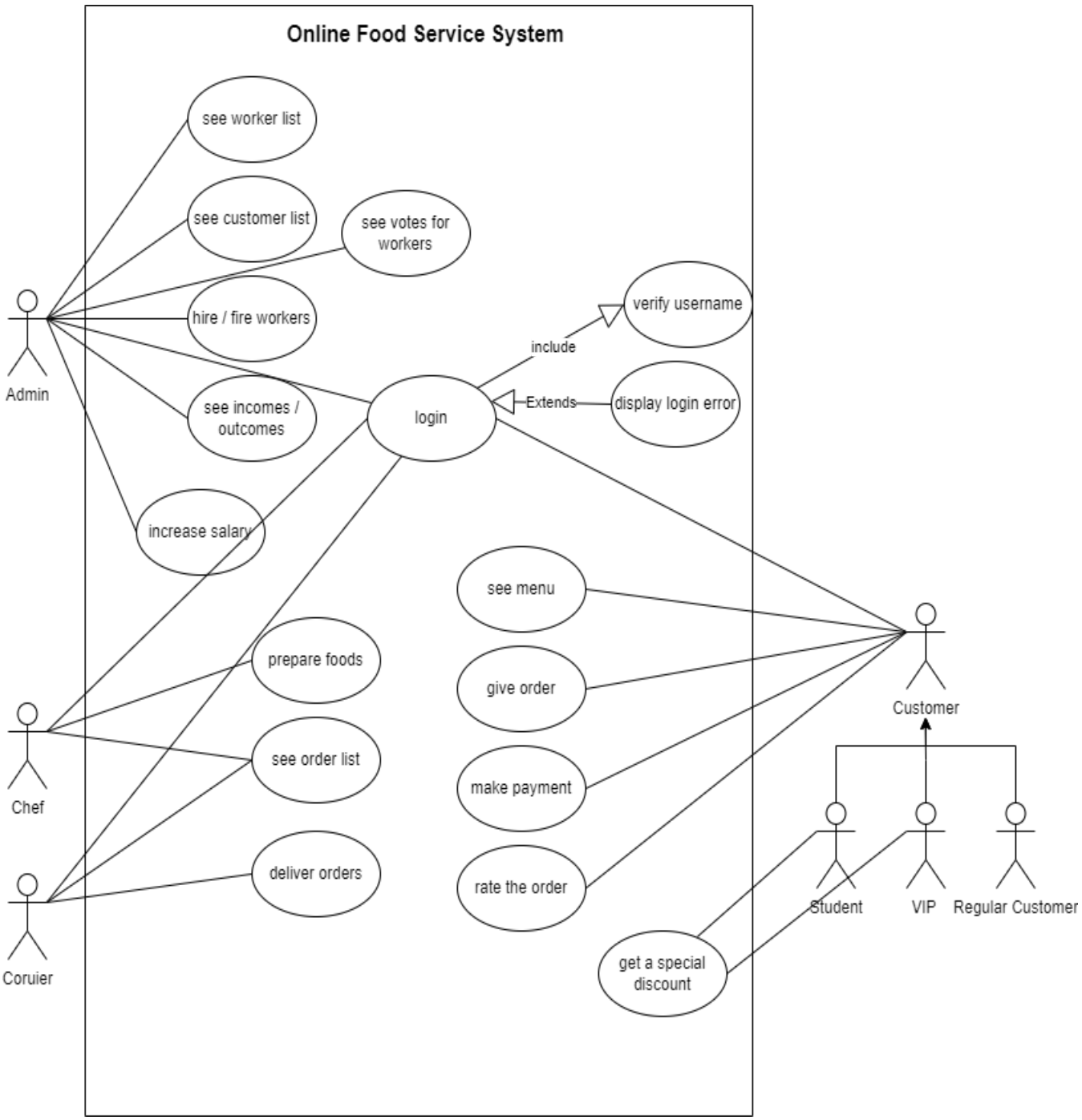
a. Functional Requirements

	ADMIN	COURIER	CHEF	STUDENT	REGULAR CUSTOMER	VIP
Login to the system	Y	Y	Y	Y	Y	Y
See the Menu	Y	N	N	Y	Y	Y
Edit the Menu	Y	N	N	N	N	N
Add/Remove Workers	Y	N	N	N	N	N
See worker information	Y	Y (only itself)	Y (only itself)	Y (not all)	Y (not all)	Y (not all)
Edit worker information	Y	Y (only itself)	Y (only itself)	N	N	N
Giving vote to workers	N	N	N	Y	Y	Y
See worker votes	Y	Y (only itself)	Y (only itself)	N	N	N
See order list	Y	Y	Y	N	N	N
Edit order list	Y	Y	N	N	N	N
Order a meal	N	N	N	Y	Y	Y
Gaining experience year	N	Y	Y	N	N	N
Gaining certificate	N	N	Y	N	N	N
Discount in order	N	N	N	Y	N	Y

b. Non- Functional Requirements

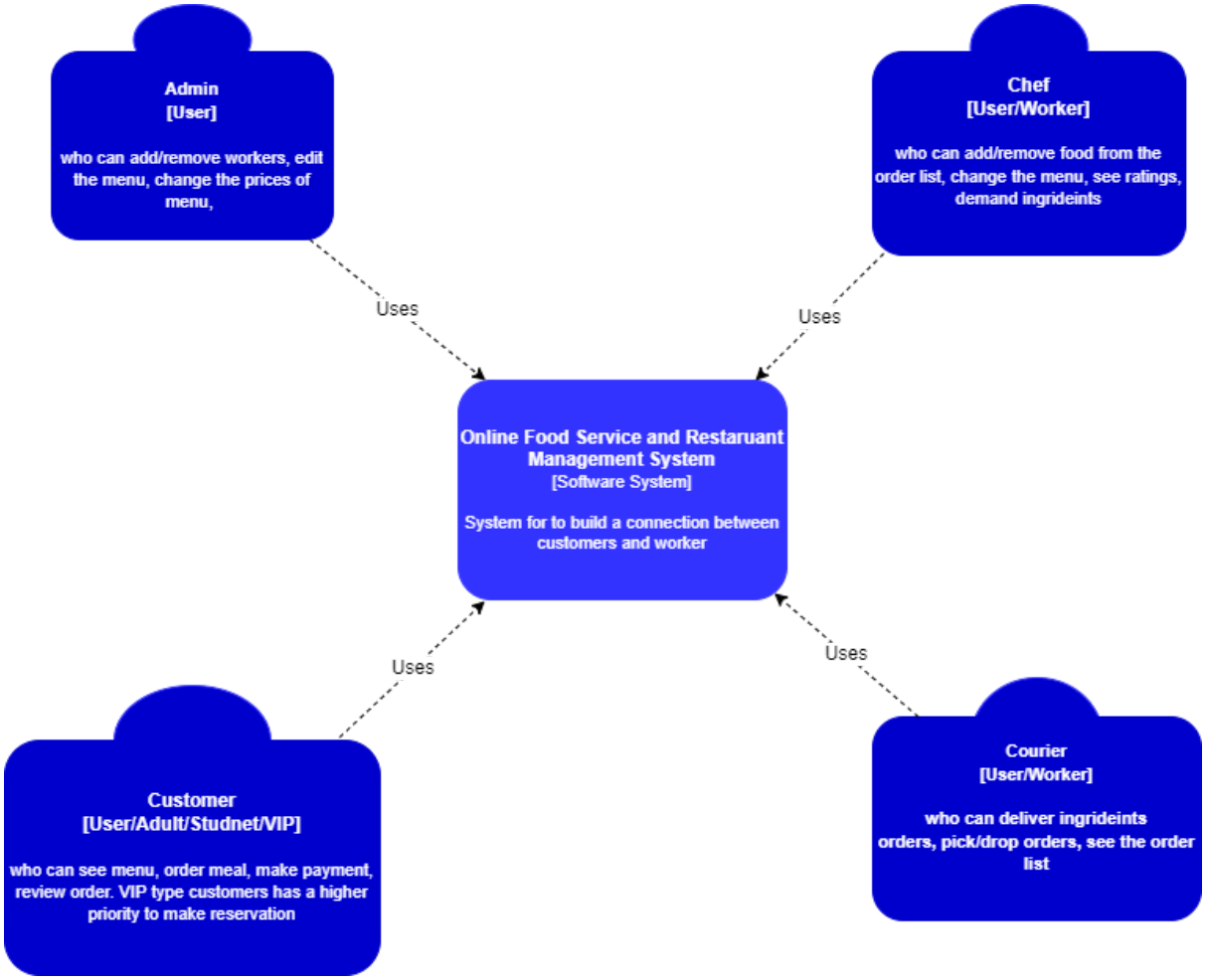
Back-end Software : Java 11
Software should be able to compile with “javac” on a linux distribution.
Hardware Interfaces: Mac, Linux and Windows operating systems.
The system follows a password policy.
The program should be maintainable.
The program should be usable. It should be safe and effective for the users to perform the desired tasks.
The program should be extendible. It should be able to receive new features and customizations for the upcoming versions.
System-Users’ information and restaurant information are kept in two separate files. When program is active, In case any login attempt by users, the system checks the validation of the user with the help of this user-information file. Other file should have restaurant information like menu and ingredients. In case any update on user information and restaurant information, related files is going to be updated simultaneously.

5. USE CASE DIAGRAM

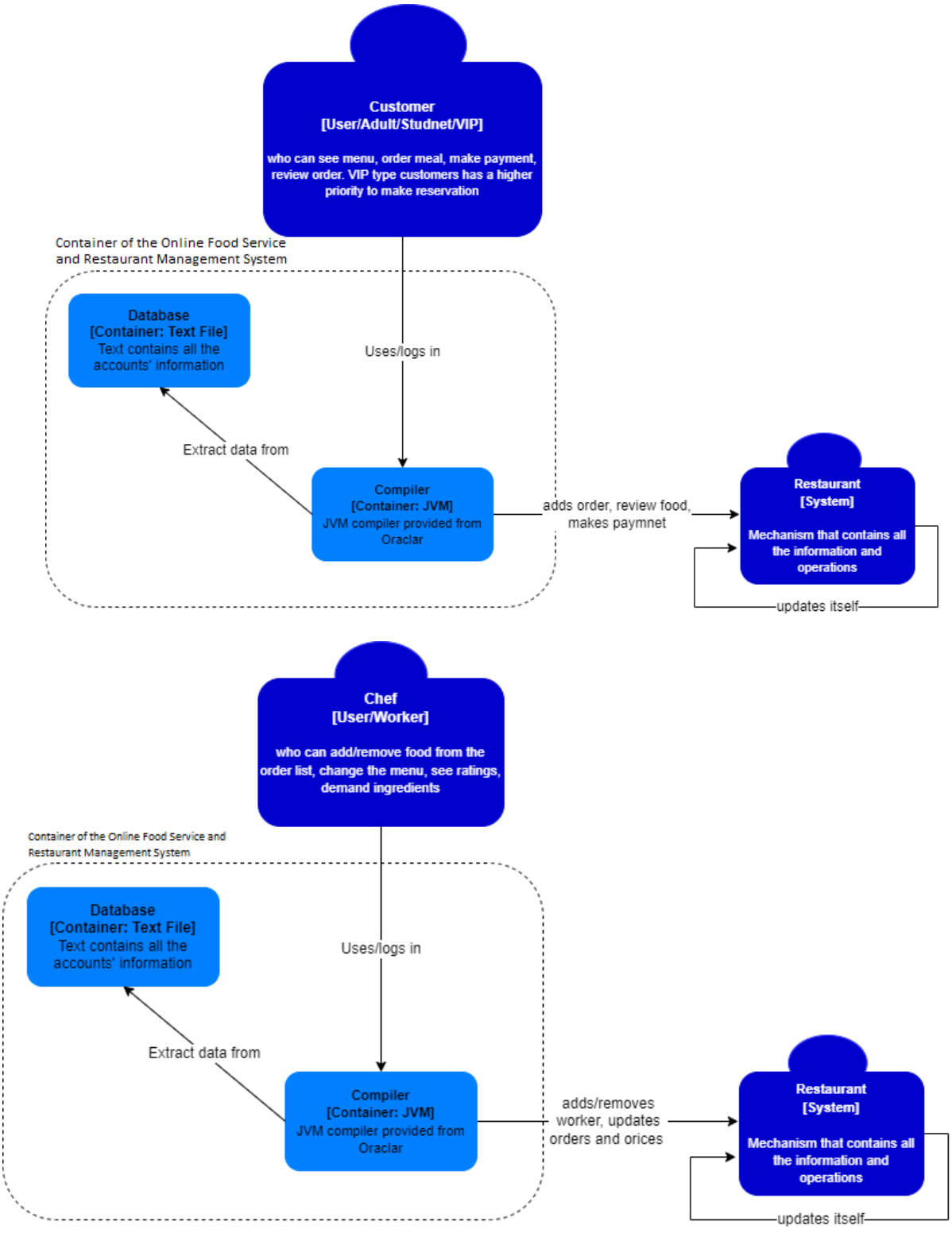


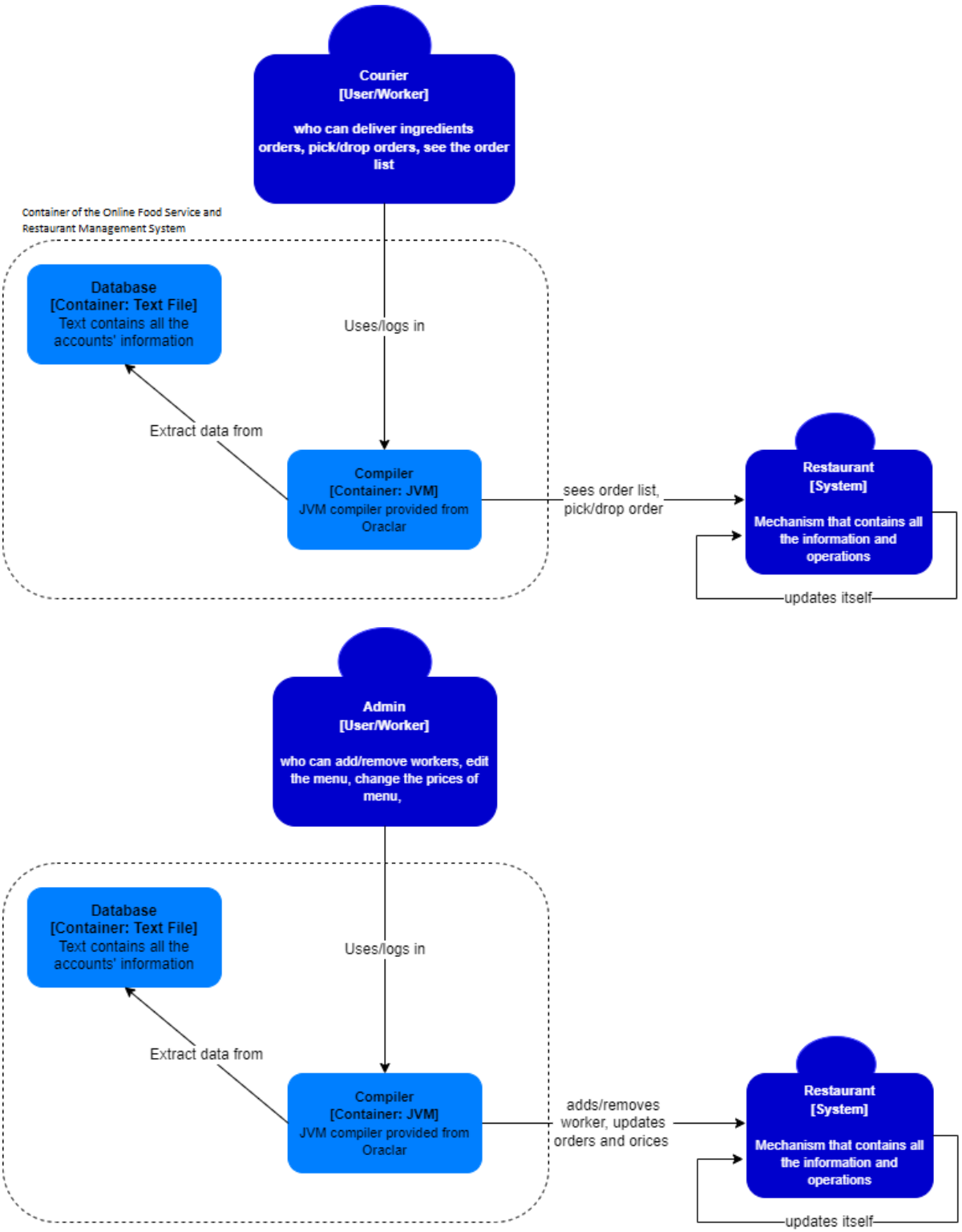
6. C4 MODEL OF THE SYSTEM

Level 1:

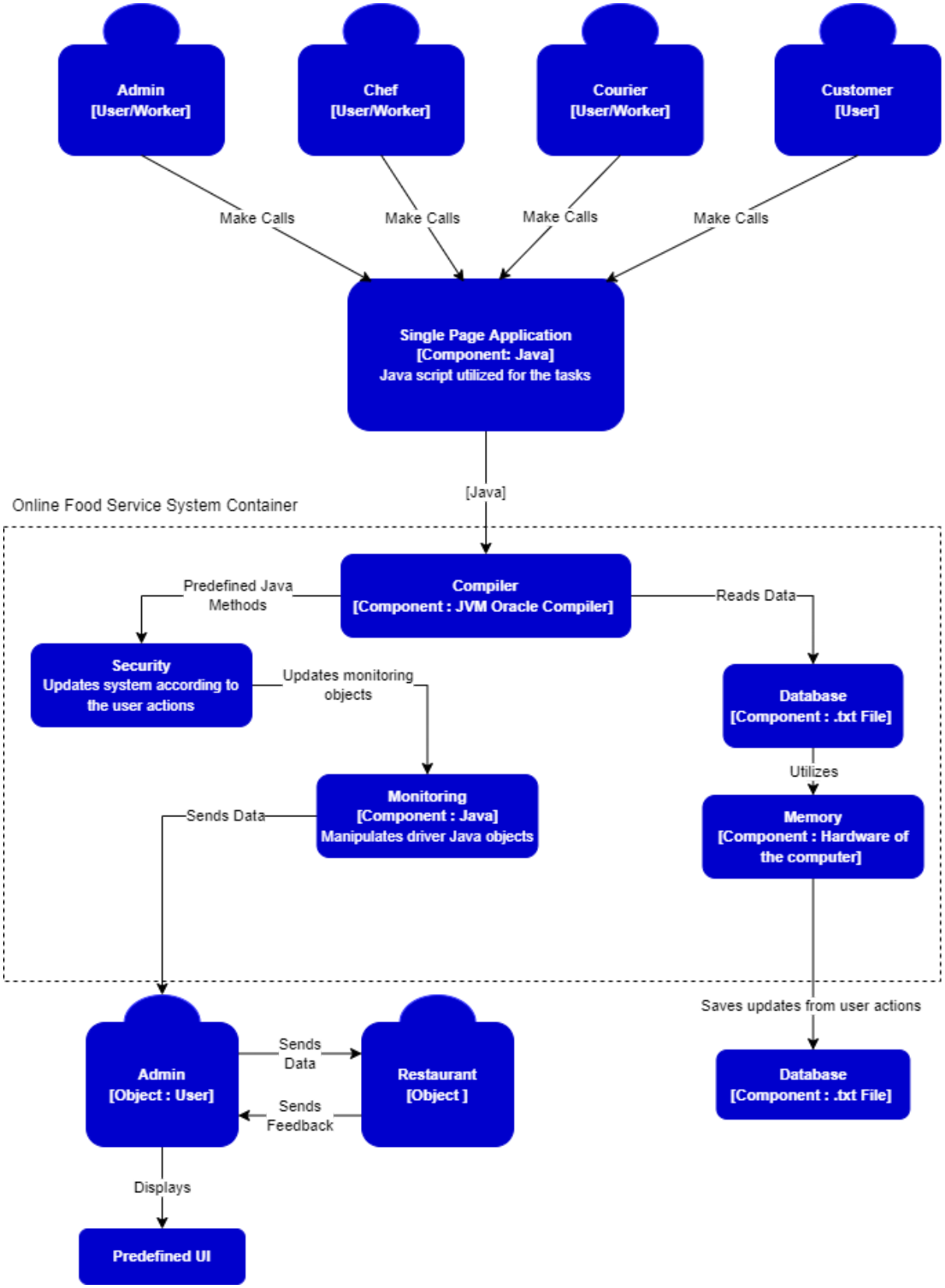


Level 2:

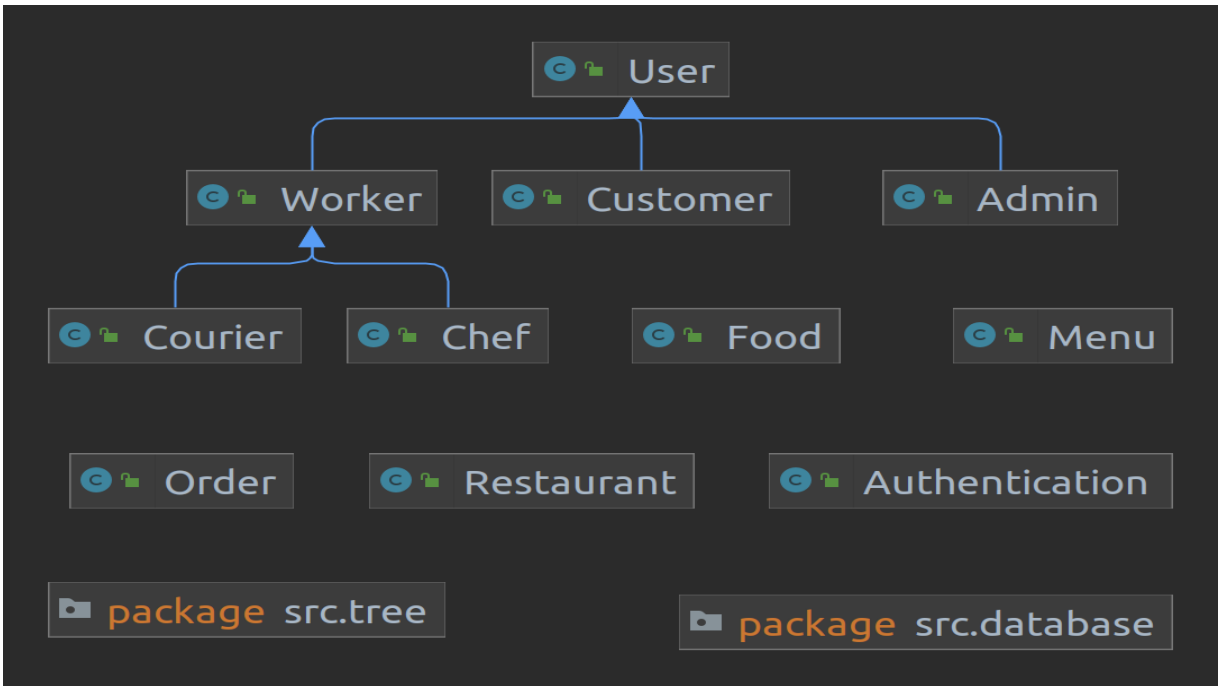

Customer
[User/Adult/Studnet/VIP]
who can see menu, order meal, make payment, review order. VIP type customers has a higher priority to make reservation
Container of the Online Food Service and Restaurant Management System
Database
[Container: Text File]
Text contains all the accounts' information
Uses/logs in
Extract data from
Compiler
[Container: JVM]
JVM compiler provided from Oraclar
adds order, review food, makes paymnet
Restaurant
[System]
Mechanism that contains all the information and operations
updates itself
Chef
[User/Worker]
who can add/remove food from the order list, change the menu, see ratings, demand ingredients
Container of the Online Food Service and Restaurant Management System
Database
[Container: Text File]
Text contains all the accounts' information
Uses/logs in
Extract data from
Compiler
[Container: JVM]
JVM compiler provided from Oraclar
adds/removes worker, updates orders and orices
Restaurant
[System]
Mechanism that contains all the information and operations
updates itself

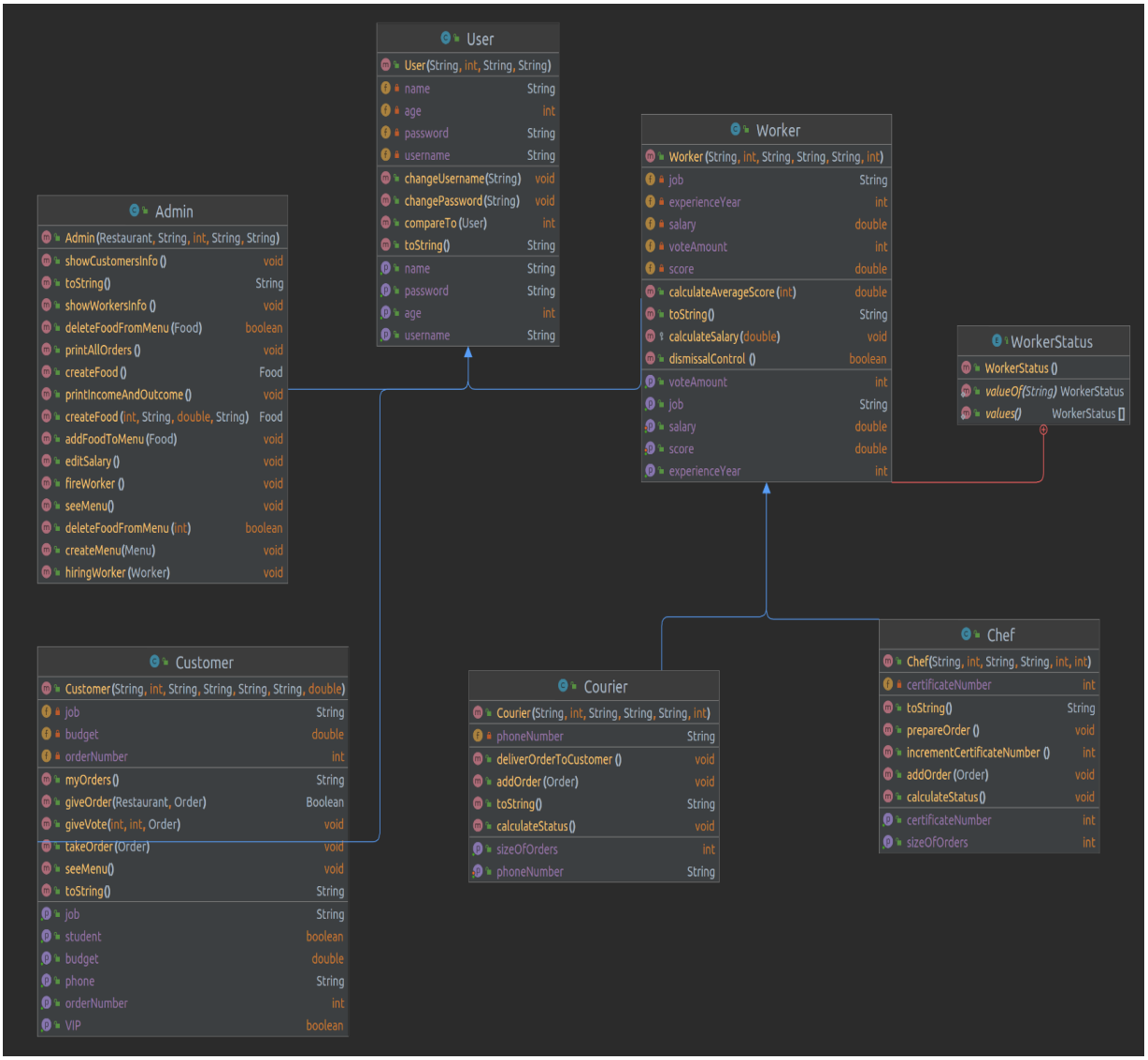

Courier
[User/Worker]
who can deliver ingredients orders, pick/drop orders, see the order list
Container of the Online Food Service and Restaurant Management System
Database
[Container: Text File]
Text contains all the accounts' information
Uses/logs in
Extract data from
Compiler
[Container: JVM]
JVM compiler provided from Oraclar
sees order list, pick/drop order
Restaurant
[System]
Mechanism that contains all the information and operations
updates itself
Admin
[User/Worker]
who can add/remove workers, edit the menu, change the prices of menu,
Database
[Container: Text File]
Text contains all the accounts' information
Uses/logs in
Extract data from
Compiler
[Container: JVM]
JVM compiler provided from Oraclar
adds/removes worker, updates orders and orices
Restaurant
[System]
Mechanism that contains all the information and operations
updates itself

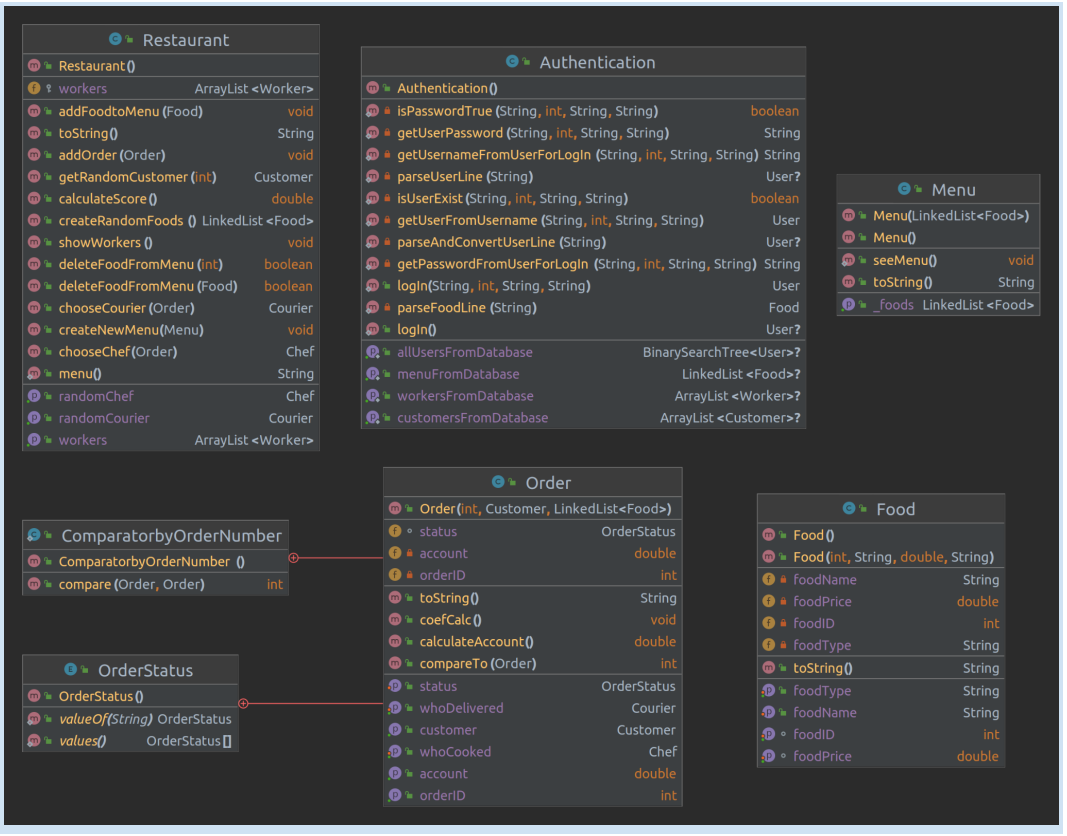
Level 3:

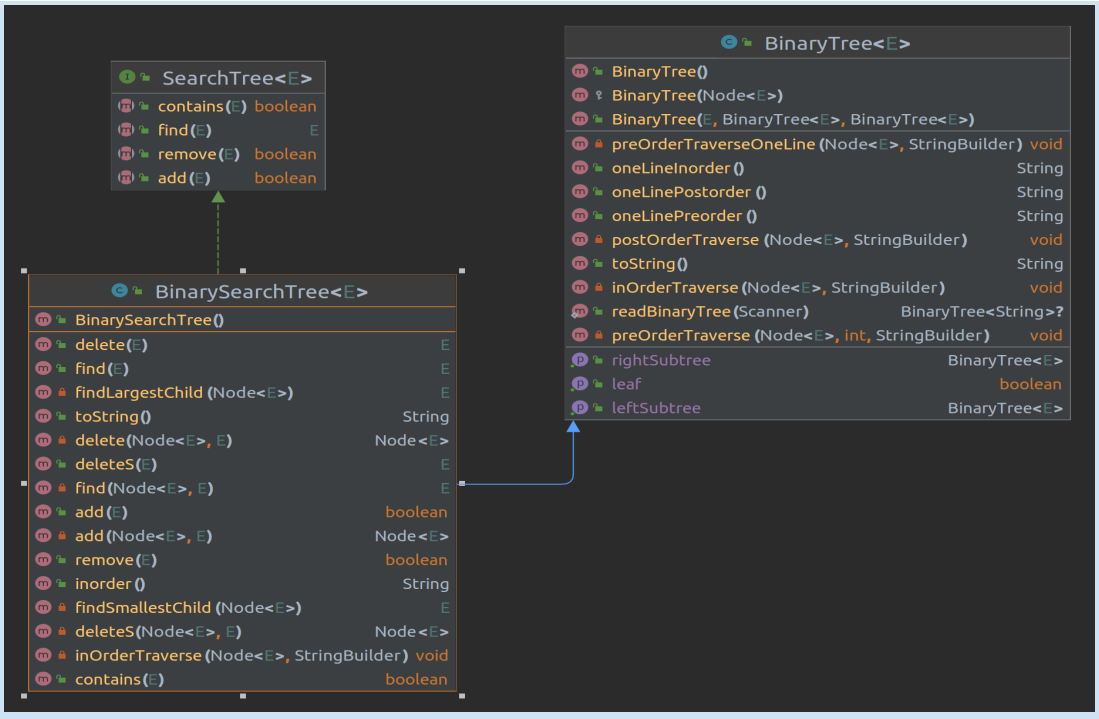

Admin
[User/Worker]
Chef
[User/Worker]
Courier
[User/Worker]
Customer
[User]
Make Calls
Make Calls
Make Calls
Make Calls
Single Page Application
[Component: Java]
Java script utilized for the tasks
[Java]
Online Food Service System Container
Compiler
[Component : JVM Oracle Compiler]
Predefined Java Methods
Reads Data
Security
Updates system according to the user actions
Updates monitoring objects
Database
[Component : .txt File]
Utilizes
Monitoring
[Component : Java]
Manipulates driver Java objects
Sends Data
Memory
[Component : Hardware of the computer]
Saves updates from user actions
Admin
[Object : User]
Sends Data
Sends Feedback
Restaurant
[Object]
Database
[Component : .txt File]
Displays
Predefined UI

7. CLASS DIAGRAMS

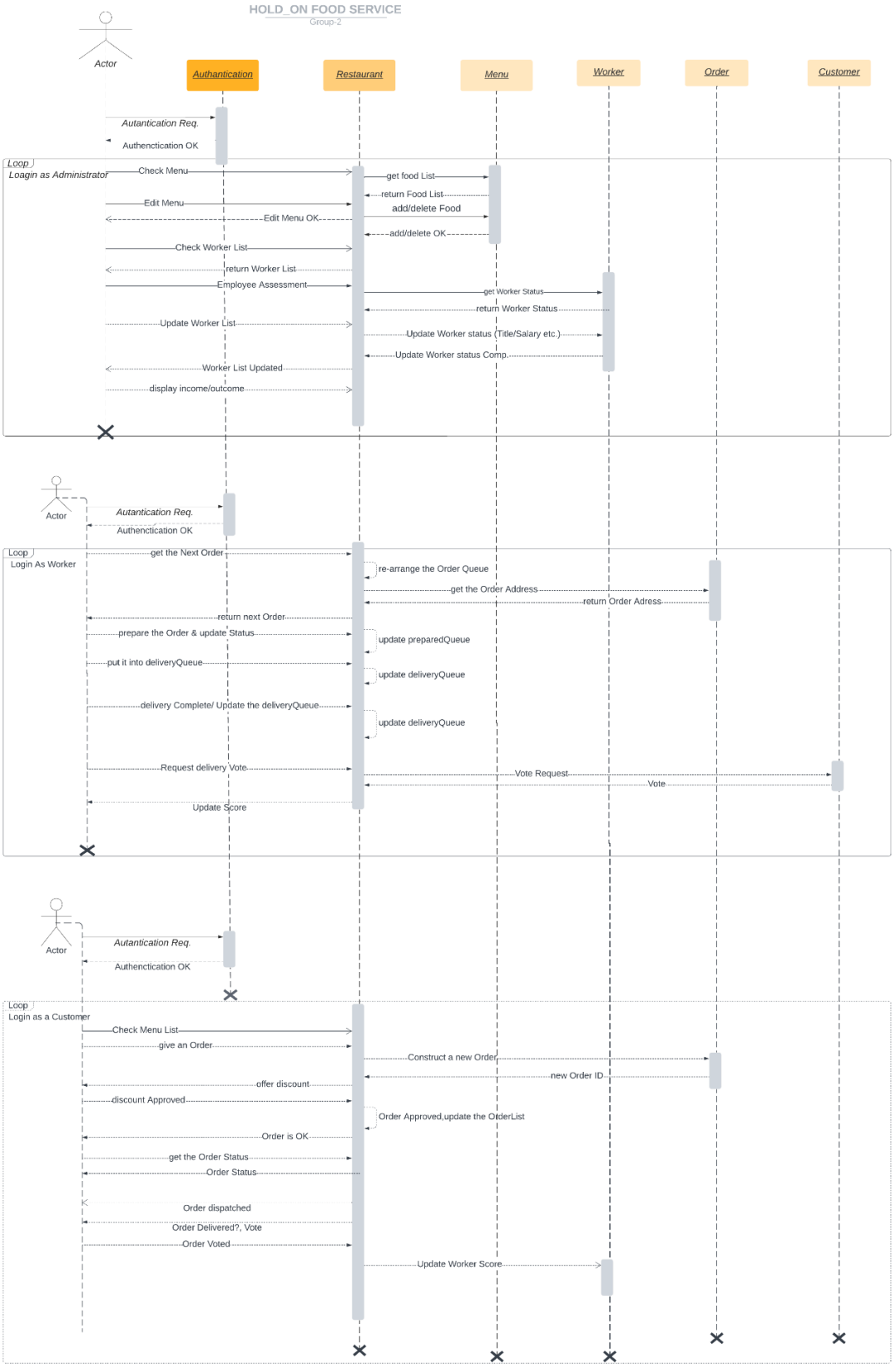

User
Worker
Customer
Admin
Courier
Chef
Food
Menu
Order
Restaurant
Authentication
package src.tree
package src.database


User
User(String, int, String, String)
name String
age int
password String
username String
changeUsername(String) void
changePassword(String) void
compareTo(User) int
toString() String
name String
password String
age int
username String
Admin
Admin(Restaurant, String, int, String, String)
showCustomersInfo() void
toString() String
showWorkersInfo() void
deleteFoodFromMenu(Food) boolean
printAllOrders() void
createFood() Food
printIncomeAndOutcome() void
createFood(int, String, double, String) Food
addFoodToMenu(Food) void
editSalary() void
fireWorker() void
seeMenu() void
deleteFoodFromMenu(int) boolean
createMenu(Menu) void
hiringWorker(Worker) void
Worker
Worker(String, int, String, String, String, int)
job String
experienceYear int
salary double
voteAmount int
score double
calculateAverageScore(int) double
toString() String
calculateSalary(double) void
dismissalControl() boolean
voteAmount int
job String
salary double
score double
experienceYear int
WorkerStatus
WorkerStatus()
valueOf(String) WorkerStatus
values() WorkerStatus[]
Customer
Customer(String, int, String, String, String, String, double)
job String
budget double
orderNumber int
myOrders() String
giveOrder(Restaurant, Order) Boolean
giveVote(int, int, Order) void
takeOrder(Order) void
seeMenu() void
toString() String
job String
student boolean
budget double
phone String
orderNumber int
VIP boolean
Courier
Courier(String, int, String, String, String, int)
phoneNumber String
deliverOrderToCustomer() void
addOrder(Order) void
toString() String
calculateStatus() void
sizeOfOrders int
phoneNumber String
Chef
Chef(String, int, String, String, int, int)
certificateNumber int
toString() String
prepareOrder() void
incrementCertificateNumber() int
addOrder(Order) void
calculateStatus() void
certificateNumber int
sizeOfOrders int

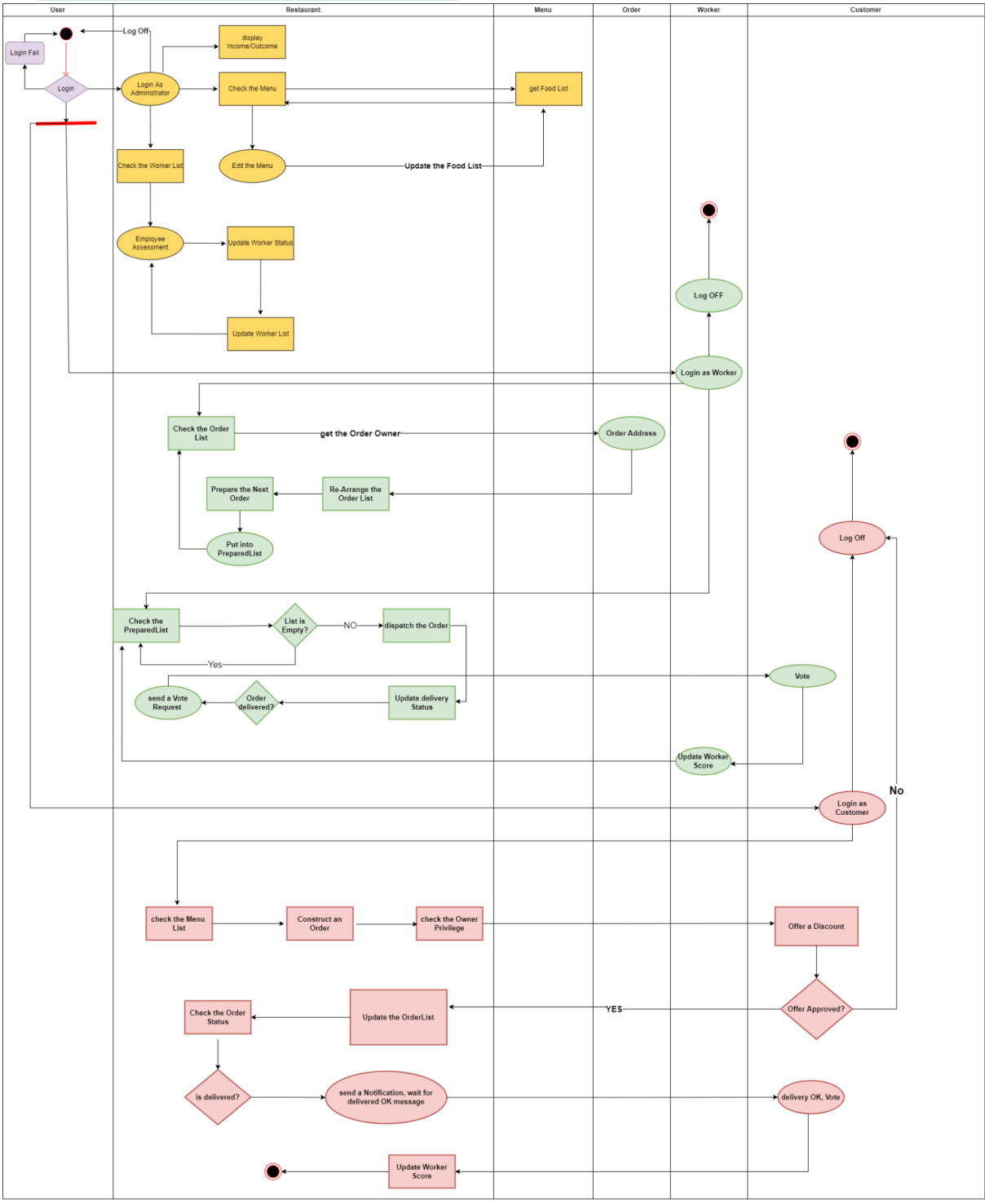

Restaurant
Restaurant()
workers ArrayList <Worker>
addFoodtoMenu (Food) void
toString() String
addOrder (Order) void
getRandomCustomer (int) Customer
calculateScore () double
createRandomFoods () LinkedList <Food>
showWorkers () void
deleteFoodFromMenu (int) boolean
deleteFoodFromMenu (Food) boolean
chooseCourier (Order) Courier
createNewMenu(Menu) void
chooseChef(Order) Chef
menu() String
randomChef Chef
randomCourier Courier
workers ArrayList <Worker>
Authentication
Authentication()
isPasswordTrue (String, int, String, String) boolean
getUserPassword (String, int, String, String) String
getUsernameFromUserForLogIn (String, int, String, String) String
parseUserLine (String) User?
isUserExist (String, int, String, String) boolean
getUserFromUsername (String, int, String, String) User
parseAndConvertUserLine (String) User?
getPasswordFromUserForLogIn (String, int, String, String) String
logIn(String, int, String, String) User
parseFoodLine (String) Food
logIn() User?
allUsersFromDatabase BinarySearchTree<User>?
menuFromDatabase LinkedList <Food>?
workersFromDatabase ArrayList <Worker>?
customersFromDatabase ArrayList <Customer>?
Menu
Menu(LinkedList<Food>)
Menu()
seeMenu() void
toString() String
_foods LinkedList <Food>
Order
Order(int, Customer, LinkedList<Food>)
status OrderStatus
account double
orderID int
toString() String
coefCalc() void
calculateAccount() double
compareTo (Order) int
status OrderStatus
whoDelivered Courier
customer Customer
whoCooked Chef
account double
orderID int
ComparatorbyOrderNumber
ComparatorbyOrderNumber ()
compare (Order, Order) int
OrderStatus
OrderStatus()
valueOf(String) OrderStatus
values() OrderStatus []
Food
Food()
Food(int, String, double, String)
foodName String
foodPrice double
foodID int
foodType String
toString() String
foodType String
foodName String
foodID int
foodPrice double


SearchTree<E>
contains(E) boolean
find(E) E
remove(E) boolean
add(E) boolean
BinarySearchTree<E>
BinarySearchTree()
delete(E) E
find(E) E
findLargestChild (Node<E>) E
toString() String
delete(Node<E>, E) Node<E>
deleteS(E) E
find(Node<E>, E) E
add(E) boolean
add(Node<E>, E) Node<E>
remove(E) boolean
inorder() String
findSmallestChild (Node<E>) E
deleteS(Node<E>, E) Node<E>
inOrderTraverse (Node<E>, StringBuilder) void
contains(E) boolean
BinaryTree<E>
BinaryTree()
BinaryTree(Node<E>)
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
preOrderTraverseOneLine (Node<E>, StringBuilder) void
oneLineInorder() String
oneLinePostorder() String
oneLinePreorder() String
postOrderTraverse (Node<E>, StringBuilder) void
toString() String
inOrderTraverse (Node<E>, StringBuilder) void
readBinaryTree (Scanner) BinaryTree<String>?
preOrderTraverse (Node<E>, int, StringBuilder) void
rightSubtree BinaryTree<E>
leaf boolean
leftSubtree BinaryTree<E>

8. SEQUENCE DIAGRAMS



9. ACTIVITY DIAGRAMS



10. NON-TRIVIAL IMPLEMENTATION DETAILS

Properties that makes the project implementation non-trivial are:

- Working as a group, rather than individually
- Concentrate on different aspects of a system as whole, rather than selecting one and work on it
- Being dependent other parts of the system, being affected from changes. This also requires stay up to date and follow updates
- Evolving to better and more reasonable way from planned at the beginning of the project.

In line with those criterias, we came up with solution methods to the project to work as a team. First of all, process is improved in github environment, to being up to date and following changes easily. Also looking back to comments shows us to evaluate the going of the project and measuring how well the initial decisions are followed, if required. Dividing the classes into managable parts and perform related operations inside that class provide us to capability of seeing dependent parts with others, make changes easier and see realations better. The point reached as we progressed through the process showed the lack/wrongness of some initial decisions, changing the menu is one of them. And those kind of properties are re-considered and removed if necessary.

Data structures to implement established system's components are selected considering their prominant features.

Workers and customers are hold as ArrayList in Restaurant class due to it's fast access by index.

To implement giveOrder, deliverOrderToCustomer LinkedList is used due to it's traverse fast, also adding new one takes constant time and removing, editing can be fastest performed by LinkedList.

Courier's order following system is designed as Priority Queue, in order to see the expensive order first and deliver it first.

When searching for customers from the database, binary search tree is used to perform the search process in the fastest way.

11. TEST CASES

A restaurant is established.

```
1  Restaurant kebelekCafe = new Restaurant();
```

Order instances are created

```
1  for (int i = 0; i < 10; i++) {
2    LinkedList<Food> foods = kebelekCafe.createRandomFoods();
3    Order order = new Order(i, customer, foods);
4    orders.add(order);
5  }
```

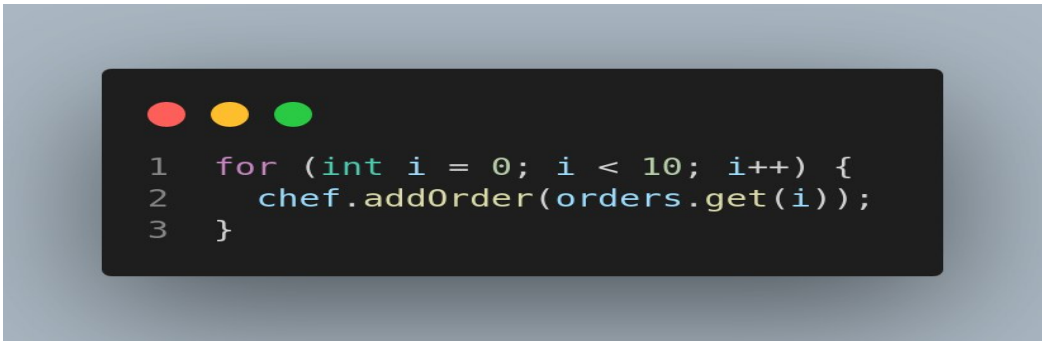
User database has been extended with adding new users.

```
for (int i = 0; i < 10; i++) {
  auth.createUser(new User("test", 10, "a" + i + ".th", "pass"));
}
```

Customer’s capability of order to given restaurant is tested.

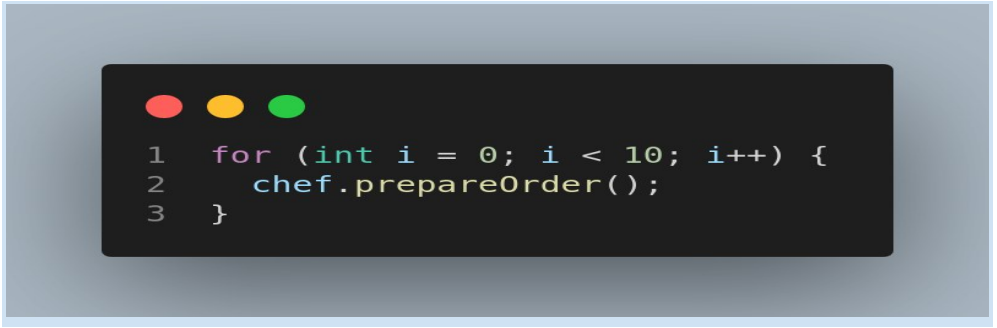
```
for (int i = 0; i < 10; i++) {
  customer.giveOrder(kebelekCafe, orders.get(i));
}
```

Chef's ability to see and cook the next order was tested.

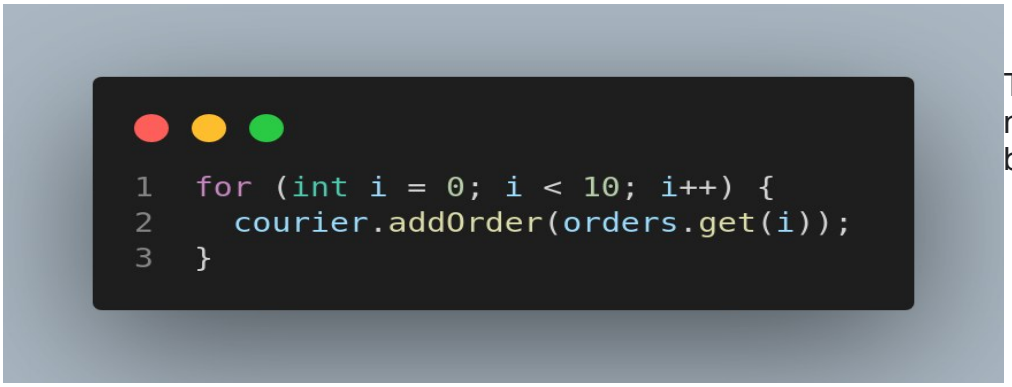


```
for (int i = 0; i < 10; i++) {
  chef.addOrder(orders.get(i));
}
```

Chef assigns meal status as “prepared” and it removes from chef’s order list.



```
for (int i = 0; i < 10; i++) {
  chef.prepareOrder();
}
```



```
for (int i = 0; i < 10; i++) {
  courier.addOrder(orders.get(i));
}
```

The courier adds a new order to the listto be ordered.

Courier delivers the first order in the queue (order with the highest cost) to the relevant customer

```
for (int i = 0; i < 10; i++) {
  courier.deliverOrderToCustomer();
}
```

```
Admin admin = new Admin(
  kebelekCafe,
  "Fatih Erdogan",
  40,
  "gtu1234",
  "1234"
);
```

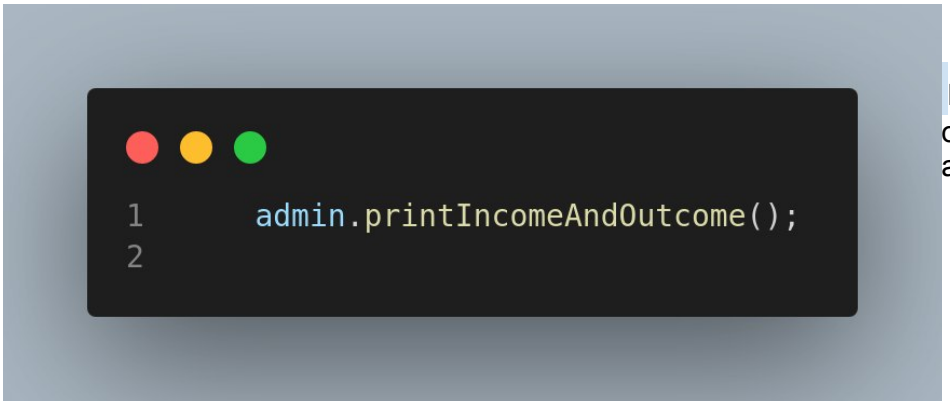
To control all system, an admin has been created that has capabilities mentioned in the user part of the report.

```
admin.showWorkersInfo();

```

Admin can see the workers info.

```
admin.showCustomersInfo();

```

Admin can see the customers info.



Restaurant's income and outcome can be seen by admin.

Admin can see all the orders.

If a worker is ranked over 4 by at least 10 customer, admin can fire the worker.

New workers can be added to restaurant, by looking their experience year, occupational capability etc.

```
admin.hiringWorker(
  new Worker("Ahmet", 30, "ahmet", "gtu1234", "student", 3)
);
```

Admin can edit the salary of worker according to determined coefficents, experience year etc.

```
admin.editSalary();
```

New foods can be created.

```
admin.createFood(3, "cacık", 15, "corba");
```

New food can be added to the menu.

```
admin.addFoodToMenu(new Food(3, "cacık", 15, "corba"));
```

Food can be removed from the menu.

```
1      admin.deleteFoodFromMenu(new Food(3, "cacık", 15, "corba"));
2
```

A new menu can be created.

```
1      admin.createMenu(new Menu());
2
```

Menu can be seen by admin totally.

```
1      admin.seeMenu();
2
```

12. PERFORMANCE ANALYSIS

a. Theoretical Analysis

Worker Class

METHOD NAME	COMPLEXITY
calculateSalary	Θ(1)
calculateAverageScore	Θ(1)
dismissalControl	Θ(1)

Admin Class

METHOD NAME	COMPLEXITY
ShowWorkersInfo	Θ(n)
ShowCustomersInfo	Θ(n)
printIncomeAndOutcome	Θ(1)
printAllOrders	Θ(n)
FireWorker	Θ(n)
HiringWorker	Θ(1)
editSalary	Θ(n)
CreateFood	Θ(1)

Menu Class

METHOD NAME	COMPLEXITY
seeMenu	Θ(n)

Customer Class

METHOD NAME	COMPLEXITY
myOrders	Θ(n)
give Vote	Θ(1)
giveOrder	Θ(1)
takeOrder	Θ(1)
isVIP/isStudent	Θ(1)

Chef Class

METHOD NAME	COMPLEXITY
addOrder	Θ(1)
prepareOrder	Θ(1)
getCertificateNumber	Θ(1)
incrementCertificateNumber	Θ(1)
calculateStatus	Θ(1)
getSizeOfOrders	Θ(1)
toString	Θ(1)

User Class

METHOD NAME	COMPLEXITY
just setters/getters	Θ(1)

Courier Class

METHOD NAME	COMPLEXITY
addOrder	Θ(log(n))
deliverOrderToCustomer	Θ(log(n))
setPhoneNumber	Θ(1)
getPhoneNumber	Θ(1)
calculateStatus	Θ(1)
getSizeOfOrders	Θ(1)
toString	Θ(1)

Authentication Class

METHOD NAME	COMPLEXITY
logIn	Θ(n)
createUser	Θ(log(n))
getMenuFromDatabase	Θ(n)
getCustomersFromDatabase	Θ(n)
getAllUsersFromDatabase	Θ(n.log(n))
getUsernameFromUserForLogIn	Θ(log(n))
getPasswordFromUserForLogIn	Θ(n)
isUserExist	Θ(log(n))
isPasswordTrue	Θ(n)
getUserPassword	Θ(log(n))
getUserFromUsername	Θ(log(n))

parseFoodLine	Θ(n)
parseAndConvertUserLine	Θ(n)
parseUserLine	Θ(n)

Order Class

METHOD NAME	COMPLEXITY
coefCalc	Θ(1)
calculateAccount	O(n)
toString	O(n)
Compare	Θ(1)

Food Class

METHOD NAME	COMPLEXITY
toString	Θ(n)

Restaurant Class

METHOD NAME	COMPLEXITY
createRandomFoods	Θ(1)
calculateScore	Θ(n)
addOrder	Θ(1)
chooseChef	O(n)
chooseCourier	O(n)
showWorkers	Θ(n)
menu	Θ(n)

deleteFoodFromMenu(int)	O(n)
deleteFoodFromMenu(food)	O(n)
addFoodtoMenu	Θ(1)
createNewMenu	Θ(1)
toString	Θ(n)

b. Experimental Analysis

```
TESTING MAIN DATA STRUCTURE METHODS FOR 10 INPUTS
BST createUser() Method For 10 Inputs Time: 11329
Customer giveOrder() Method For 10 Inputs Time: 1624
Chef addOrder() Method For 10 Inputs Time: 296
Chef prepareOrder() Method For 10 Inputs Time: 353
Courier addOrder() Method For 10 Inputs Time: 582
Courier deliverOrderToCustomer() Method For 10 Inputs Time: 619

TESTING MAIN DATA STRUCTURE METHODS FOR 100 INPUTS
BST createUser() Method For 100 Inputs Time: 14066
Customer giveOrder() Method For 100 Inputs Time: 6149
Chef addOrder() Method For 100 Inputs Time: 658
Chef prepareOrder() Method For 100 Inputs Time: 528
Courier addOrder() Method For 100 Inputs Time: 1415
Courier deliverOrderToCustomer() Method For 100 Inputs Time: 3738

TESTING MAIN DATA STRUCTURE METHODS FOR 1000 INPUTS
BST createUser() Method For 1000 Inputs Time: 51792
Customer giveOrder() Method For 1000 Inputs Time: 25014
Chef addOrder() Method For 1000 Inputs Time: 5179
Chef prepareOrder() Method For 1000 Inputs Time: 5906
Courier addOrder() Method For 1000 Inputs Time: 8698
Courier deliverOrderToCustomer() Method For 1000 Inputs Time: 25532
```

BST createUser() -> Uses binary search tree data structures T(n) = Θ(logn)

Customer giveOrder() -> Uses LinkedList data structures T(n) = Θ(1)

Chef addOrder() -> Uses Queue data structures T(n) = Θ(1)

Chef prepareOrder() -> Uses Queue data structures T(n) = Θ(1)

Courier addOrder() -> Uses Priority Queue data structures T(n) = Θ(logn)

Courier deliverOrderToCustomer() -> Uses Priority Queue data structures T(n) = Θ(logn)