Documentation of a project for the purpose of the course BIE-SI1.

## Contents

# 1. Architecture

The logical architecture describes the principles of implementation of the application. It identifies the basic components of the application and the way how these components are connected together.
This project contains a common used three-layered architecture very popular for separating the presentation logic, the business logic and the data logic.
This chapter describes the architecture of the web application of the Restaurant Information System.

The web application for public information about books and their authors will be implemented as a web application based on the following technologies:

- ASP.NET framework
- C# 6.0
- csHTML

The architecture is divided into three independent layers:

- Presentation Layer - layer responsible for presentation of application data. For this particular project it is only a visual representation for the packages containing the home controller and the views because in reality it doesn't exist
- Business Layer- layer responsible for all business logic of the application
- Data Layer - layer responsible for data persistence

## 1.1 Architecture

The layers are mainly packages containing related classes in terms of functionalities.

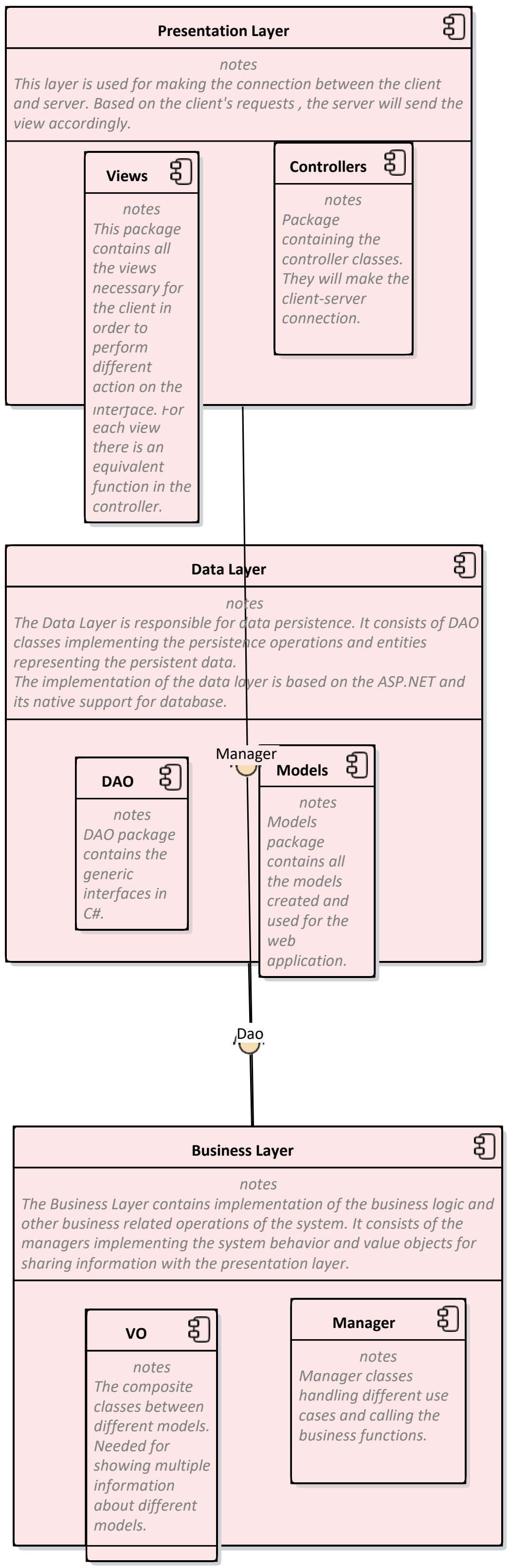

Presentation Layer
notes
This layer is used for making the connection between the client and server. Based on the client's requests , the server will send the view accordingly.
Views
notes
This package contains all the views necessary for the client in order to perform different action on the interface. For each view there is an equivalent function in the controller.
Controllers
notes
Package containing the controller classes. They will make the client-server connection.
Data Layer
notes
The Data Layer is responsible for data persistence. It consists of DAO classes implementing the persistence operations and entities representing the persistent data.
The implementation of the data layer is based on the ASP.NET and its native support for database.
Manager
DAO
notes
DAO package contains the generic interfaces in C#.
Models
notes
Models package contains all the models created and used for the web application.
Dao
Business Layer
notes
The Business Layer contains implementation of the business logic and other business related operations of the system. It consists of the managers implementing the system behavior and value objects for sharing information with the presentation layer.
VO
notes
The composite classes between different models. Needed for showing multiple information about different models.
Manager
notes
Manager classes handling different use cases and calling the business functions.

Figure 1 - Architecture

### 1.1.1 Business Layer

The Business Layer contains implementation of the business logic and other business related operations of the system. It consists of the managers implementing the system behavior and value objects for sharing information with the presentation layer.

#### 1.1.1.1 Manager

Manager classes handling different use cases and calling the business functions.

#### 1.1.1.2 VO

The composite classes between different models. Needed for showing multiple information about different models.

### 1.1.2 Data Layer

The Data Layer is responsible for data persistence. It consists of DAO classes implementing the persistence operations and entities representing the persistent data.
The implementation of the data layer is based on the ASP.NET and its native support for database.

#### 1.1.2.1 DAO

DAO package contains the generic interfaces in C#.

#### 1.1.2.2 Models

Models package contains all the models created and used for the web application.

### 1.1.3 Presentation Layer

This layer is used for making the connection between the client and server. Based on the client's requests , the server will send the view accordingly.

#### 1.1.3.1 Controllers

Package containing the controller classes. They will make the client-server connection.

#### 1.1.3.2 Views

This package contains all the views necessary for the client in order to perform different action on the interface. For each view there is an equivalent function in the controller.

# 2. Design Class Model

The design class model describes the structure of the implementation. It defines the individual implementation classes, their methods and properties, interfaces and relations.

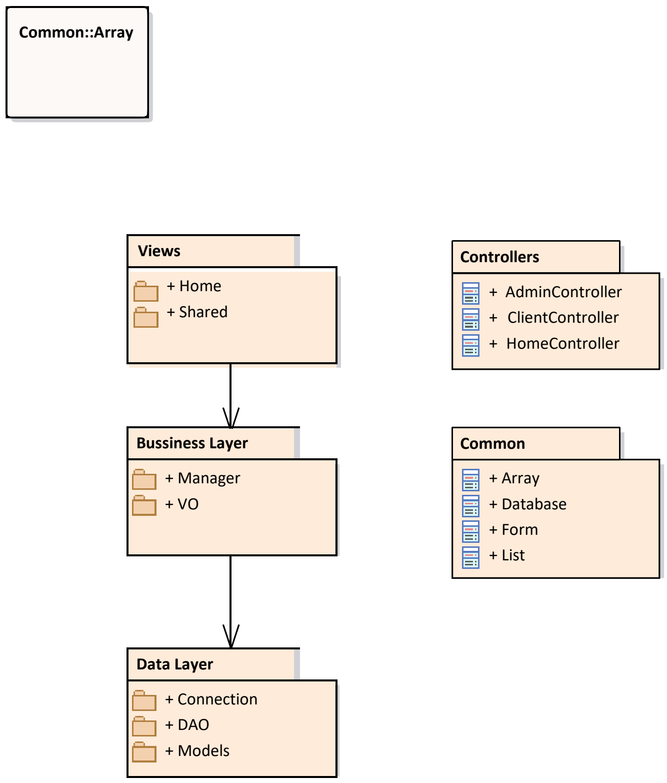


Figure 2 - Design Class Model

These classes and interfaces are divided into various packages forming the individual components and namespaces.

## 2.1 Bussiness Layer

This package contains the classes of the entities, defining the data objects synchronized with a persistent storage.

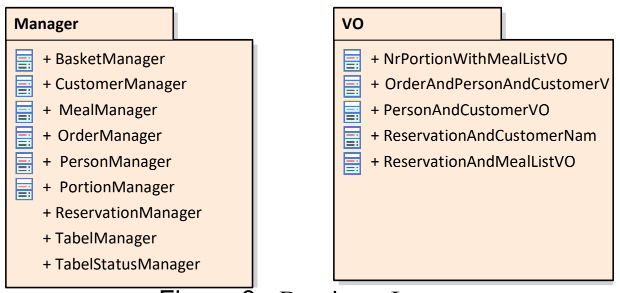


Figure 3 - Bussiness Layer

### 2.1.1 Manager

Contains the classes handling the business processes and errors.

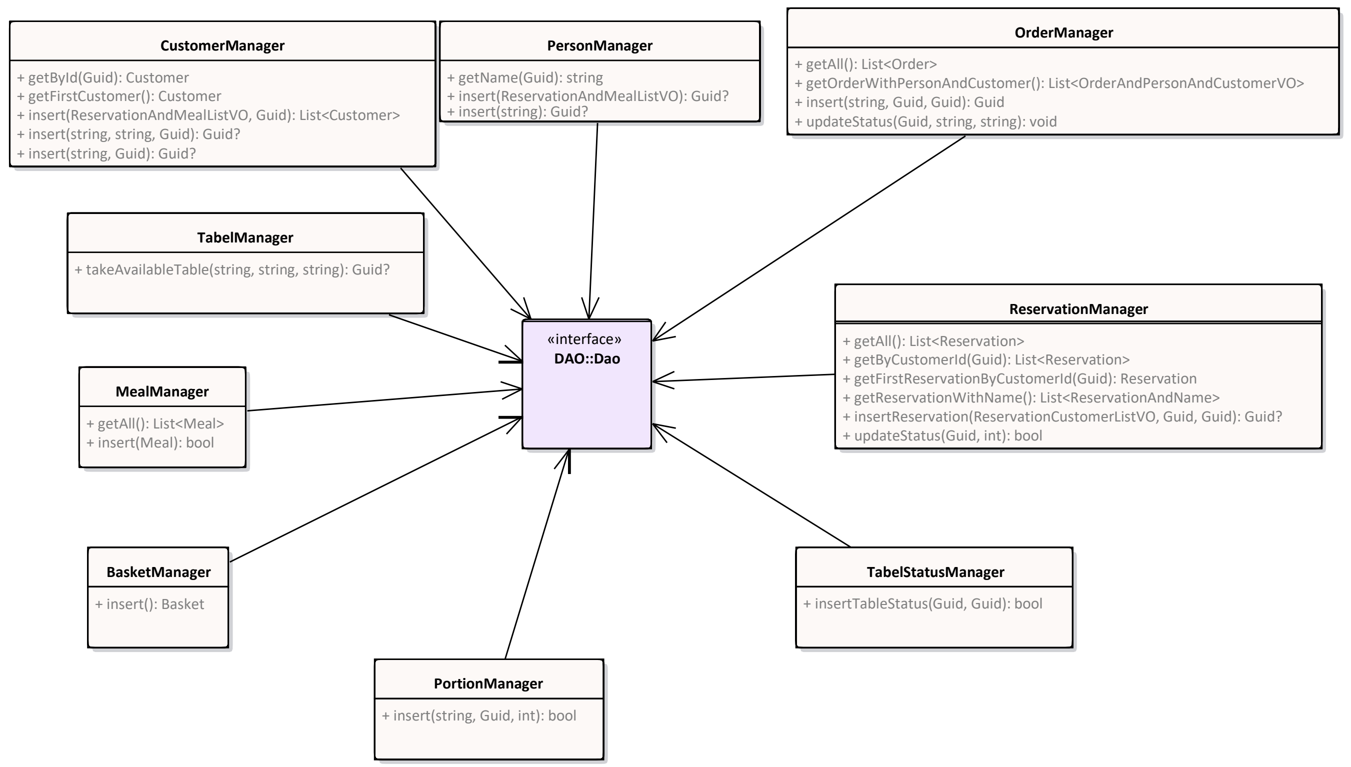

CustomerManager
+ getById(Guid): Customer
+ getFirstCustomer(): Customer
+ insert(ReservationAndMealListVO, Guid): List<Customer>
+ insert(string, string, Guid): Guid?
+ insert(string, Guid): Guid?
PersonManager
+ getName(Guid): string
+ insert(ReservationAndMealListVO): Guid?
+ insert(string): Guid?
OrderManager
+ getAll(): List<Order>
+ getOrderWithPersonAndCustomer(): List<OrderAndPersonAndCustomerVO>
+ insert(string, Guid, Guid): Guid
+ updateStatus(Guid, string, string): void
TabelManager
+ takeAvailableTable(string, string, string): Guid?
«interface»
DAO::Dao
ReservationManager
+ getAll(): List<Reservation>
+ getByCustomerId(Guid): List<Reservation>
+ getFirstReservationByCustomerId(Guid): Reservation
+ getReservationWithName(): List<ReservationAndName>
+ insertReservation(ReservationCustomerListVO, Guid, Guid): Guid?
+ updateStatus(Guid, int): bool
MealManager
+ getAll(): List<Meal>
+ insert(Meal): bool
BasketManager
+ insert(): Basket
TabelStatusManager
+ insertTableStatus(Guid, Guid): bool
PortionManager
+ insert(string, Guid, int): bool

Figure 4 - Manager

#### 2.1.1.1 BasketManager

Manager used for handling the operation of inserting into a basket and catching the errors.

| Method name | Return type | Description |
|---|---|---|
| insert | Basket | Method for inserting an object of type basket into the database. |

#### 2.1.1.2 CustomerManager

Manager used for retrieving a customer by it's name or id , or for inserting a customer information (encapsulated in different VO objects).

| Method name | Return type | Description |
|---|---|---|
| getById | Customer | Method for searching into the database a customer by its ID.<br>Parameters:<br>**id: Guid -** |
| getFirstCustomer | Customer | Method for retrieving the first customer from database and creating an object. |

| Method name | Return type | Description |
|---|---|---|
| | | |
| insert | List<Customer> | Method for inserting a customer into the database. It takes the necessary information from the VO object passed as an argument and also it requires a person ID .<br>Parameters:<br>**model: ReservationAndMealListVO** -<br>Parameters:<br>**personID: Guid** - |
| insert | Guid? | Method for inserting into the database a customer based on his email ,address and the person ID.<br>Parameters:<br>**address: string** -<br>Parameters:<br>**email: string** -<br>Parameters:<br>**personID: Guid** - |
| insert | Guid? | Inserts a customer into the database having as arguments his email and the ID of the person created into the database.<br>Parameters:<br>**model: string** -<br>Parameters:<br>**personId: Guid** - |

#### 2.1.1.3 MealManager

Manager for getting the list of all meals and inserting different ones into the database.

| Method name | Return type | Description |
|---|---|---|
| getAll | List<Meal> | Method for retrieving all the meals from the database. |
| insert | bool | Inserts into the database a meal object.<br>Parameters:<br>**model: Meal** - |

#### 2.1.1.4 OrderManager

Manager for making sure that deliveries are inserted and handled correctly.

| Method name | Return type | Description |
|---|---|---|
| getAll | List<Order> | Method for retrieving all the orders from the data layer. |
| getOrderWithPersonAndCustomer | List<OrderAndPersonAndCustomerVO> | Method for retrieving a list of custom objects having also information about the client . |
| insert | Guid | Method for inserting an order object passing as arguments the phone number for the delivery , the id of customer with his necessary information and the id of the basket with the client's |

| Method name | Return type | Description |
|---|---|---|
| | | meals.<br>Parameters:<br>**phone: string** -<br>Parameters:<br>**basketID: Guid** -<br>Parameters:<br>**customerID: Guid** - |
| updateStatus | void | Method for updating the status of an order. The status can be rejected,impeding or accepted( -1 , 0 or 1).<br>Parameters:<br>**id: Guid** -<br>Parameters:<br>**prTime: string** -<br>Parameters:<br>**status: string** - |

### 2.1.1.5 PersonManager

Manager for handling the person objects. This class will make sure to retrieve or insert the person .

| Method name | Return type | Description |
|---|---|---|
| getName | string | Gets the name of a person object based on his id .<br>Parameters:<br>**id: Guid** - |
| insert | Guid? | Method for inserting a person into the database based on the information passed by the model given as an argument to the function.<br>Parameters:<br>**model: ReservationAndMealListVO** - |
| insert | Guid? | Method for inserting a person having the name passed as an argument.<br>Parameters:<br>**name: string** - |

### 2.1.1.6 PortionManager

Manager for

| Method name | Return type | Description |
|---|---|---|
| insert | bool | Method for inserting a portion object into the database .<br>Parameters:<br>**mealID: string** -<br>Parameters:<br>**basketID: Guid** -<br>Parameters:<br>**quantity: int** - |

| Method name | Return type | Description |
|---|---|---|
| | | |

### 2.1.1.7 ReservationManager

Manager for handling the reservation objects such as checking the status, inserting correctly an object or selecting reservation based on different criteria.

| Method name | Return type | Description |
|---|---|---|
| getAll | List<Reservation> | Retrieves a list of all reservation from the database. |
| getByCustomerId | List<Reservation> | Gets a specific reservation for a customer based on the customer ID.<br>Parameters:<br>**customerID: Guid** - |
| getFirstReservationByCustomerId | Reservation | Method for retrieving only one reservation based on the customer ID.<br>Parameters:<br>**id: Guid** - |
| getReservationWithName | List<ReservationAndName> | Retrieves the list of custom VO objects having information about the reservation and some information about the customer. |
| insertReservation | Guid? | Method for inserting a reservation object with the information given from the model passed as a parameter,the basket id for the list of meals and the customer ID of this reservation.<br>Parameters:<br>**model: ReservationCustomerListVO** -<br>Parameters:<br>**customerID: Guid** -<br>Parameters:<br>**basketID: Guid** - |
| updateStatus | bool | Updates the status of the reservation having as an argument the ID reservation, and the status as an integer with the values -1 for rejected, 0 for impeding and 1 for accepted.<br>Parameters:<br>**reservationID: Guid** -<br>Parameters:<br>**status: int** - |

### 2.1.1.8 TabelManager

Manager for a table, it will retrieve a table from the database.

| Method name | Return type | Description |
|---|---|---|
| takeAvailableTable | Guid? | Method for taking an available table with the condition it available on the date and time of reservation and it can keep all of the persons passed as an argument. |

| Method name | Return type | Description |
|---|---|---|
| | | Parameters:<br>**date: string** -<br>Parameters:<br>**time: string** -<br>Parameters:<br>**nrPersons: string** - |

#### 2.1.1.9 TabelStatusManager

Manager of table status, auxiliary table between reservations and tables.

| Method name | Return type | Description |
|---|---|---|
| insertTableStatus | bool | Method for inserting a mapping between the reservation and the table into the table status .<br>Parameters:<br>**reservationID: Guid** -<br>Parameters:<br>**tableID: Guid** - |

### 2.1.2 VO

Contains composite models used for parsing data from viewer or for sending data to the viewer.

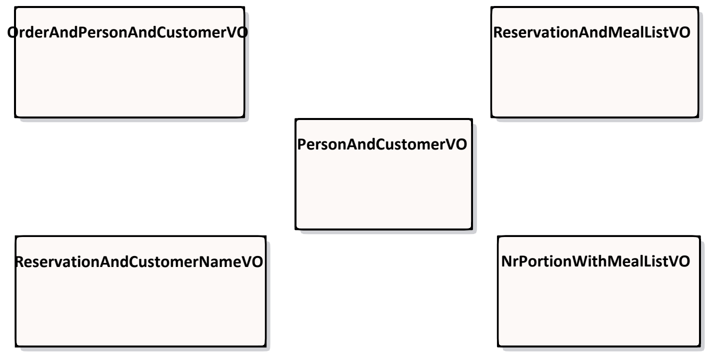


Figure 5 - VO

#### 2.1.2.1 NrPortionWithMealListVO

VO object containing the list of meals to be shown and an argument for the number of portions chosen for a meal.

#### 2.1.2.2 OrderAndPersonAndCustomerVO

Object used for showing the necessary information about the delivery made by an user , such as his name, email, address , the status of the delivery and the preparation time.

#### 2.1.2.3 PersonAndCustomerVO

Object used for keeping the necessary information for both a customer and a person.

#### 2.1.2.4 ReservationAndCustomerNameVO

VO object which contains the details about a customer and his reservation for a table in the restaurant. It will keep in touch with the status of reservation ,the number of persons ,the name of the customer who created the reservation and so on.

#### 2.1.2.5 ReservationAndMealListVO

Contains a list of meals to be displayed and some fields for keeping the information about the customer. This object will be used for the view of creating a reservation( having on the same view both the form for completing the information about the client and also the list of meals from which to choose).

## 2.2 Common

This package describes the classes and packages of the implementation of the web application based on the .NET Framework.
In the whole model, only important classes and their methods are shown. The primitive getters and setters are not shown.

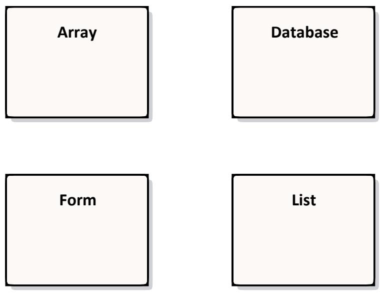


Figure 6 - Common

### 2.2.1 Array

A generic array object provided by the C# itself.

### 2.2.2 Database

A class from the .NET Framework representing the general connection to the database. It offers operations for queries and other operations in the database.

### 2.2.3 Form

A generic .NET class representing a form on a page.

### 2.2.4 List

A generic list object provided by the C# itself.

## 2.3 Controllers

A controllesr is responsible for the initial processing of the request and instantiation of the model.
The methods of the package controller takes the result of the model's processing (if any) and returns either the proper view and its associated view data or the result of the API call.

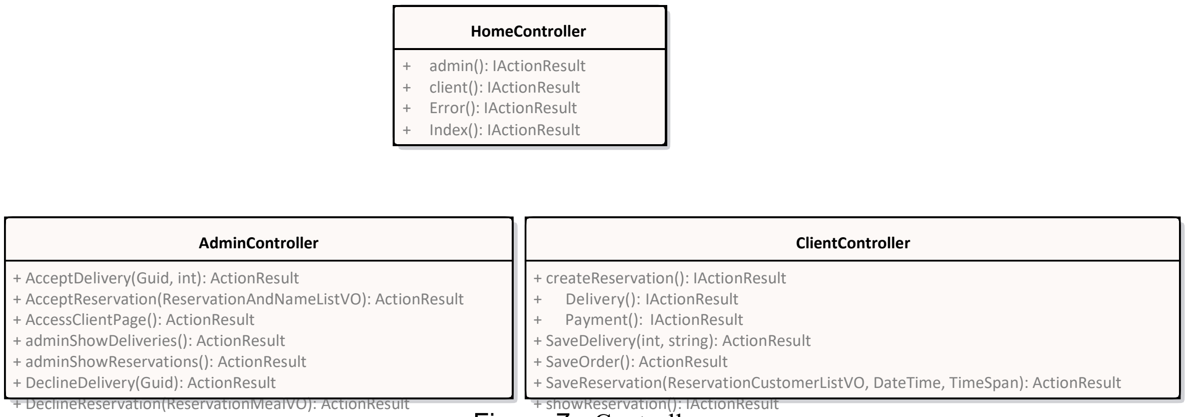

HomeController
+ admin(): IActionResult
+ client(): IActionResult
+ Error(): IActionResult
+ Index(): IActionResult
AdminController
+ AcceptDelivery(Guid, int): ActionResult
+ AcceptReservation(ReservationAndNameListVO): ActionResult
+ AccessClientPage(): ActionResult
+ adminShowDeliveries(): ActionResult
+ adminShowReservations(): ActionResult
+ DeclineDelivery(Guid): ActionResult
+ DeclineReservation(ReservationMealVO): ActionResult
ClientController
+ createReservation(): IActionResult
+ Delivery(): IActionResult
+ Payment(): IActionResult
+ SaveDelivery(int, string): ActionResult
+ SaveOrder(): ActionResult
+ SaveReservation(ReservationCustomerListVO, DateTime, TimeSpan): ActionResult
+ showReservation(): IActionResult

Figure 7 - Controllers

### 2.3.1 HomeController

This class is handles the data received from the client and also the data retrieved from the data base and makes the connection between the server and the client with the business functions.

| Method name | Return type | Description |
|---|---|---|
| admin | IActionResult | The first page of the admin have the functionalities of the administrator. |
| client | IActionResult | The first page of the client with the method of the client. |
| Error | IActionResult | |
| Index | IActionResult | |

## 2.4 Data Layer

This package contains the classes and packages of the data layer.

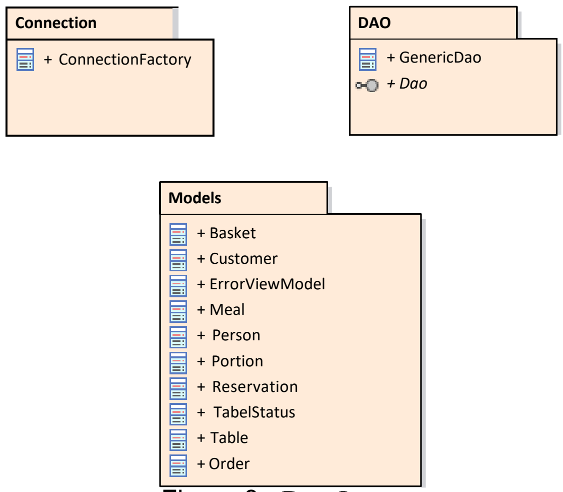


Figure 8 - Data Layer

### 2.4.1 Connection

Connection package is equal to how to work with the data in a database and the first obvious step is the connection. SqlCommand in C# allow the user to query and send the commands to the database. SQL command is specified by the SQL connection object.

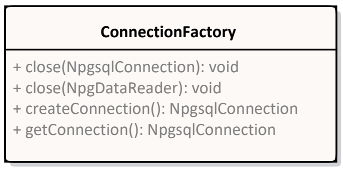

ConnectionFactory
+ close(NpgsqlConnection): void
+ close(NpgDataReader): void
+ createConnection(): NpgsqlConnection
+ getConnection(): NpgsqlConnection

Figure 9 - Connection

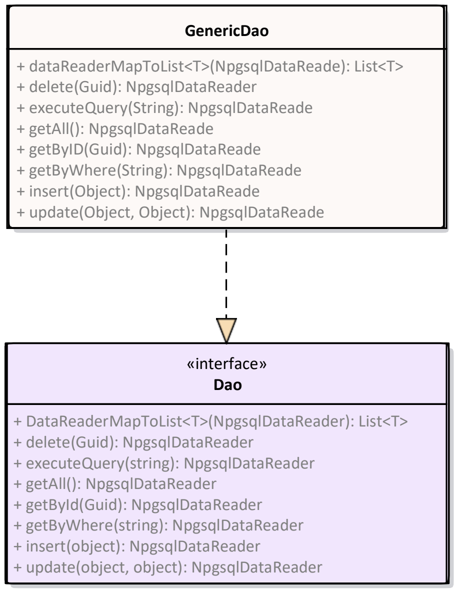
#### 2.4.1.1 ConnectionFactory

This class make the connection between the data base and the server.

| Method name | Return type | Description |
|---|---|---|
| close | void | This Methos is closing the connection.<br>Parameters:<br>**connection: NpgsqlConnection -** |
| close | void | This method is closing a string received from postgresql.<br>Parameters:<br>**reader: NpgDataReader -** |
| createConnection | NpgsqlConnection | This method is creating the connection between the database and the server. |
| getConnection | NpgsqlConnection | This method is getting the instance of a connection. |

### 2.4.2 DAO

DAO represents the classes accessing the database using different queries. It is divided by the business layer using an interface and an abstract generic class.


GenericDao
+ dataReaderMapToList<T>(NpgsqlDataReade): List<T>
+ delete(Guid): NpgsqlDataReader
+ executeQuery(String): NpgsqlDataReade
+ getAll(): NpgsqlDataReade
+ getByID(Guid): NpgsqlDataReade
+ getByWhere(String): NpgsqlDataReade
+ insert(Object): NpgsqlDataReade
+ update(Object, Object): NpgsqlDataReade
«interface»
Dao
+ DataReaderMapToList<T>(NpgsqlDataReader): List<T>
+ delete(Guid): NpgsqlDataReader
+ executeQuery(string): NpgsqlDataReader
+ getAll(): NpgsqlDataReader
+ getById(Guid): NpgsqlDataReader
+ getByWhere(string): NpgsqlDataReader
+ insert(object): NpgsqlDataReader
+ update(object, object): NpgsqlDataReader
Figure 10 - DAO

#### 2.4.2.1 GenericDao

This is the class that constructs the queries and retrieves data for making the operations (inserting,deleting,updating, selecting) on the database.

| Method name | Return type | Description |
|---|---|---|
| dataReaderMapToList<T> | List<T> | This method is mapping a data string to a list of objects of T type.<br>Parameters:<br>**dr: NpgsqlDataReade** - |
| delete | NpgsqlDataReader | This method is deleting a object from the data base, using the id passed as an argument.<br>Parameters:<br>**id: Guid** - |
| executeQuery | NpgsqlDataReade | This method executes a query received as an argument.<br>Parameters:<br>**query: String** - |
| getAll | NpgsqlDataReade | This method makes a selection of some T type objects. |
| getByID | NpgsqlDataReade | This method makes a selection over the T object and it gives as an parameter the ID. |

| Method name | Return type | Description |
|---|---|---|
| | | Parameters:<br>**id: Guid** - |
| getByWhere | NpgsqlDataReade | This method makes a selection over the T entity having a where condition which receives a query as parameter.<br>Parameters:<br>**param: String** - |
| insert | NpgsqlDataReade | This method inserts into the T object the source s given as a parameter.<br>Parameters:<br>**src: Object** - |
| update | NpgsqlDataReade | This method makes an update over the src object and updates it with the dst object.<br>Parameters:<br>**src: Object** -<br>Parameters:<br>**Dst: Object** - |

### 2.4.2.2 Dao

Dao interface is an abstraction between the manager package and the Dao classes containing all the necessary methods for accessing the data base which will be implemented by the generic DAO.

| Method name | Return type | Description |
|---|---|---|
| DataReaderMapToList<T> | List<T> | Parameters:<br>**dr: NpgsqlDataReader** - |
| delete | NpgsqlDataReader | aa<br>Parameters:<br>**id: Guid** - |
| executeQuery | NpgsqlDataReader | Parameters:<br>**query: string** - |
| getAll | NpgsqlDataReader | |
| getById | NpgsqlDataReader | Parameters:<br>**id: Guid** - |
| getByWhere | NpgsqlDataReader | Parameters:<br>**param: string** - |
| insert | NpgsqlDataReader | Parameters: |

| Method name | Return type | Description |
|---|---|---|
| | | **src: object** - |
| update | NpgsqlDataReader | Parameters:<br>**src: object** -<br>Parameters:<br>**dst: object** - |

### 2.4.3 Models

The models are classes according to the database.

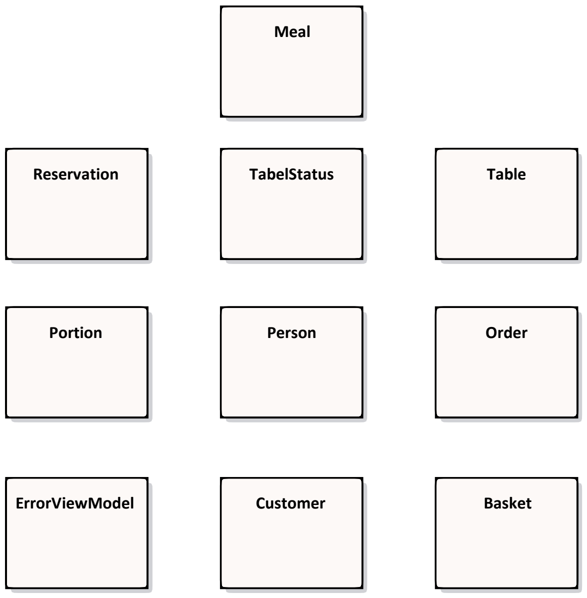


Figure 11 - Models

#### 2.4.3.1 Basket

An object creating a basket for portions.

#### 2.4.3.2 Customer

An object representing a customer of the restaurant having an email and a address in order to make booking and deliveries.

#### 2.4.3.3 ErrorViewModel

Asp.net class auto generated for handling the errors.

#### 2.4.3.4 Meal

An object which contains the necessary about the meal such as ingredients, allergens, name price etc.

#### 2.4.3.5 Order

An order object contains all the necessary information for a customer in order to make a delivery having a list of meals. For this case we will keep the id of basket and of the customer and some information about the status of the order and the preparation time for the list of meals.

#### 2.4.3.6 Person

A model for creating a client having a name.

#### 2.4.3.7 Portion

It represents an object containing the amount of meals ordered in a delivery or in a table reservation. For this specific reason we save the meal ID.

#### 2.4.3.8 Reservation

This specific class represents a model of what a reservation should keep in the database, such as the date and time and the number of persons of the booking, and also the status of the reservation in case is accepted or declined.

#### 2.4.3.9 TabelStatus

Table status was created with the purpose of making a mapping between the reservation and the table.

#### 2.4.3.10 Table

Represents the entity of a physical table in the restaurant.

## 2.5 Views

The Views package contains classes implementing the web user interface.

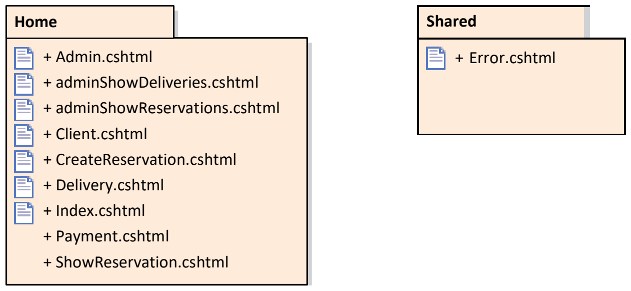


Figure 12 - Views

### 2.5.1 Home

Contains the main home controller which starts the web application for client and server.

#### 2.5.1.1 «web page» Admin.cshtml

Admin web page contains the 2 buttons for showing reservation and show delivery as functionality for handling the deliveries and the reservations.

#### 2.5.1.2 «web page» adminShowDeliveries.cshtml

In this form appears all the deliveries, and on this page the admin confirms/rejects the order.

#### 2.5.1.3 «web page» adminShowReservations.cshtml

The admin can confirm/reject the reservation on this form.

#### 2.5.1.4 «web page» Client.cshtml

Client web page contains the 2 buttons for showing reservation and show delivery as functionality for handling the deliveries and the reservations.

#### 2.5.1.5 «web page» CreateReservation.cshtml

Create Reservation contains a form which contains the dates about the client, number of persons, and in case the client want to choose some meals, he will be able to check the number of portions of each meals available.

#### 2.5.1.6 «web page» Delivery.cshtml

Delivery contains a form which contains the dates about the client, number of persons, and in case the client want to choose some meals, he will be able to check the number of portions of each meals available.

#### 2.5.1.7 «web page» Index.cshtml

Index is the main form of the application witch contains the client and the admin buttons for navigate between the forms.

#### 2.5.1.8 «web page» ShowReservation.cshtml

Show reservation is the form of the client where he can see his own reservation and its status (pending, accepted, declined).

### 2.5.2 Shared

View contains .cshtml pages for displaying information to the user.

#### 2.5.2.1 «web page» Error.cshtml

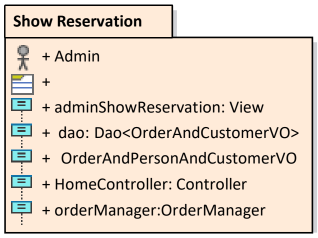
Automatically generated by the ASP.NET.

# 3. Realization Model

The realization model describes the realization of important Use Cases in the logic and structure of the system. Such realization is usually described by sequence diagrams consisting of lifelines of various objects and method calls between them.

## 3.1 Realization Model

In this chapter, the realization of some of the use cases of the Restaurant Information System is described, showing the communication of classes of the Web application.



Show Reservation

+ Admin
+
+ adminShowReservation: View
+ dao: Dao<OrderAndCustomerVO>
+ OrderAndPersonAndCustomerVO
+ HomeController: Controller
+ orderManager:OrderManager

Figure 13 - Realization Model

### 3.1.1 Reservation BL

The client tries to create a reservation after he inserted all the data into the reservation form and clicks on submit. The controller will receive the information and will try to find a table corresponding to his date, hour and the number of people. If he doesn't find any table it will redirect the client to the same page and show an error message. In case of finding a table, it will generate in the following order a basket, portion, person, customer, reservation and table status.
After the creation and insertion into the database of each object, the view will redirect client to the index page containing a message of success.

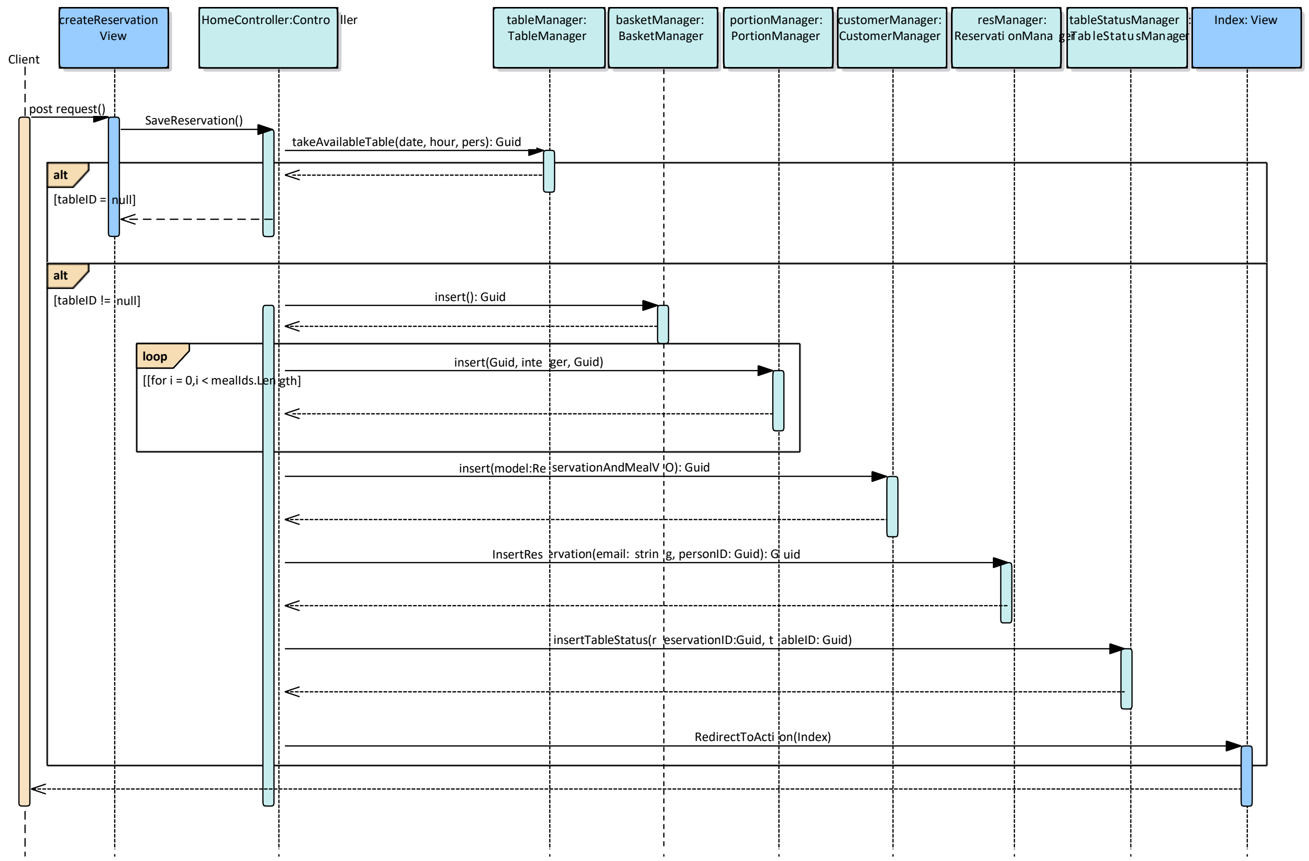
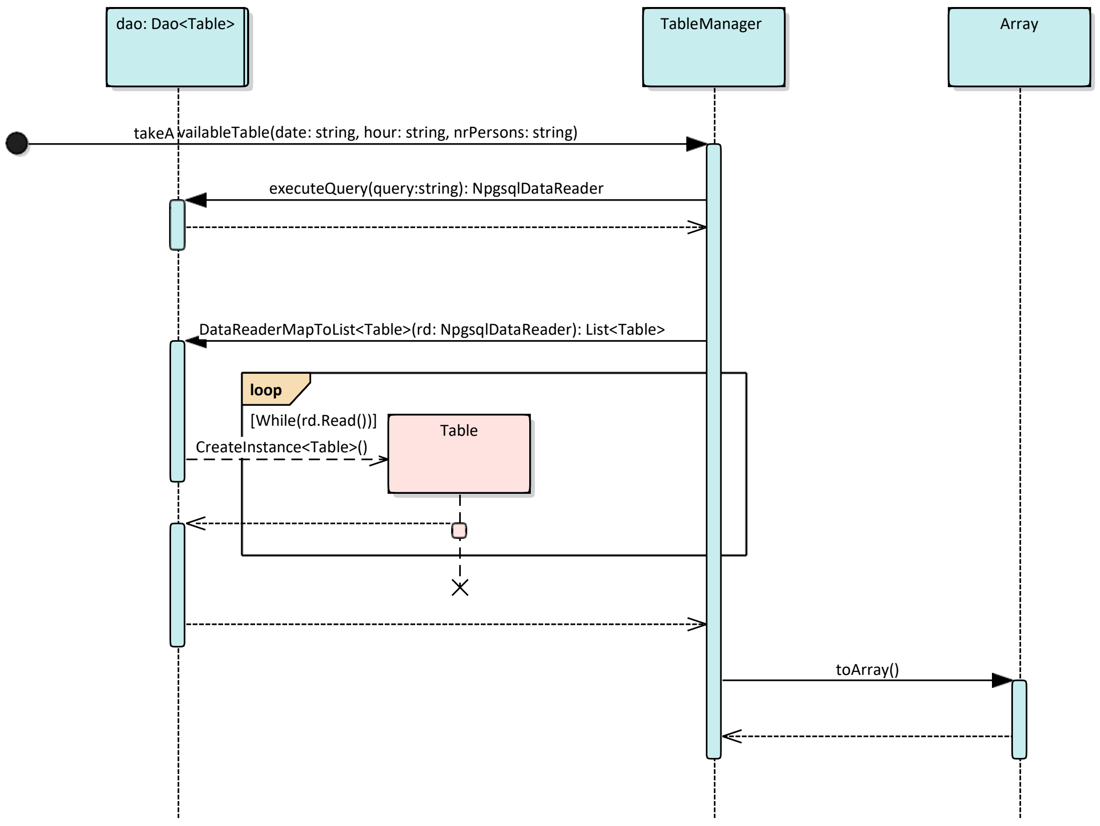

Client
createReservation View
HomeController:Controller
tableManager: TableManager
basketManager: BasketManager
portionManager: PortionManager
customerManager: CustomerManager
resManager: ReservationManager
tableStatusManager : TableStatusManager
Index: View
post request()
SaveReservation()
takeAvailableTable(date, hour, pers): Guid
alt
[tableID = null]
alt
[tableID != null]
insert(): Guid
loop
[[for i = 0,i < mealIds.Length]
insert(Guid, integer, Guid)
insert(model:ReservationAndMealVO): Guid
InsertReservation(email: string, personID: Guid): Guid
insertTableStatus(reservationID:Guid, tableID: Guid)
RedirectToAction(Index)

Figure 14 - Delivery BL

## 3.1.2 Reservation DL table

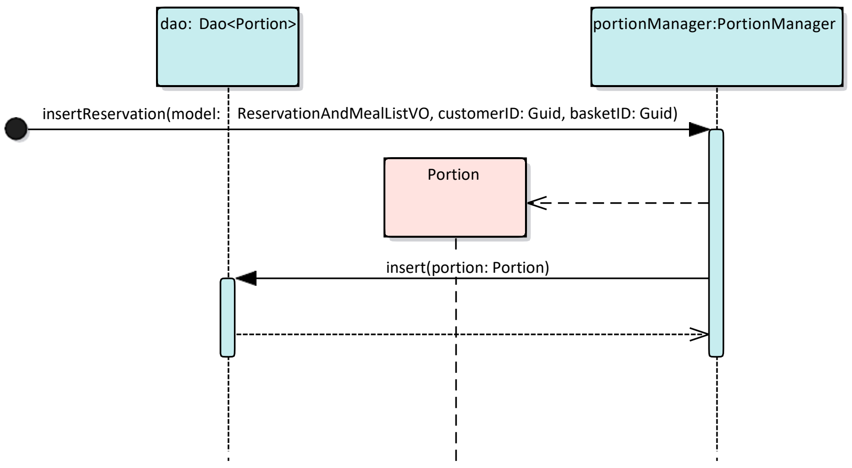


dao: Dao<Table>
TableManager
Array
takeAvailableTable(date: string, hour: string, nrPersons: string)
executeQuery(query:string): NpgsqlDataReader
DataReaderMapToList<Table>(rd: NpgsqlDataReader): List<Table>
loop
[While(rd.Read())]
CreateInstance<Table>()
Table
toArray()

Figure 15 - Delivery DL table

## 3.1.3 Reservation DL portion

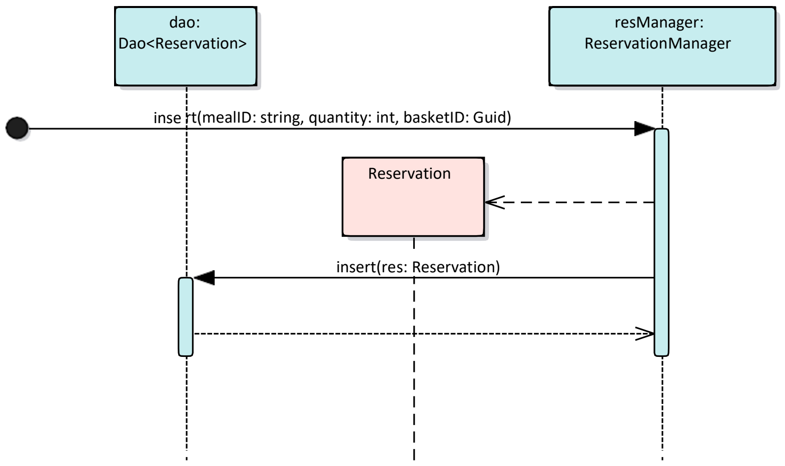
The portion manager will create a portion for each meal selected by the client. Based on the method called by the manager, it will take the arguments and create a portion object, it will populate it and insert into the database.



dao: Dao<Portion>
portionManager:PortionManager
insertReservation(model: ReservationAndMealListVO, customerID: Guid, basketID: Guid)
Portion
insert(portion: Portion)

Figure 16 - Delivery DL portion

## 3.1.4 Reservation DL reservation

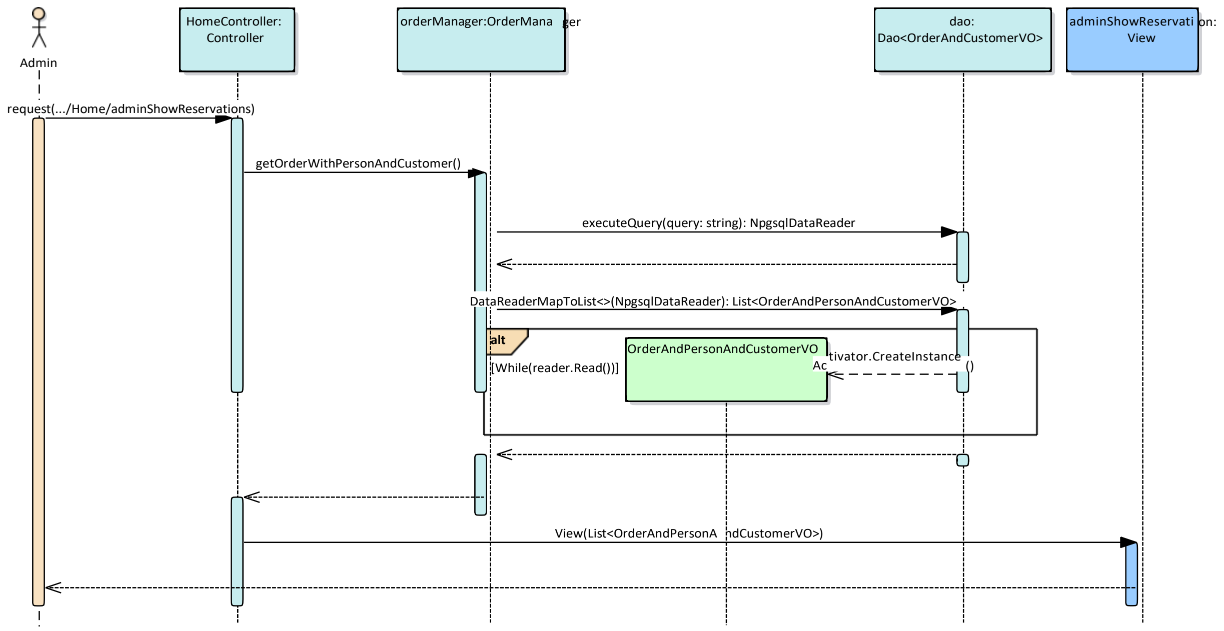


dao: Dao<Reservation>
resManager: ReservationManager
insert(mealID: string, quantity: int, basketID: Guid)
Reservation
insert(res: Reservation)

Figure 17 - Delivery DL reservation

## 3.1.5 Show Reservation

The administrator tries to view all the reservation in the database by clicking on button "show reservation". The controller class receives the message and calls the manager to get all the orders and also the information of the customer(having a specific person ID). Manager will call the query and for each line it will generate an object using the function of Mapping to a list. After the mapping is over, the list is returned to the manager and then to the controller which will call the view having as an argument the list of orders with clients.



Admin
HomeController: Controller
orderManager:OrderManager
dao: Dao<OrderAndCustomerVO>
adminShowReservation: View
request(.../Home/adminShowReservations)
getOrderWithPersonAndCustomer()
executeQuery(query: string): NpgsqlDataReader
DataReaderMapToList<>(NpgsqlDataReader): List<OrderAndPersonAndCustomerVO>
alt
[While(reader.Read())]
OrderAndPersonAndCustomerVO
Activator.CreateInstance()
View(List<OrderAndPersonAndCustomerVO>)

Figure 18 - Create reservation

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE