Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: QRDER

Low Level Design Report

Mustafa Oğuz Güngör

Taylan Bartu Yoran

Mehmet Akif Kılıç

Supervisor: Varol Akman

Jury Members: Uğur Güdükbay, Fazlı Can

Low Level Design Report
Oct 5, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction

Comparing last decades, more people prefer eating outside rather than at home. Since there may be many reasons [2] , it can be said making restaurants serve meals fast and good has become harder. Since number of people who go to well known restaurants increases, restaurant owners try to find a new ways to maintain quality of dishes and restaurant. Since there are some knows standards [2] , it may be impossible to meet those when restaurant is too crowded. Also, these standards are dependent to waiters / waitresses. Finding a good waiter or making menu updated and well designed may be costly. So , any owner that wants good service quality and make this sustainable must pay big amounts of money. However, “Qrder” aims to make ordering and payment online so that restaurants may focus on food quality rather than getting order and checking the bill. The key idea behind the “Qrder” is make ordering, choosing the meal , checking the bill and displaying the menu digital. So that, waiters will have specific missions like delivering the meal instead of being responsible for any mission. To sum up . Qrder automizes the order and payment process so that it will reduce the workload of restaurants. For example, imagine yourself going a crowded restaurant with your friends. You would waste a lot of time waiting for a water to make an order or making payment. When that is the case, Qrder helps you in that case. If you would make an order and payment by using 3 Qrder, you would not had to waste time for ordering and payment. Furthermore, if restaurants let Qrder to be used, their responsibility will be decreased. This application can be used for different purpose. For example, a restaurant wants to change their menu design anytime they want. For example, when they want to update the price of the meal , they can change it instantly. Another example is changing order. When clients want to change/ cancel the order, they can do it via Qrder. So, to sum up , Qrder let clients and restaurants to changes their decisions instantly. In this report, analysis of the system will be demonstrated. We will start by giving information about existing systems and their scopes. Then, Qrder will be explained detailed and it's highlighting features will be demonstrated. After that, functional , nonfunctional and pseudo requirements will be showed. So that, we will have an idea about scope of the application and technologies that we might need. Then, system models and diagrams will be generated. Then, possible scenarios will be produced to guess what may happen. Later that, screen mock-ups and navigational paths will be created. Finally, social dimensions will be considered.

In this report, it is aimed to provide information about low level design of the Qrder system. Low level design includes detailed information about design phase and widely introductory step before implementation. Thus, this report explains trade-offs in terms of quality attributes, packages of the back-end and front-end systems and detailed explanation of the classes that reside in these systems. Main scope of the report is to determine design choices and generate a structure according to these choices.

1.1 Object Design Trade-Offs

Efficiency vs Concurrency

There were a lot of ways to implement communication mechanisms between endpoints. Easiest and the most efficient one to use REST API in whole communication due to TCP based structure and easy-to-use libraries,but , repetitive periodic REST requests are needed to establish data concurrency in all endpoints.On the other hand, socket programming or lower level mechanism(which is MQTT protocol for our project)can also be used to establish a better data concurrency.However, it results in additional network traffic and busier ports for back-end with low utility rates. Therefore, the communication model becomes more inefficient in this approach. Between these two, concurrency of the data is more critical for the Qrder system because the main purpose of the project is to reduce waiting times in restaurants. Thus, MQTT protocol will be used for communications between back-end and front-end to provide a communication model with better concurrency[6].

Feasibility vs Extensibility

As it is mentioned in the first trade-off, MQTT protocol will be used for communication. MQTT is less feasible than REST API due to server support and lack of security mechanism. There are publish/subscribe mechanisms instead of point-to-point communication in MQTT protocol. Therefore, messages are reachable by other subscribers and to avoid the leak of the information, messages have to be encrypted. On the other hand, REST API uses point-to-point communication, so messages cannot be reached by other endpoints. Nevertheless, we will use encryption in REST messages to strengthen the security, but it is not as necessary as MQTT protocol. Thus, MQTT is less feasible than REST API. However, MQTT has more extensibility potential because it is implementable on digital blocks. In later updates, an alternative guest paging system may be integrated with Qrder. Extensibility is more critical for the offered system because there can be additional features after first release.

Scalability vs Efficiency

In the back-end server, there were two options to utilize individual sessions which are keeping state in memory and another one is using authentication tokens to identify the owner of the request. Keeping state is a more scalable solution in terms of implementation and improvability, but it uses more memory space in the back-end and becomes inefficient as connections increase. On the other hand, stateless servers use no ram and cache memory in the back-end server but it is harder to implement and add an encryption phase for further uses. Thus, it is more efficient but less scalable. For Qrder project, inefficiency with increasing connections may cause server memory overflow,and so crashes may occur at the back-end server. Therefore, using authentication tokens is chosen to prevent crashes at the back-end server to increase memory efficiency.

Functionality vs Usability

This trade-off is made differently in client and admin endpoints.

Admin

In the front-end for the admin side, there should be a lot of functionality to manage all processes related to orders of the customer. Modification of the data mainly done in this endpoint, therefore the user interface should be more functional instead of more usable. Thus, Interface of the admin side may be harder to learn,but it provides a lot of functionalities to modify and to manage all transactions related to orders.

User

In the front-end for the user side, the main function of the application should be to create orders and send them to the back-end server. Therefore, there is not much functionality needed in the user interface, instead a more usable one should be prefered. Therefore, the usability aspect is chosen in this side of the system instead of functionality to provide an interface that is easier to learn about.

1.2 Interface Documentation Guidelines

In the following sections, whole class names are named in standard format which means meaningful and singular. This procedure applies for variable and method names too. [7]While describing a class, class name, the purpose of the class, attributes, methods and purposes of methods will be provided. In the following table, a sample table is shown below.

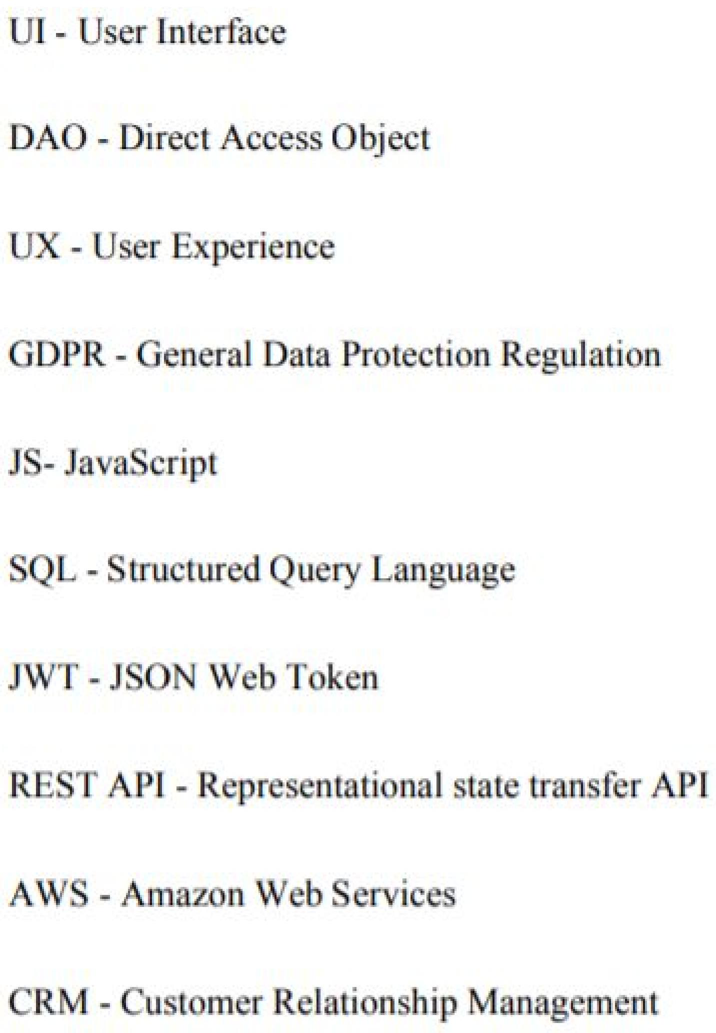
class Sample	
why class is created	
Attributes	
private int sample1 private String sample2	
Methods	

public String sampleMethod1
public int sampleMethod2
	return sample2
	return sample1

1.3 Engineering Standards

UML guidelines will be followed strictly for the classes, diagrams, scenarios and any subsystem compositions. (UML KAYNAK BUL). Since we used it in our previous projects and Bilkent University encourages us to learn and use it, we will utilize it for the rest of the report and the project itself.

1.4 Definitions,Acronyms and Abbreviations



UI - User Interface

DAO - Direct Access Object

UX - User Experience

GDPR - General Data Protection Regulation

JS- JavaScript

SQL - Structured Query Language

JWT - JSON Web Token

REST API - Representational state transfer API

AWS - Amazon Web Services

CRM - Customer Relationship Management

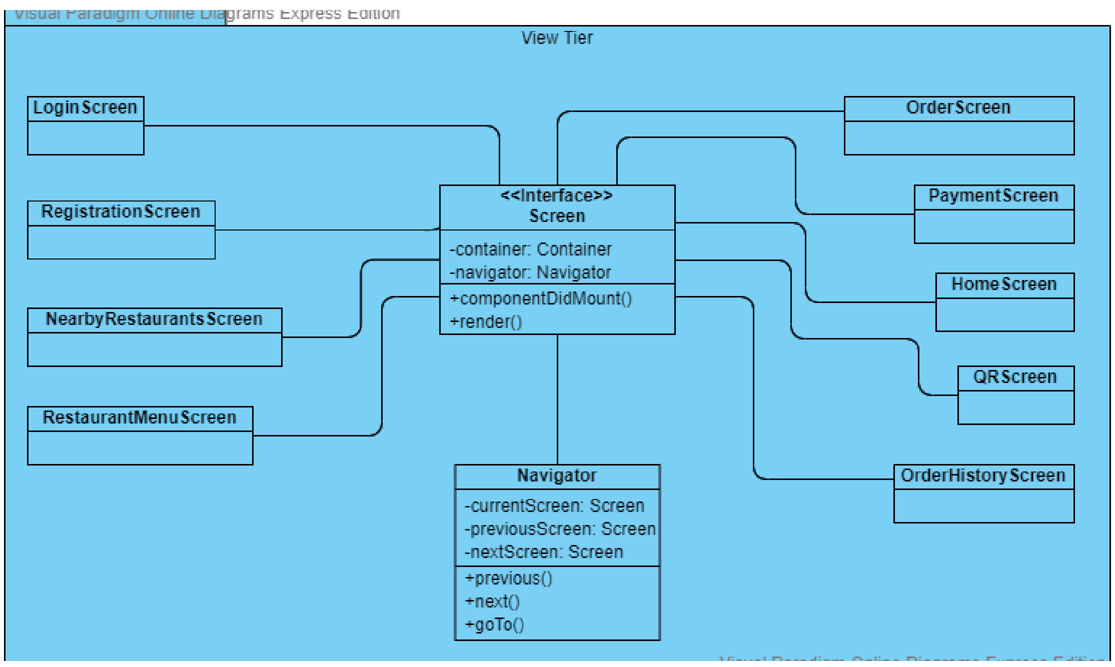
2 Packages

There are 2 subsystems in Qrder project, which are admin,client and server. Admin subsystems manage the operations of the restaurant, Client subsystem creates requests for orders and listens for notifications if there is any ,server subsystems establish connections between these two and do database interactions and transactions. Admin and Client subsystems will be implemented as front-end, therefore MVC pattern is preferred. Therefore, they consist of 3 parts which are model,view and controller tiers. Server subsystem is back-end server for the system,so it should manage the request and database transactions in the back-end. Therefore, 3 layer pattern will be used in this subsystem which are route,logic and data tiers.

2.1 Client

Client subsystem consists of 3 tiers which are model,view and controller. Model tier is to keep required up-to-date information for view tier. View tier makes rendering for the end-user to create the user interface. Controller tier makes modifications on both sides with respect to changes in the application, creates requests to send to the back-end server,dispatches responses and manages modifications according to these responses.

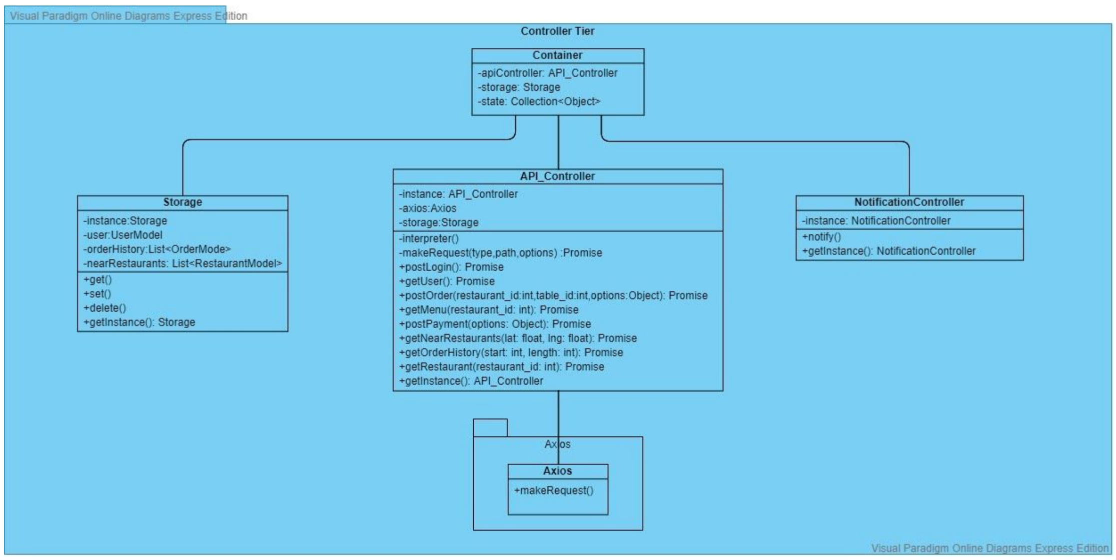
2.1.1 View Tier



View tier is the subsystem for UI components.

Class	**Description**
Screen<<Interface>>	This interface class provides a link between controller tier in order to combine the functionality of the controller tier.
Navigator	This class provides a navigation system to the app. It handles switches between screens.
LoginScreen	This class provides the login components to the user.
RegistrationScreen	This class provides the registration components to the user.
HomeScreen	This class provides special offers to the user and also links to the other pages.
NearbyRestaurantsScreen	This class provides users the restaurants that are in the near, according to the location of users.
OrderHistoryScreen	This class provides users their own order history.
QRScreen	This class provides users to scan QR codes that are provided by restaurants.
RestaurantMenuScreen	This class provides users the menu of the selected restaurant via scanner QR code.
OrderScreen	This class provides users management of their orders.
PaymentScreen	This class provides users to pay the order.

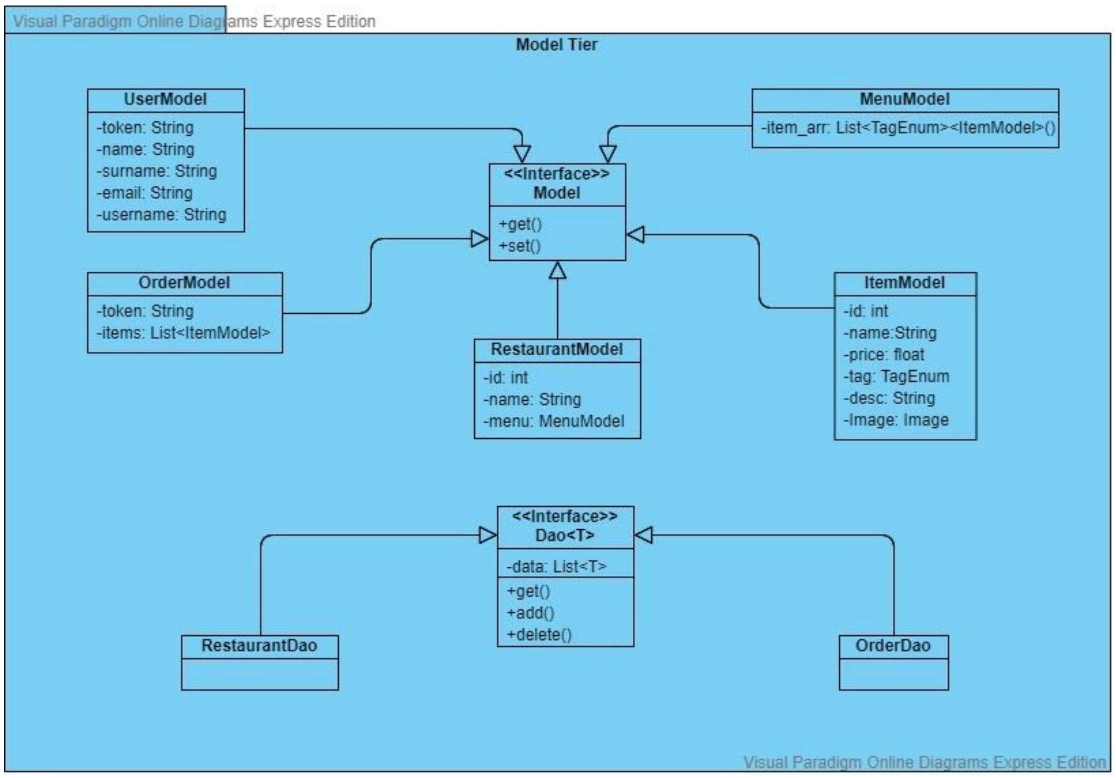
2.1.2 Controller Tier


Visual Paradigm Online Diagrams Express Edition
Controller Tier
Container
-apiController: API_Controller
-storage: Storage
-state: Collection<Object>
Storage
-instance:Storage
-user:UserModel
-orderHistory:List<OrderMode>
-nearRestaurants: List<RestaurantModel>
+get()
+set()
+delete()
+getInstance(): Storage
API_Controller
-instance: API_Controller
-axios:Axios
-storage:Storage
-interpreter()
-makeRequest(type,path,options) :Promise
+postLogin(): Promise
+getUser(): Promise
+postOrder(restaurant_id:int,table_id:int,options:Object): Promise
+getMenu(restaurant_id: int): Promise
+postPayment(options: Object): Promise
+getNearRestaurants(lat: float, lng: float): Promise
+getOrderHistory(start: int, length: int): Promise
+getRestaurant(restaurant_id: int): Promise
+getInstance(): API_Controller
NotificationController
-instance: NotificationController
+notify()
+getInstance(): NotificationController
Axios
Axios
+makeRequest()
Visual Paradigm Online Diagrams Express Edition

Controller tier is the subsystem in order to handle events, requests and responses.

Class	**Description**
Container	This class provides UI classes access to local storage, connection with server, private state and listening notification service.
Storage	This class provides access to the local storage of the device. Any type of data can be saved as an encrypted form.
API_Controller	This class provides sending http requests -get, post, put, delete- to the server via instance of Axios class.
NotificationController	This is a singleton class that handles notifications of the app
Axios	This class belongs to third-party package Axios and is able to send http requests to a specified url and a port [3].

2.1.3 Model Tier



Model tier is the subsystem for Data elements in order to communicate easily and reliably.

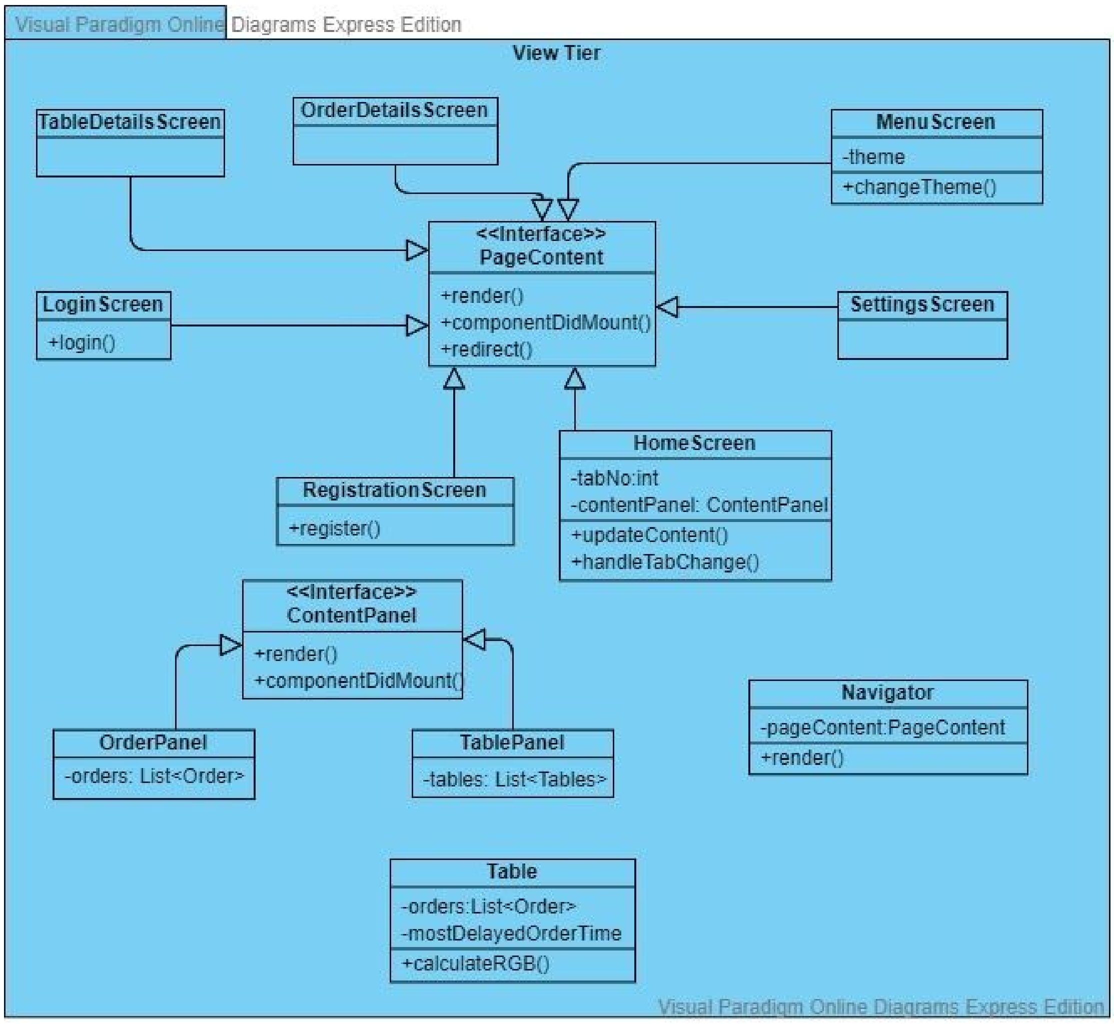
Class	**Description**
Dao <<Interface>>	Dao stands for Data Access Object and isolates the layer of business from relational databases by providing abstract API [4].
RestaurantDao ~ *Dao*	This class provides a Dao for RestaurantModel class.
OrderDao ~ *Dao*	This class provides a Dao for OrderModel class.
Model <<Interface>>	This interface provides get and set methods.

UserModel ~ *Model*	This class provides a model of a User object.
RestaurantModel ~ *Model*	This class provides a model of a Restaurant object.
MenuModel ~ *Model*	This class provides a model of a Menu object.
ItemModel ~ *Model*	This class provides a model of an Item object.
OrderModel ~ *Model*	This class provides a model of an Order object.

2.2 Administrative User

Admin subsystem has the same pattern with the client subsystem, so there are 3 same tiers in this subsystem which are model,view and controller. They have the same purpose inside the system, but functionalities and methods of the classes are different from clients.

2.2.1 View Tier

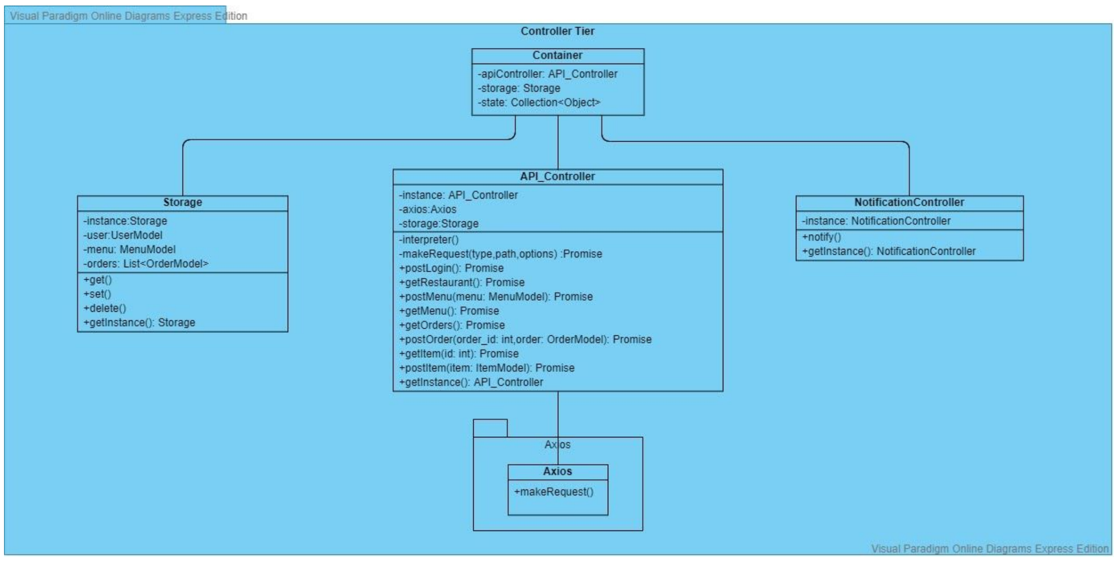


View tier is the subsystem for UI components.

Class	**Description**
Navigator	This class provides a navigation system to the app. It handles switches between screens.
LoginScreen	This class provides the login components to the user.

RegistrationScreen	This class provides the registration components to the user.
HomeScreen	This class provides tabs for the management of the orders and tables.
OrdersDetailsScreen	This class provides management of the orders and details of them.
TableDetailsScreen	This class provides management of the tables and details of them.
MenuScreen	This class provides restaurant owners management of the restaurant menu.
SettingsScreen	This class provides restaurant owners specific settings.

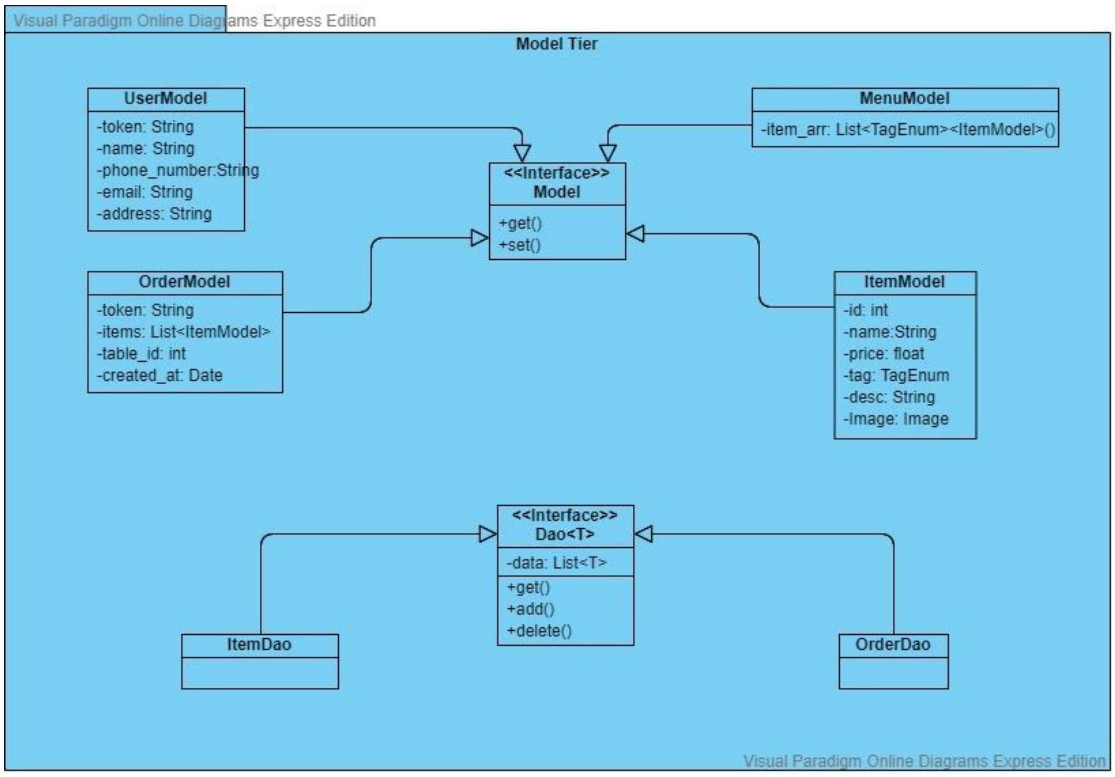
2.2.2 Controller Tier


Visual Paradigm Online Diagrams Express Edition
Controller Tier
Container
-apiController: API_Controller
-storage: Storage
-state: Collection<Object>
Storage
-instance:Storage
-user:UserModel
-menu: MenuModel
-orders: List<OrderModel>
+get()
+set()
+delete()
+getInstance(): Storage
API_Controller
-instance: API_Controller
-axios:Axios
-storage:Storage
-interpreter()
-makeRequest(type,path,options) :Promise
+postLogin(): Promise
+getRestaurant(): Promise
+postMenu(menu: MenuModel): Promise
+getMenu(): Promise
+getOrders(): Promise
+postOrder(order_id: int,order: OrderModel): Promise
+getItem(id: int): Promise
+postItem(item: ItemModel): Promise
+getInstance(): API_Controller
NotificationController
-instance: NotificationController
+notify()
+getInstance(): NotificationController
Axios
Axios
+makeRequest()
Visual Paradigm Online Diagrams Express Edition

Controller tier is the subsystem in order to handle events, requests and responses.

Class	**Description**
Container	This class provides UI classes access to local storage, connection with server, private state and listening notification service.
Storage	This class provides access to the local storage of the device. Any type of data can be saved as an encrypted form.
API_Controller	This class provides sending http requests -get, post, put, delete- to the server via 25 instance of Axios class.
NotificationController	This is a singleton class that handles notifications of the app
Axios	This class belongs to third-party package Axios and is able to send http requests to a specified url and a port [3].

2.2.3 Model Tier



Model tier is the subsystem for Data elements in order to communicate easily and reliably.

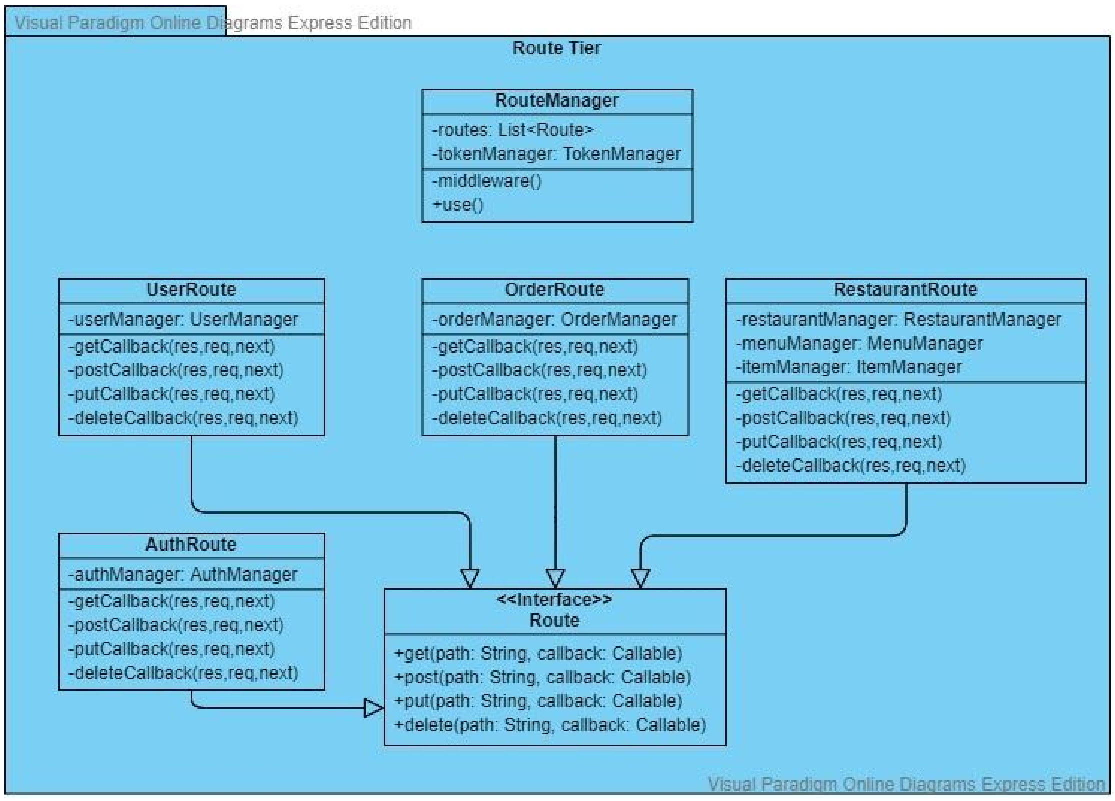
Class	**Description**
Dao <<Interface>>	Dao stands for Data Access Object and isolates the layer of business from relational databases by providing abstract API [4].
ItemDao ~ *Dao*	This class provides a Dao for ItemModel class.
OrderDao ~ *Dao*	This class provides a Dao for OrderModel class.

Model <<Interface>>	This interface provides get and set methods.
UserModel ~ *Model*	This class provides a model of a User object.
MenuModel ~ *Model*	This class provides a model of a Menu object.
ItemModel ~ *Model*	This class provides a model of an Item object.
OrderModel ~ *Model*	This class provides a model of an Order object.

2.3 Server

Server subsystem represents the back-end server in the project. Purpose of this subsystem is to make database transactions, create responses for front-end subsystems and establish data concurrency between subsystems. There are 3 main tiers in this subsystem which are route tier, logic tier and data tier. Route tier manages routing for requests that come from front-end subsystems and parses these requests after routing. After the parsing is done, requests are directed to the logic tier to manage calculations,modifications and dispatching. And finally, required database interactions are done via data tier.

2.3.1 Route Tier



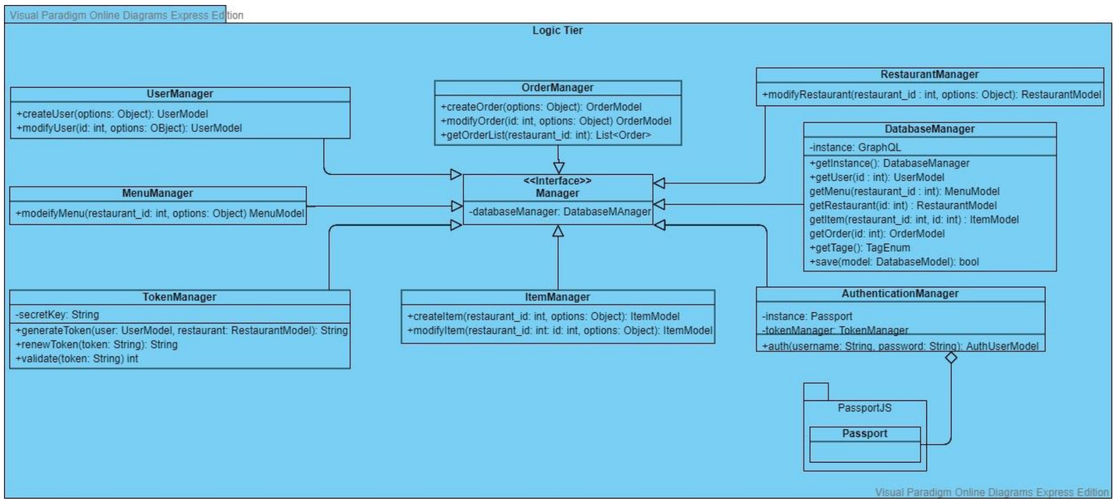
Visual Paradigm Online Diagrams Express Edition
Route Tier
RouteManager
-routes: List<Route>
-tokenManager: TokenManager
-middleware()
+use()
UserRoute
-userManager: UserManager
-getCallback(res,req,next)
-postCallback(res,req,next)
-putCallback(res,req,next)
-deleteCallback(res,req,next)
OrderRoute
-orderManager: OrderManager
-getCallback(res,req,next)
-postCallback(res,req,next)
-putCallback(res,req,next)
-deleteCallback(res,req,next)
RestaurantRoute
-restaurantManager: RestaurantManager
-menuManager: MenuManager
-itemManager: ItemManager
-getCallback(res,req,next)
-postCallback(res,req,next)
-putCallback(res,req,next)
-deleteCallback(res,req,next)
AuthRoute
-authManager: AuthManager
-getCallback(res,req,next)
-postCallback(res,req,next)
-putCallback(res,req,next)
-deleteCallback(res,req,next)
<<Interface>>
Route
+get(path: String, callback: Callable)
+post(path: String, callback: Callable)
+put(path: String, callback: Callable)
+delete(path: String, callback: Callable)
Visual Paradigm Online Diagrams Express Edition

Route tier is the subsystem for Route elements in order to provide the concept of rest api.

Class	**Description**
Route <<Interface>>	This class manages http requests that are get, post, put and delete.
UserRoute ~ *Route*	This class manages http requests to the path is “/user/*”
OrderRoute ~ *Route*	This class manages http requests to the path is “/order/*”
RestaurantRoute ~ *Route*	This class manages http requests to the path is “/restaurant/*”

AuthRoute ~ *Route*	This class manages http requests to the path is "/auth/*"
RouteManager	This class provides server management of the routes.

2.3.2 Logic Tier

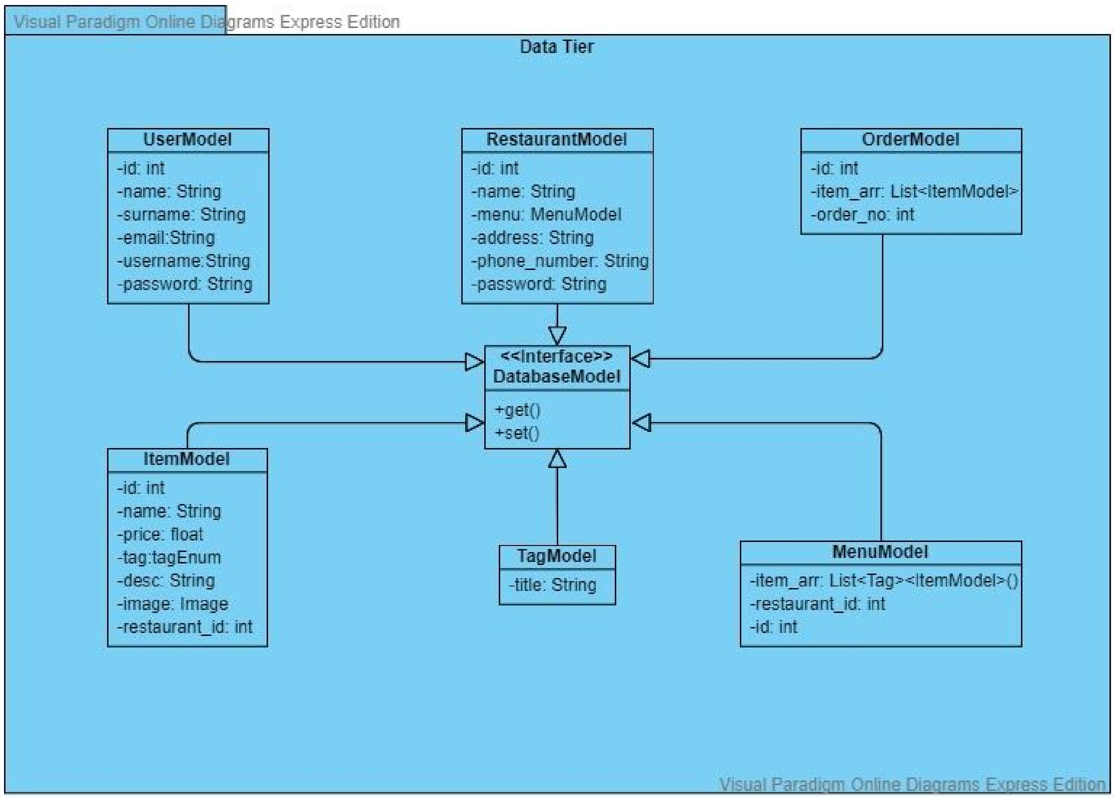

Visual Paradigm Online Diagrams Express Edition
Logic Tier
UserManager
+createUser(options: Object): UserModel
+modifyUser(id: int, options: OBject): UserModel
OrderManager
+createOrder(options: Object): OrderModel
+modifyOrder(id: int, options: Object) OrderModel
+getOrderList(restaurant_id: int): List<Order>
RestaurantManager
+modifyRestaurant(restaurant_id : int, options: Object): RestaurantModel
DatabaseManager
-instance: GraphQL
+getInstance(): DatabaseManager
+getUser(id : int): UserModel
getMenu(restaurant_id : int): MenuModel
getRestaurant(id: int) : RestaurantModel
getItem(restaurant_id: int, id: int) : ItemModel
getOrder(id: int): OrderModel
+getTage(): TagEnum
+save(model: DatabaseModel): bool
<<Interface>>
Manager
-databaseManager: DatabaseMAnager
MenuManager
+modeifyMenu(restaurant_id: int, options: Object) MenuModel
TokenManager
-secretKey: String
+generateToken(user: UserModel, restaurant: RestaurantModel): String
+renewToken(token: String): String
+validate(token: String) int
ItemManager
+createItem(restaurant_id: int, options: Object): ItemModel
+modifyItem(restaurant_id: int: id: int, options: Object): ItemModel
AuthenticationManager
-instance: Passport
-tokenManager: TokenManager
+auth(username: String, password: String): AuthUserModel
PassportJS
Passport
Visual Paradigm Online Diagrams Express Edition

Logic tier is the subsystem for management elements in order to provide the business logic.

Class	**Description**
Manager <<Interface>>	This interface creates a bridge with the DatabaseManager class.
UserManager ~ *Manager*	This class provides the business logic for user operations.
OrderManager ~ *Manager*	This class provides the business logic for order operations.

RestaurantManager ~ *Manager*	This class provides the business logic for restaurant operations.
MenuManager ~ *Manager*	This class provides the business logic for menu operations.
ItemManager ~ *Manager*	This class provides the business logic for item operations.
AuthenticationManager ~ *Manager*	This class provides the business logic for authentication operations.
DatabaseManager	This class provides control of SQL database via GraphQL class.
Passport	This class is provided by third-party package PassportJS and generates middlewares and authentication protocols for the web service by Express [5].

2.3.3 Data Tier



Model tier is the subsystem for Data elements in order to communicate easily and reliably.

Class	**Description**
DatabaseModel <<Interface>>	This interface provides get and set methods.
UserModel ~ *DatabaseModel*	This class provides a model of the User entity in the database.
RestaurantModel ~ *DatabaseModel*	This class provides a model of the Restaurant entity in the database.
MenuModel ~ *DatabaseModel*	This class provides a model of the Menu entity in the database.

ItemModel ~ *DatabaseModel*	This class provides a model of the Item entity in the database.
OrderModel ~ *DatabaseModel*	This class provides a model of the Order entity in the database.
TagModel ~ *DatabaseModel*	This class provides a model of the Tag entity in the database.

3 Class Interfaces

In this section, signatures, properties and methods of the classes will be explained.

3.1 Client

In this section, subsystems that are dedicated to clients will be explained in detail.Since Qrder is not officially finished and the implementation process is still in progress, classes and their properties might be changed.

3.1.1 View Tier

interface Screen	
This class provides access to the controller tier.	
Attributes	
private Container container private Navigator navigator	
Methods	
public void componentDidMount() public void render()	called when screen is mounted by React renders components of screen, create a gui

class Navigator	
This class provides navigation between screens.	
Attributes	
private Screen currentScreen private Screen previousScreen private Screen nextScreen	
Methods	
public void previous() public void next() public void goTo(Screen screen)	go to screen assigned to previousScreen go to screen assigned to nextScreen go to give screen as parameter

class LoginScreen, RegistrationScreen, NearbyRestaurantScreen, RestaurantMenuScreen, OrderScreen, PaymenScreen, HomeScreen, QRScreen, OrderHistoryScreen	
This class is an instance of Screen interface. It provides a gui to users and communicates with the contailler tier.	
Attributes	
-	
Methods	
-	-

3.1.2 Controller Tier

class Storage	
This class provides access to the local storage of the device.	
Attributes	
private Storage instance private UserModel user private List<OrderModel> orderHistory private List<RestaurantModel> nearRestaurants	
Methods	
public String get(String key) public boolean set(String key, String value) public boolean delete(String key) public Storage getInstance()	returns value of the given key sets value for the given key deletes key-value pair in the storage returns the storage

class Container
This class provides UI classes access to local storage, connection with server, private state and listening notification service.
Attributes
private API_Controller apiController private Storage storage private Collection<Object> state

class NotificationController	
This is a singleton class that handles notifications of the app	
Attributes	
private NotificationController	
Methods	
public void notify() public NotificationController getInstance()	send notification returns notification controller

class API_Controller	
This class provides sending http requests -get, post, put, delete- to the server via 25 instance of Axios class	
Attributes	
private API_Controller apiController private Axios axios private Storage storage	
Methods	
public interpreter() public Promise makeRequest(type,path,options) public Promise postLogin() public Promise getUser() public Promise postOrder(int restaurant_id, int table_id, Objects options) public Promise getMenu(int restaurant_id) public Promise postPayment(.object options) public Promise getNearRestaurants(float lat, float lang) public Promise getOrderHistory(int start, int length) public Promise getRestaurant(int restaurant_id) public API_Controller getInstance()	it interpreters result before returning it requesting service from server delivering login post message to server requesting user from server delivering order post message to server getting the menu of the desired restaurant delivering payment post message to server getting restaurants that are close to client getting order history from database getting desired restaurant returns api controller instance

3.1.3 Model Tier

class UserModel
This class is responsible for describing a simple user and it’s basic attributes.
Attributes
private String token private String name private String surname private String email private String username

class OrderModel
This class is responsible for describing a simple order and it’s basic attributes.
Attributes
private String token private List<İtemModel> items

class MenuModel
This class is responsible for describing the menu and it’s attributes.
Attributes
List>TagEnum><ItemModel> item_arr

class ItemModel
This class is responsible for describing simple ştem and it’s attributes.
Attributes
private int id private String name private float price private TagEnum tag private String desc private Image image

class RestaurantModel
This class is responsible for describing a simple restaurant and it’s attributes.
Attributes
private int id private String name private MenuModel menu

interface Model	
This interface holds the definiton of the get and set methods for the UserModel, OrderModel, MenuModel, ItemModel and RestaurantModel.	
Methods	
public int get() public void set()	gets desired attribute sets the new value to desired attribute

interface Dao<T>	
This interface holds the definition of the objects that are fetched from the database.	
Attributes	
private List<T> data	
Methods	
public JSON get() public void add() public void delete()	returns object in JSON format add object to data attribute deletes object from data attribute

class RestaurantDao
This class is responsible for holding the desired restaurant that is fetched from database.

class OrderDao
This class is responsible for holding the desired order that is fetched from database.

3.2 Administrative User

In this section, subsystems that are dedicated to administative users will be explained in detail.Since Qrder is not officially finished and implementation process is still on progress, classes and their properties might be changed.

3.2.1 View Tier

interface PageContent	
This class provides access to the controller tier.	
Attributes	
-	
Methods	
public void componentDidMount() public void render() public void redirect(path)	called when screen is mounted by React renders components of screen, create a gui redirects user to another path

interface ContentPanel	
This class provides gui layout for various aims.	
Attributes	
-	
Methods	
public void componentDidMount() public void render()	called when screen is mounted by React renders components of screen, create a gui

class Navigator	
This class provides access to the controller tier.	
Attributes	
private PageContent pageContent	
Methods	
public render()	renders components of screen, create a gui

class Table	
Class for rendering individual table components. Table components have shifting color to demonstrate the elapsed time after order was taken.	
Attributes	
private List<Order> orders private Time mostDelayedOrderTime	
Methods	
public void calculateRGB()	calculate RGB values of the table component respect to elapsed time after order was taken

class LoginScreen, RegistrationScreen, TableDetailsScreen, OrderDetailsScreen, MenuScreen, SettingsScreen, HomeScreen	
This class is an instance of the PageContent interface. It provides a gui to users and communicates with the containtier tier.	
Attributes	
private String theme* private Int tableNo* private ContentPanel contentPanel*	
Methods	
public boolean login()* public boolean changeTheme()* public void updateContent()* public void handleTabChange()*	uses controller tier and logins changes theme of the app update contents of the screens after any action taken in view change main content components between order panel and table panel
*This methods and attributes are unique for PageContent instances	

class OrderPanel	
Main component to show orders in a list under orders tab.	
Attributes	
private List<Order> orders	
Methods	
-	-

class TablePanel	
Main component to show tables in a grid view under tables tab.	
Attributes	
private List<Table> tables	
Methods	
-	-

3.2.2 Controller Tier

class Storage	
This class provides access to the local storage of the device.	
Attributes	
private Storage instance private UserModel user private List<MenuModel> menu private List<OrderModel> orders	
Methods	
public JSON get() public void set() public void delete() public Storage getInstance()	returns object in JSON format add object to data attribute deletes object from data attribute returns the storage object

class API_Controller	
This class provides sending http requests -get, post, put, delete- to the server via 25 instance of Axios class	
Attributes	
private API_Controller apiController private Storage storage private Collection<Object> state	
Methods	
public interpreter()	it interpreters result before returning it
public Promise makeRequest(type,path,options)	make a request from server
public Promise postLogin()	delivering login message to the server
public Promise getRestaurant()	requesting restaurant from the server
public Promise postMenu()	delivering menu message to the server
public Promise getOrders()	fetches orders from the server
public Promise postOrder(int oder_id, OrderModel order)	delivering order message to the server
public Promise getItem(int id)	gets desired Item
public Promise postItem(ItemModel item)	delivers post item to the server
public API_Controller getInstance()	returns api controller

class Container
This class provides UI classes access to local storage, connection with server, private state and listening notification service.
Attributes
private API_Controller apiController private Storage storage private Collection<Object> state

class NotificationController	
This is a singleton class that handles notifications of the app	
Attributes	
private NotificationController	
Methods	
public void notify()	send notification
public NotificationController getInstance()	returns notification controller

3.2.3 Model Tier

class UserModel
This class is responsible for describing a simple user and it’s basic attributes.
Attributes
private String token private String name private String surname private String email private String username

class OrderModel
This class is responsible for describing a simple order and it’s basic attributes.
Attributes
private String token private List<İtemModel> items private int table_int private Date created_at

class MenuModel
This class is responsible for describing the menu and it’s attributes.
Attributes
List>TagEnum><ItemModel> item_arr

class ItemModel
This class is responsible for describing simple ştem and it’s attributes.
Attributes
private int id private String name private float price private TagEnum tag private String desc private Image image

interface Model	
This interface holds the definiton of the get and set methods for the UserModel, OrderModel, MenuModel, ItemModel and RestaurantModel.	
Methods	
public int get() public void set()	gets desired attribute sets the new value to desired attribute

interface Dao<T>	
This interface holds the definition of the objects that are fetched from the database.	
Attributes	
private List<T> data	
Methods	
public JSON get() public void add() public void delete()	returns object in JSON format add object to data attribute deletes object from data attribute

class ItemDao
This class is responsible for holding the desired item that is fetched from database.

class OrderDao
This class is responsible for holding the desired order that is fetched from database.

3.3 Server

In this section, subsystems that are dedicated to servers will be explained in detail.Since Qrder is not officially finished and the implementation process is still on progress, classes and their properties might be changed.

3.3.1 Route Tier

class UserRoute	
This class manages http requests to the path is "/user/*"	
Attributes	
private UserManage userManager	
Methods	
public JSON getCallback(res,req,next) public JSON postCallback(res,req, next) public JSON putCallback(res,req,next) public JSON deleteCallback(res,req,next)	callback function for get request callback function for post request callback function for put request callback function for delete request

class AuthRoute	
This class manages http requests to the path is "/auth/*"	
Attributes	
private AuthManager authManager	
Methods	
public JSON getCallback(res,req,next) public JSON postCallback(res,req, next) public JSON putCallback(res,req,next) public JSON deleteCallback(res,req,next)	callback function for get request callback function for post request callback function for put request callback function for delete request

class OrderRoute	
This class manages http requests to the path is "/order/*"	
Attributes	
private OrderManager orderManager	
Methods	
public JSON getCallback(res,req,next) public JSON postCallback(res,req, next) public JSON putCallback(res,req,next) public JSON deleteCallback(res,req,next)	callback function for get request callback function for post request callback function for put request callback function for delete request

class RestaurantRoute	
This class manages http requests to the path is “/restaurant//*”	
Attributes	
private RestaurantManager restaurantManager private MenuManager menuManager private ItemManager itemManager	
Methods	
public JSON getCallback(res,req,next) public JSON postCallback(res,req, next) public JSON putCallback(res,req,next) public JSON deleteCallback(res,req,next)	callback function for get request callback function for post request callback function for put request callback function for delete request

<<interface>> Route	
This class manages http requests to the path is “/restaurant//*”	
Methods	
public get (String path, Callable callback) public post(String path, callable callback) private put(String path, Callable callback) private delete(String path, Callable callback)	callback function for get request of path callback function for post request of path callback function for put request of path callback function for delete request of path

3.3.2 Logic Tier

class UserManager	
This class provides the business logic for user operations.	
Methods	
public UserModel createuser(Object options) public UserModel modifyUser(int id, Object options)	creates user modifies user

class MenuManager	
This class provides the business logic for menu operations.	
Methods	
public MenuModel modifyMenu(int restaurant_id, Object options)	modifies menu

class TokenManager	
This class provides the business logic for token operations.	
Attributes	
private String secretKey	
Methods	
public String generateToken(UserModel user, RestaurantModel restaurantt) public String renewToken(String token) public int validate(String token)	generates token renews token checks if the token is valid

class OrderManager	
This class provides the business logic for order operations.	
Methods	
public OrderModel createOrder(Object options) public OrderModel modifyOrder(int id, Object option) public List<Order> getOrderList(int restaurant_id)	creates order modifies order gets the order list

class RestaurantManager	
This class provides the business logic for restaurant operations.	
Methods	
public RestaurantModel modifyRestaurant(int restaurant_id, Object options)	modifies restaurant

class AuthenticationManager	
This class provides the business logic for authentication operations.	
Attributes	
private Passport instance private TokenManager tokenManager	
Methods	
public AuthUserModel auth(String username, string pwd)	auths user

class DatabaseManager	
This class provides the business logic for database operations.	
Attributes	
private GraphQL instance	
Methods	
public DatabaseManager getInstance() public UserModel getUser(int id) public MenuModel getMenu(int restaurant_id) public RestaurantModel getRestaurant(int id) public ItemModel getItem(int res_id, int id) public OrderModel getOrder(int id) public TagEnum getTage() public bool save(DatabaseModel model)	gets instance of database model gets user gets menu gets restaurant gets item gets order gets tag saves current state

3.3.3 Data Tier

class UserModel
This class is responsible for describing a simple user and it’s basic attributes.
Attributes
private int id private String name private String surname private String email private String username private String password

class ItemModel
This class is responsible for describing simple ştem and it’s attributes.
Attributes
private int id private String name private float price private TagEnum tag private String desc private Image image private int restaurant_id

class RestaurantModel
This class is responsible for describing a simple restaurant and it’s basic attributes.
Attributes
private int id private String name Private MenuModel menu private String address private String phone_number private String password

class OrderModel
This class is responsible for describing a simple restaurant and it’s basic attributes.
Attributes
private int id private List<ItemModel> item_arr private int order_no

class MenuModel
This class is responsible for describing a simple menu and it’s basic attributes.
Attributes
private int id private List<ItemModel> item_arr private int restaurant_id

class TagModel
This class is responsible for describing a simple tag and it’s basic attributes.
Attributes
private String title

interface DatabaseModel	
This interface holds the definiton of the get and set methods for the UserModel, OrderModel, MenuModel, ItemModel and RestaurantModel.	
Methods	
public int get() public void set()	gets desired attribute sets the new value to desired attribute

4 Glossary

AWS - Amazon Web Services

REST	Representational State Transfer
HTTP	Hyper-Text Transfer Protocol
JS	JavaScript
QR Code	Quick response code
GDPR	General Data Privacy Regulation
ORM	Object-relational mapping

5 References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] "Service Quality In Restaurants" *Service Quality In Restaurants*, https://www.ukessays.com/essays/marketing/service-quality-in-restaurants-marketing-essay.php/. [Accessed: 21- May-2020].

[3] Axios. "Axios/Axios." GitHub, 7 Mar. 2020, github.com/axios/axios

[4] Baeldung. "The DAO Pattern in Java." Baeldung, 21 Mar. 2020, www.baeldung.com/java-dao-pattern.

[5] "Passport.js." Passport.js, www.passportjs.org/.

[6] "MQTT: The Standard for IoT Messaging" MQTT https://mqtt.org/ [Accessed: 04-Oct-2020]

[7] "What Is Uml." *What Is Unified Modeling Language (UML)?*, www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 04- Oct-2020]