



# [CIS-641 ASSIGNMENT 5]

## ABSTRACT:

IN THIS ASSGINMENT WE COVERED 6 CLASS DIAGARAMS OF OUR APPLICATION WITH POST CONDITION AND PRE CONDITION AND ALSO ADEDD THE OBJECT DIAGRAMS.

## [TEAM MEMBERS]

ANUDEEP AMARA: G02495899

SUNDEEP GANTYADA: G02494036

RAMYA NAVLURI: G02494845

Method name: allocation

class name: location

SD: 03

Clients: users, Restaurants

Associated use cases: user, Restaurant, Admin.

Description of responsibility: while user registering himself into the application, he needs to select the location. It will store that data and assign the location-id to user.

Arguments Received: The moment user selected the location it will call that method and update the user details in the database.

Types of values Returned: This will list the location which were assigned restaurants and \$00 ok. other will return 200 BAD Request.

Pre-condition: The pre-condition will be need to store the locations on DB so that user can select few locations.

Post-condition: whenever user has been selected and assigned to user. It will show all the restaurants under that location with the location-id.



Method Name : Login to Application

ClassName : UserLogin

So: 01

Client : users

Associated use case : login, Signup

Description of Responsibility : for this condition when ever user tries to login there should be username and password. It should validate the values from the data box. It should not suppose to be null.

Arguments Received : It will call the usercontroller method with the @RequestMapping from the controller and it will go and verify the DB. It will be coming the user data.

Types of value returned : It will return the data from the DB. Other wise it will throw HTTP 500 or HTTP 400 status ~~error~~ message. or 200 OK.

Pre condition : for this case the precondition will be there needs to be the valid username and password. and it should not suppose to be null.

Post condition : we are using the static variable  $used = 0$  ; It will never go and fetch the user details from the repository manually.

@Repository

Public interface User



method name : FoodController

class : food

so : 05

clients : users

Associated Uays : user, restaurant, cart, orders

Description of responsibilities : On this class of user store all the data of the food items for the application. Along with the attributes. This will be seen in the restaurant - it.

Arguments received : The user traverses the application on the food page they can see the food items and they will be seen along with the restaurant - it and food - it.

Types of values returned : It will return all the food items in that particular restaurant and display them. user can able to add them in cart. so on. this response.

Pre-condition : All the food items need to have an id and loaded into the DB. along with the restaurant id. on the location.

Post-condition : This will go to the DB and fetch all the data on the restaurants and display all the food items for the user.



Method name : Restaurant Controller

Class : Restaurant

IO : 04

Clients : users

Associated Tables :- user, food, location

Description of responsibilities :- This class will be responsible to act as the Restaurant details with the administrator and it will contain the Restaurant ID, that will store in the DB.

Arguments Received :- The Admin needs to add the restaurants in the DB with the IDs, and it will be allocated in the location IDs.

Types of value received :- This class will fetch all the data from location table and show all the restaurants with 200 OK HTTP response.

Pre-condition :- The Pre-condition will be restaurants not to store in the database and it will be having an location ID's to show the user.

Post-condition :- The moment user goes to the Restaurant page it will display all the restaurant details in that particular location.



Method Name : Controller

Classname : Controller

ID : 08

Clients : users

Associated usecase : user, food, orders.

Description of responsibility : when ever user added items into his cart. There will be a list of items which will be displayed. user should not be able to be null.

Arguments received : when ever Controller tries to fetch the cart items list, it will pass the cart items into the cart with his userID. from the DB.

Type of value returned : It will return the HTTP status 200 OK or 500 Bad request.

Pre - condition : The pre condition would be the user should have items in the cart.

Post - condition : This will fetch the data from the

Controller and validate the userID, and pass the cart items into cart.



Method name : orderItems()

Class : orderItems

SP : 06

Clients : users

Associated classes : cart, user

Description of responsibility : This use case is responsible for the order history to get all the data of the user booking at the local time and local date. of user get with orderItem - it ;

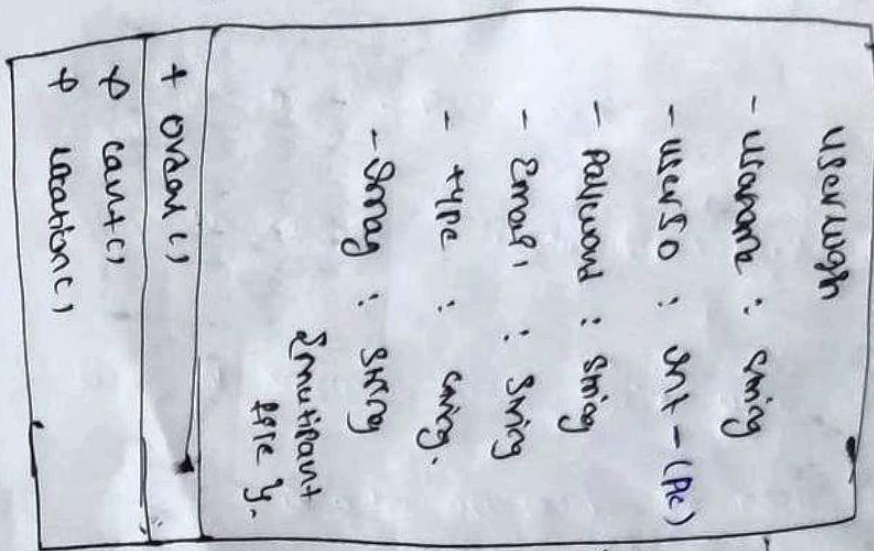
Arguments received : The orderItems - it will store all the data of user and it will process all the items from the DB.

Types of values received : This case will return the table data and it will show the 200 OK response and 500 HTTP status bad response.

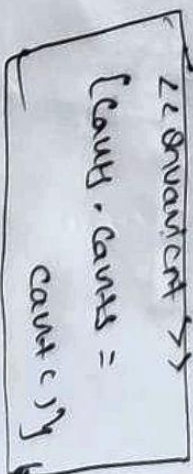
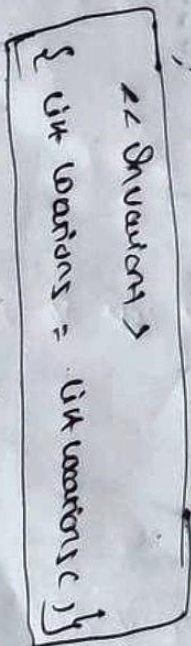
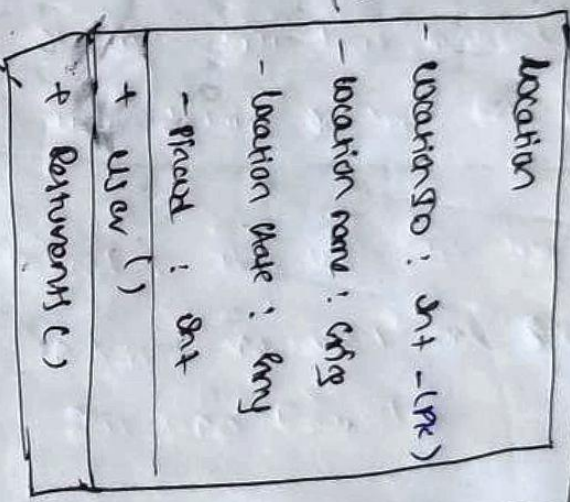
Pre-condition : The pre-condition for this case will be all the items in the cart need to be stored and validated under the orderItems.

Post-condition : It will collect all the table data of the DB and display the cart list of items from the user - it and cart - it.

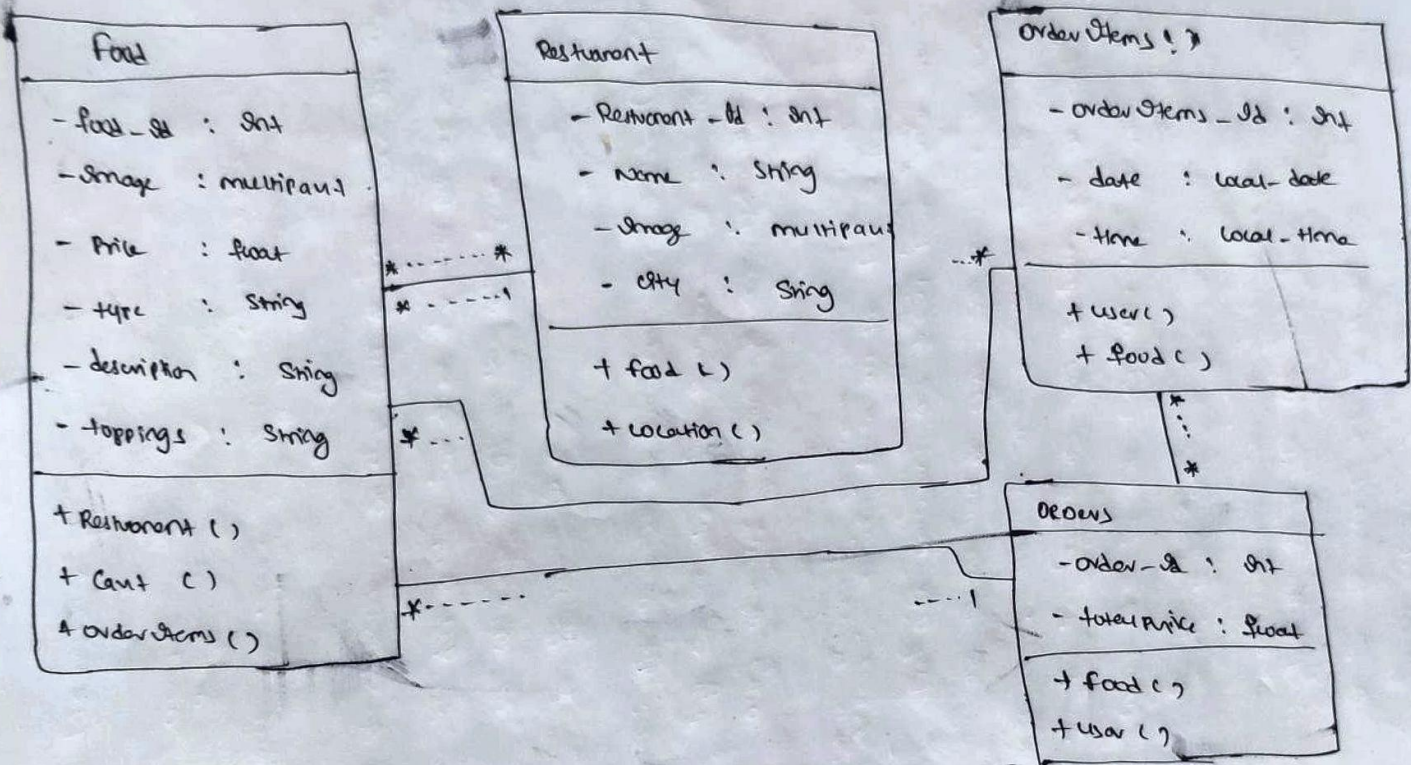




user cant



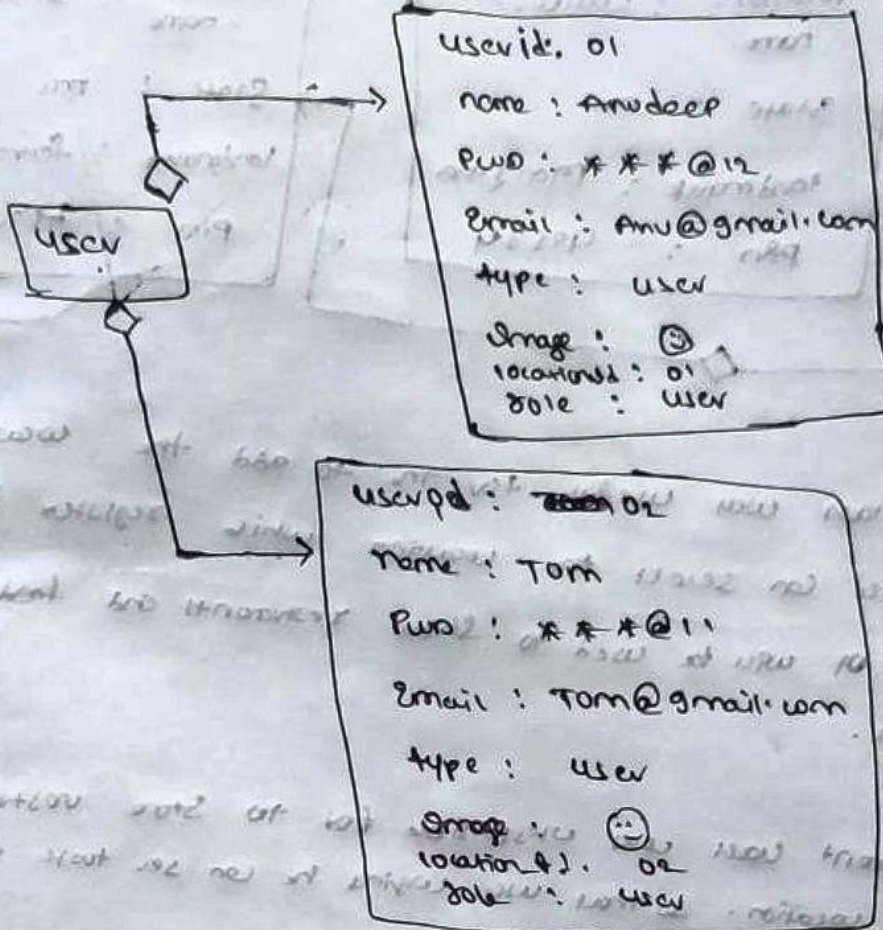






## OBIEU Diagram:

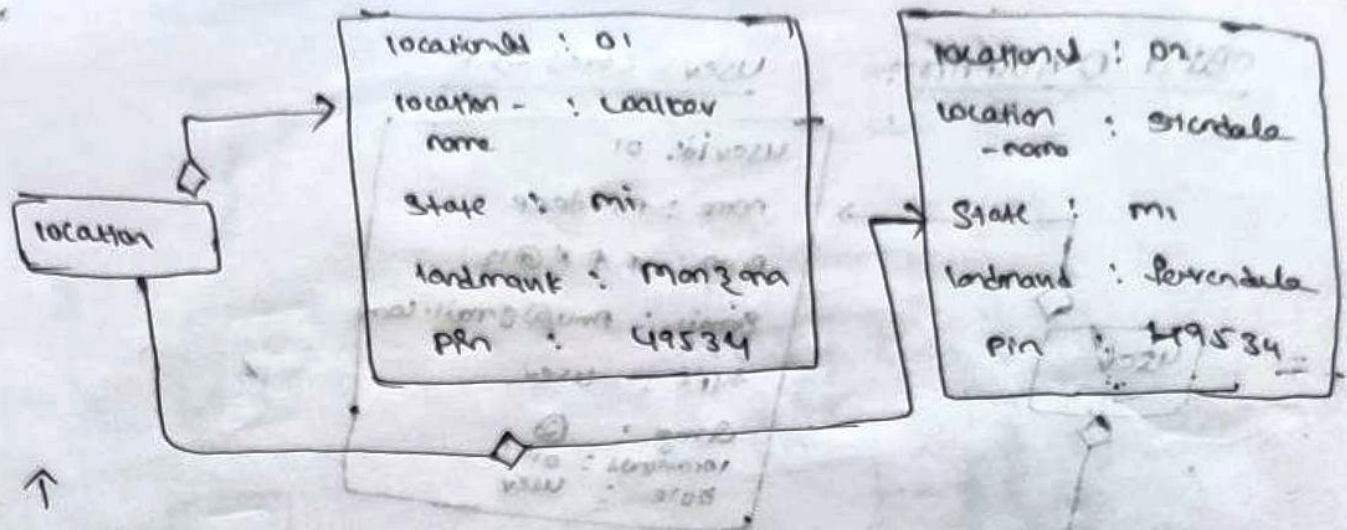
### User - class -> 1



## class Description:-

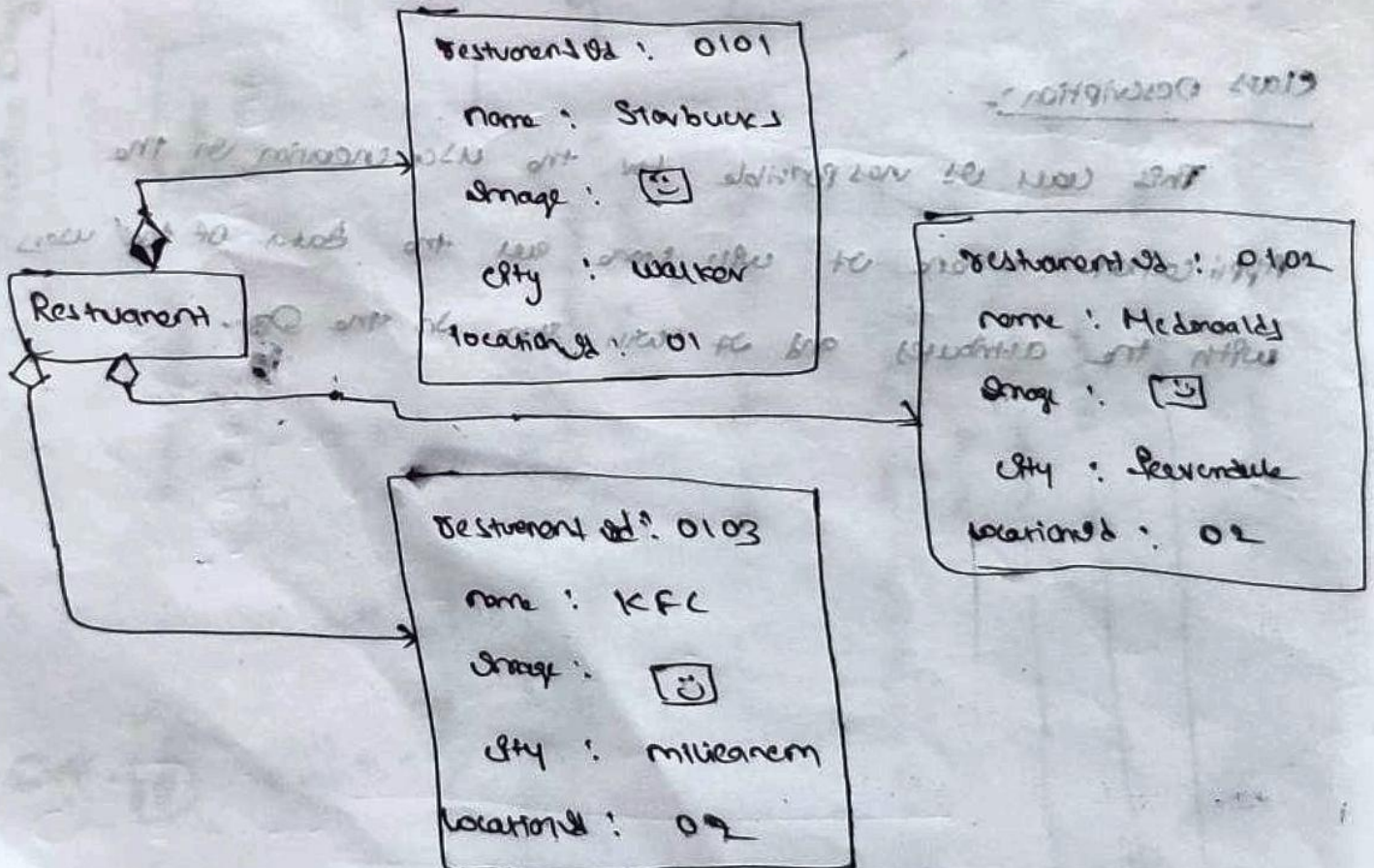
This class is responsible for the user creation in the Application where it will store all the data of the users with the attributes and it will store in the DB.






Description: This class was useful for the to add the location. with that we can select the location while register into application. It will be used to show restaurants and fast in the model.


Description: This restaurant class was responsible for to store restaurants under the location. so that user login he can see those restaurants.

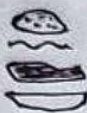




food

Food Id : 0101  
name : coffee  
Image :   
Description : latte with blended seeds.  
Price : 3.0\$  
type : veg  
toppings : caramel  
Restaurant Id : 0101

Food Id : 0203  
name : pizza  
Image :   
Description : cheese burst  
022A  
Price : 9.0\$  
type : veg  
toppings : onions, lettuce  
Restaurant Id : 0202

Food Id : 0304  
name : Burger  
Image :   
Description : Pome burger with veggies and mayo  
Price : 7.0\$  
type : non-veg  
toppings : mayo, lettuce  
Restaurant Id : 0202

Restaurant

Home Code

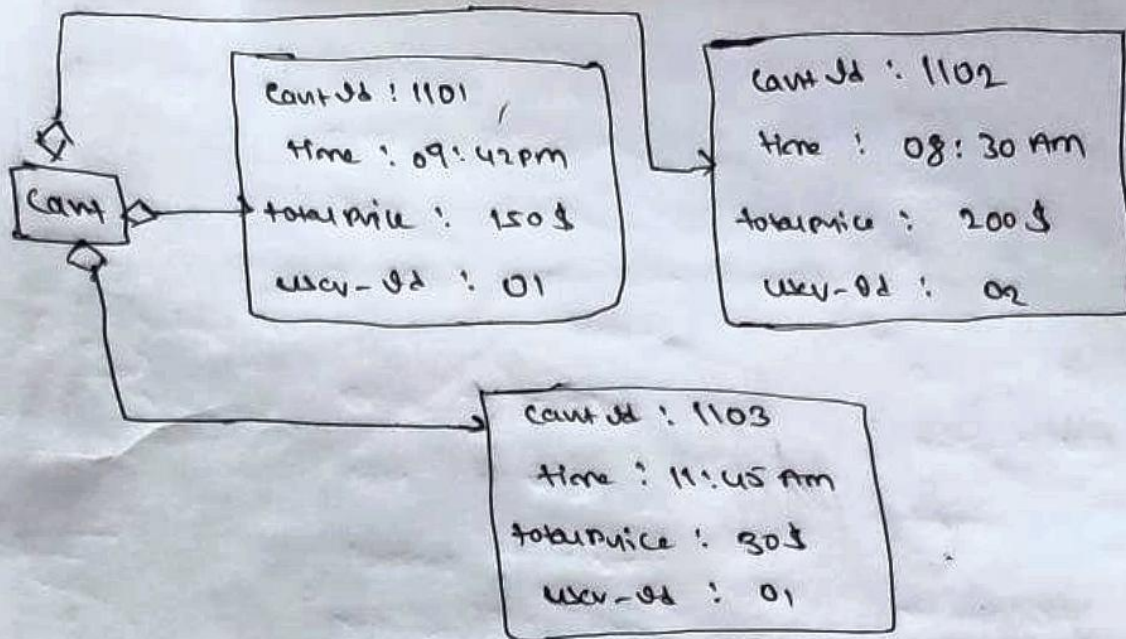
Restaurant Code

Description:- This class was responsible for the food items to store in the restaurants and users will be able to see and add to cart and place the order.

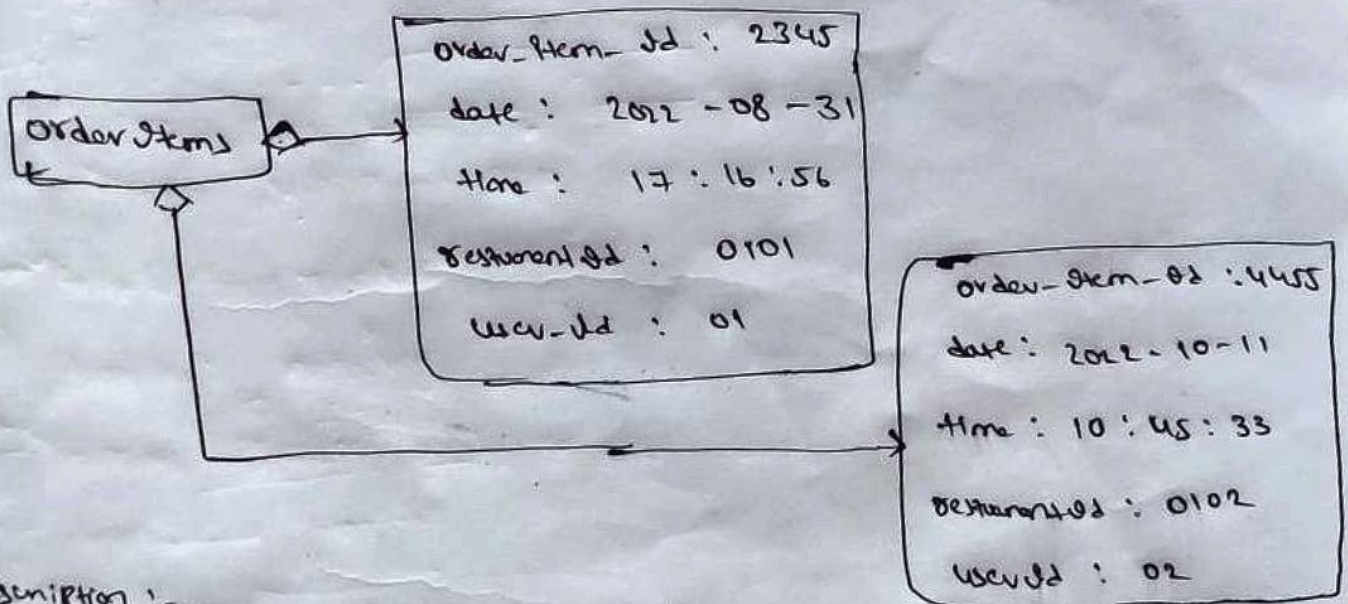
\_\_\_\_\_

USER





Description:- This class is responsible for the user added items in the cart and he can see and proceed for the checkout.



Description:- This class was used to fetch all the orders which was placed by the user with his user-id.

```

//method for the login to application:
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int userId;
    private String name;
    private String username;
    private String password;
    private String email;
    private String type;
    private String role;

    @Lob
    private String image;

    @OneToMany(mappedBy = "user", targetEntity = Orders.class, fetch = FetchType.EAGER)
    private List<Orders> orders;

    @OneToOne(mappedBy = "user", fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true)
    private Cart cart;

//Pre condition for this method:
@RequestMapping(value = "/login")
public String userloginpage(Model model, @ModelAttribute("user") User user) {
    List<User> users = userService getUsers();

    List<User> us = new ArrayList<User>();
    System.out.println(user.getName() + user.getPassword());

    List<User> us = new ArrayList<User>();
    System.out.println(user.getName() + user.getPassword());

    List<User> user1 = users.stream()
        .filter(e -> e.getUsername().equals(user.getUsername()) && e.getPassword().equals(user.getPassword()))
        .collect(Collectors.toList());

    for (User u : user1) {

//post condition for this method.
        if ((u.getUsername().equals(user.getUsername()) && u.getPassword().equals(user.getPassword()))) {

            userId = u.getUserId();

            if (u.getRole().equalsIgnoreCase("admin")) {
                for (User usl : users) {
                    if (usl.getUserId() == userId) {
                        us.add(usl);
                    }
                }
                model.addAttribute("us", us);
                return "adminhomepage";
            } else {
                for (User usl : users) {
                    if (usl.getUserId() == userId) {
                        us.add(usl);
                    }
                }
                model.addAttribute("us", us);

                return "homepage";
            }
        }

//invariant for this method to verify the data.
    } else {
        model.addAttribute("error", new ErrorDto("invalid user and password"));
        return "userlogin";
    }
}

```



```

//method for the login to application:
@Entity
public class CartItems {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int cartItemId;
    private int foodId;
    private String foodName;

    private String description;
    private double price;
    private String type;

    private LocalDateTime time;

    @ManyToOne(cascade = CascadeType.ALL)
    private Cart cart;

    //Pre condition for this method:

    @RequestMapping("/cartlistt")
    public String getCart(Model model, @ModelAttribute(name = "cart") Cart cart) {
        List<Cart> carts1 = service.getAllCart();
        List<Food> carts = new ArrayList<Food>();
        User user = new User();
        //post condition for this method.
        userId = new UserController().userId();
        for (Cart c : carts1) {
            if (c.getUser().getUserId() == userId) {
                List<Food> food1 = c.getFood();
                for (Food f : food1) {
                    carts.add(f);
                }
            }
        }

        //Invariant for this method.
        model.addAttribute("carts", carts);

        return "cart";
    }
}

```

```

@Controller
public class Foodcontroller {

    static int userId = 0;

    @Autowired
    private FoodServiceImpl service;

    @Autowired
    private CartServiceImpl cartService;;

    @RequestMapping("/foodlist/{restuarentId}")
    public String getallfoods(@PathVariable int restuarentId, Model model,
        @ModelAttribute(name = "food") Restuarent restuarent) {

        List<Food> listfood = service.getallfoods(restuarentId);
        model.addAttribute("listfood", listfood);

        return "food";
    }

    @RequestMapping("/delete/{foodId}")
    public String deleteitems(@PathVariable int foodId) {

        cartService.deleteFood(foodId);

        return "redirect:/homepage";
    }

    @RequestMapping("/uploadFoodImage")
    public ModelAndView UploadFoodImage() {
        ModelAndView mav = new ModelAndView("uploadFoodImage");
        List<Food> foods = service.getAllfoods();
        mav.addObject("foods", foods);
    }
}

```

```

@Override
public List<Food> getallfoods(int restuarentId) {
    Restuarent rest = restRepo.findById(restuarentId).orElse(null);
    List<Food> foods = rest.getFoods();

    return foods;
}

@Override
public List<Food> showFoodsbyfoodId(int foodId, int userId) {
    Food food = repo.findById(foodId).orElse(null);
    List<Food> foods = new ArrayList<Food>();

    User users = new User();
    User user = userRepo.findById(userId).orElse(null);

    foods.add(food);
    Cart cart = new Cart();
    cart.setUser(user);
    cart.setFood(foods);
    cart.setTotalprice(food.getPrice());
    Cart cart1 = cartrepo.save(cart);

    return foods;
}

@Override
public Cart getallcarts(int foodId, int userId) {

    return null;
}

@Autowired
private FoodServiceImpl foodservice;

@RequestMapping("/restuarent")
public String getAllrestuarents(Model model) {

    userId = new UserController().userId();

    List<Restuarent> listrest = service.getAllrestuarents(userId);

    model.addAttribute("listrest", listrest);
    return "restuarent";
}

@RequestMapping("/newRestuarent")
public String addRestuarent(@ModelAttribute("restuarent") Restuarent restuarent, Model model) {
    List<Location> listlocations = service.listlocations();
    model.addAttribute("listlocations", listlocations);

    return "New_Restuarent";
}

@RequestMapping(value = "/saverestFood", method = RequestMethod.POST)
public String saveFood(@ModelAttribute("food") Food food, @ModelAttribute("restuarent") Restuarent restuarent,
    @RequestParam("locationId") int locationId, Model model) {

    service.save(restuarent, locationId);

    id = restuarent.getRestuarentId();

    return "dummy1";
}

```



```

@Autowired
private LocationRepository locrepo;

@Override
public List<Restuarent> getAllLocations() {
    return repo.findAll();
}

@Override
public List<Restuarent> getAllrestuarents(int userId) {

    List<Restuarent> rest = repo.findAll();
    User user = userrepo.findById(userId).orElse(null);
    List<Restuarent> rest1 = rest.stream()
        .filter(e -> e.getLocation().getLocationId() == user.getLocation().getLocationId())
        .collect(Collectors.toList());

    return rest1;
}

@Override
public List<Location> listlocations() {
    // TODO Auto-generated method stub
    return locrepo.findAll();
}

@Override
public void save(Restuarent restuarent, int locationId) {
    Location location = locrepo.findById(locationId).orElse(null);
    restuarent.setLocation(location);
    repo.save(restuarent);
}

@RequestMapping("/history")
public String gethistory(Model model) {
    Map<searchDto, List<Food>> map = new HashMap<searchDto, List<Food>>();

    List<OrderItems> itemList = service.getAllItems();

    for (OrderItems i : itemList) {
        List<Food> foods1 = i.getFood();
        List<Food> foods = new LinkedList<Food>();
        foods.addAll(foods1);
        searchDto s = new searchDto();
        s.setOrderItemId(i.getOrderItemId());
        s.setDate(i.getDate());
        s.setTime(i.getTime());

        double sum = foods.stream().map(e -> e.getPrice()).reduce(0.0, (a, b) -> a + b);
        s.setTotalPrice(sum);

        map.put(s, foods);
    }

    model.addAttribute("map", map);
    return "history";
}

@RequestMapping("/paysuccess")
public String paySucess(Model model, @ModelAttribute("OrderItems") OrderItems orderitems) {

    service.additem(orderitems);
    model.addAttribute("orderitems", orderitems);

    return "paymentsucess";
}

```

```

@Override
public OrderItems additem(OrderItems orderitems) {

    List<Cart> carts = cartrepo.findAll();
    List<Cart> cart = new LinkedList<Cart>();

    userId = new Usercontroller().userId();
    for (Cart c : carts) {
        if (c.getUser().getUserId() == userId) {
            cart.add(c);
        }
    }
    List<Food> foods = new LinkedList<Food>();
    OrderItems item = new OrderItems();

    for (Cart cl : cart) {
        foods.addAll(cl.getFood());
    }
    Restuarent re = null;
    for (Food food : foods) {
        re = food.getRestuarent();
    }
    item.setFood(foods);
    LocalDate date = LocalDate.now();
    item.setDate(date);
    LocalTime time = LocalTime.now();
    item.setTime(time);
    item.setRestuarent(re);
    item.setUser(user);

    return repo.save(item);
}

```