

**VEGEN FOOD ORDERING SYSTEM**

**Software Design Document**

– Ho Chi Minh, May 2021 –

**Table of Contents**

[I. Overview 3](#_Toc72826410)

[1. 1. Code Packages 3](#_Toc72826411)

[2. 2. Database Schema 4](#_Toc72826412)

[a. Accounts table 5](#_Toc72826413)

[b. Roles table 5](#_Toc72826414)

[c. OrderItems table 5](#_Toc72826415)

[d. Dishes table 6](#_Toc72826416)

[e. Orders table 6](#_Toc72826417)

[f. Menu table 7](#_Toc72826418)

[g. MenuItem table 7](#_Toc72826419)

[h. StatusID table 7](#_Toc72826420)

[II. Code Designs 8](#_Toc72826421)

[3. Class Diagram 8](#_Toc72826422)

[4. Class Specifications 8](#_Toc72826423)

[a. UserDAO class 8](#_Toc72826424)

[b. OrderDAO class 9](#_Toc72826425)

[c. DishDAO class 9](#_Toc72826426)

[d. HomeCookDAO class 10](#_Toc72826427)

[e. MenuDAO class 10](#_Toc72826428)

[5. Sequence Diagram(s) 11](#_Toc72826429)

[d.Database queries 11](#_Toc72826430)

# I. Overview

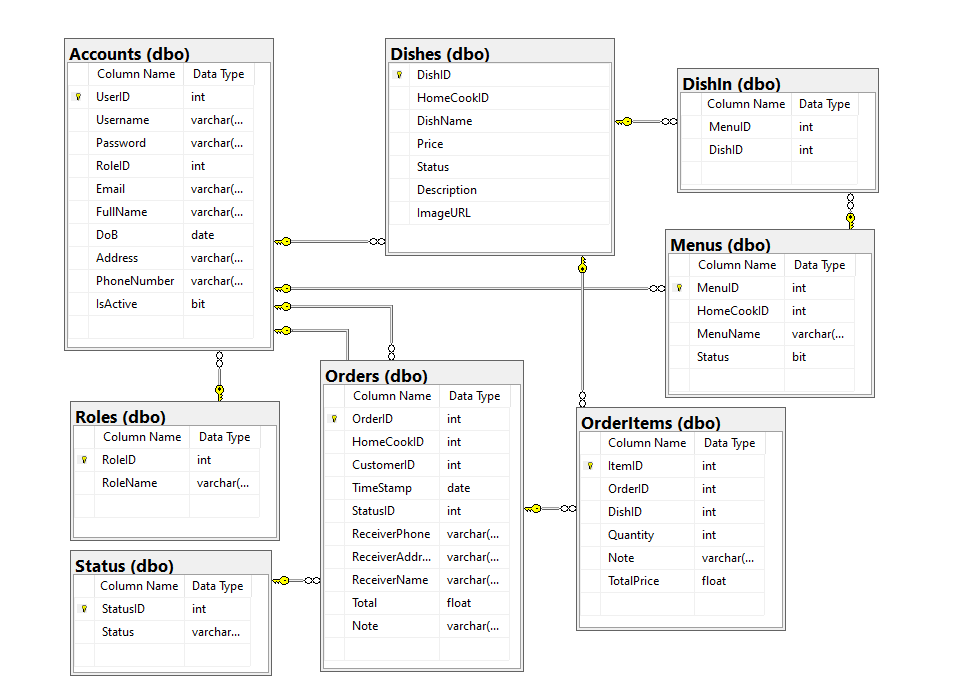
## 1. Code Packages



***Package descriptions & package class naming conventions***

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| 01 | Vegan System | The project’s folder |
| 02 | controller | The package which stores all of the controller-related packages  Naming convention: Naming convention: Filename is a noun, written in Pascal Case |
| 03 | filters | The package containing all of the system’s filters  Naming convention: Class name must be a noun, written in Pascal Case and follow the following syntax: noun + “Filter” |
| 04 | servlets | The package containing all of the system’s servlets  Naming convention: Class name must be a noun, written in Pascal Case and follow the following syntax: noun + “Servlet” |
| 05 | view | Contains all of the files related to displaying the data on the webpage  Naming convention: Filename is a noun, written in Pascal Case |
| 06 | model | Contains all of the system’s modules  Naming convention: Filename is a noun, written in Pascal Case |
| 07 | utils | Contains all of the utilities classes of the project, including data validation, database connection, …  Naming convention: Filename is a noun, written in Pascal Case |
| 08 | MenuManagement | Contains all of the component packages of menu management modules  Naming convention: Filename is a noun, written in Pascal Case |
| 09 | DishManagement | Contains all of the component packages of Dish management modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 10 | AccountManagement | Contains all of the component packages of Account management modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 11 | OrderManagement | Contains all of the component packages of Order management modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 12 | MenuDAO | Contains all of the component classes of Menu Management Modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 13 | MenuDTO | Contains the structure of Menu Object, including fields and constructors, …  Naming convention: Class name must be a noun, written in Pascal Case |
| 14 | DishDAO | Contains all of the component classes of Dish Management Modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 15 | DishDTO | Contains the structure of Dish Object, including fields and constructors, …  Naming convention: Class name must be a noun, written in Pascal Case |
| 16 | OrderDAO | Contains all of the component classes of Order Management Modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 17 | OrderDTO | Contains the structure of Order Object, including fields and constructors, …  Naming convention: Class name must be a noun, written in Pascal Case |
| 18 | AccountDAO | Contains all of the component classes of Account Management Modules  Naming convention: Class name must be a noun, written in Pascal Case |
| 19 | AccountDTO | Contains the structure of Account Object, including fields and constructors, …  Naming convention: Class name must be a noun, written in Pascal Case |

## 2. Database Schema



***Table descriptions & package class naming conventions are as below***

#### Accounts table

The table stores the information of a specific user in the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | UserID | int | 4 bytes | X | X | PK | Auto\_increment |
| 2 | Username | varchar | 20 | X | X |  |  |
| 3 | Password | varchar | 50 |  | X |  |  |
| 4 | RoleID | int | 4 bytes |  | X | FK |  |
| 5 | Email | varchar | 50 | X | X |  |  |
| 6 | FullName | varchar | 50 |  | X |  |  |
| 7 | DoB | date | 3 bytes |  | X |  |  |
| 8 | Address | varchar | 100 |  | X |  |  |
| 9 | PhoneNumber | varchar | 12 | X | X |  |  |
| 10 | Status | bit | 1 byte |  | X |  |  |

#### Roles table

The table stores the information of user’s roles in the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | RoleID | int | 4 bytes | X | X | PK | Auto\_increment |
| 2 | Role Name | varchar | 20 | X | X |  |  |

#### OrderItems table

The table stores the information of the order items in the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | ItemID | int | 4 bytes | X | X | PK | Auto\_increment, Starts from 1 |
| 2 | OrderID | int | 4 bytes |  | X | FK | The orderID, dishID pair’s value must be unique |
| 3 | DishID | int | 4 bytes |  | X | FK |
| 4 | Quantity | int | 4 bytes |  | X |  | From at least 1 |
| 5 | Note | varchar | 200 |  | X |  |  |
| 6 | TotalPrice | float | 4 bytes |  | X |  | Unit in thousand VND |

#### Dishes table

The table stores the information of all of the dishes the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | DishID | int | 4 bytes | X | X | PK | Auto\_increment, Starts from 1 |
| 2 | UserID | int | 4 bytes |  | X | FK |  |
| 3 | MenuID | int | 4 bytes |  | X | FK |  |
| 4 | DishName | varchar | 100 |  | X |  |  |
| 5 | Price | float | 4 bytes |  | X |  | Unit is in thousand VND |
| 6 | Status | bit | 1 byte |  | X |  |  |
| 7 | Description | varchar | 300 |  | X |  |  |
| 8 | ImageURL | varchar | 100 |  | X |  |  |

#### Orders table

The table stores the information of the orders in the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | OrderID | int | 4 bytes | X | X | PK | Auto\_increment, Starts from 1 |
| 2 | UserID | int | 4 bytes |  | X | FK |  |
| 3 | TimeStamp | date | 4 bytes |  | X |  |  |
| 4 | Status | int | 4 byte |  | X |  |  |
| 5 | ReceiverPhone | varchar | 12 |  | X |  |  |
| 6 | ReceiverAddress | varchar | 120 |  | X |  |  |
| 7 | ReceiverName | Varchar | 50 |  | X |  |  |
| 8 | Total | float | 4 bytes |  | X |  | Unit is in Thousand VND |
| 9 | Note | varchar | 200 |  | X |  |  |

#### Menu table

The table stores the information of all of the menus in the system

Details of the fields in the table are given in the bellow table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | MenuID | int | 4 bytes | X | X | PK | Auto\_increment, Starts from 1 |
| 2 | UserID | int | 4 bytes |  | X | FK |  |
| 3 | MenuName | varchar | 100 |  | X |  |  |
| 4 | Status | bit | 1 byte |  | X |  |  |

#### MenuItem table

The table stores the relation of Menus and Dishes in the system

Details of the fields are given in the table below

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | MenuID | int | 4 bytes |  | X | FK | The MenuID, DishID pair’s value must be unique |
| 2 | DishID | int | 4 bytes |  | X | FK |

#### StatusID table

The table stores the information of status name and the status code in the system

Details of the fields are given in the table below

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***#*** | ***Field name*** | ***Type*** | ***Size*** | ***Unique*** | ***Not Null*** | ***PK/FK*** | ***Notes*** |
| 1 | StatusID | int | 4 bytes | X | X | PK | Auto\_increment, Starts from 1 |
| 2 | Status | varchar | 20 |  | X |  |  |

# II. Code Designs

## Class Diagram

**

## Class Specifications

### a. UserDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1 | getUserByID(int) | The function takes the “UserID” as the input parameter. Then, it will run the query in the database for the corresponding user information. Then, it will return the appropriate Account object. If no user is found, it would return null |
| 2 | getUserByUNameAndPass(String, String) | The function takes Username and Password entered by users as the input parameters. It will then query the database for the corresponding information. If the user and passwords match, it will return an Account type object which contains all of the information of that account, else, it would return null |
| 3 | createUserAccount(Account) | The function takes an Account type object as the input parameter. It will then create a new record in the database for that account. If the operation succeed, it will return true, else, it will return false |
| 4 | updateUserInfoByID(Account) | The function takes an Account type object as the input parameter. It will then update the corresponding account record in the database with new inputs. If the operation succeed, it will return true, else, it will return false |

### b. OrderDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1 | getOrderByUserID(int) | The function takes the “UserID” as the input parameter. Then, it will run the query in the database for all of the order related to the corresponding UserID. It will then return an ArrayList of Order type to the user. If no record is found, it would return null |
| 2 | createNewOrder(Order) | The function takes an Order type object as the input parameters. Then, it will create a new record of the order with the input data before creating new records of order items in the database. If the operation succeed, it will return true, else, it will return false |
| 3 | changeOrderStatus(int, Boolean) | The function takes the OrderID and the status as input parameters. It will then query the database and update the corresponding order with the input data. If the operation succeed, it will return true, else, it will return false. |
| 4 | getOrderByTime(string) | The function takes the date string as the input parameter. It will then query the database for all of the order with the corresponding date. It will return an ArrayList of Order type, which contains all of the data. If no order is found, it will return null |
| 5 | getOrderItemByID(int) | The function takes the OrderID as the input parameter. Then, it will query the database for all the order items with the corresponding orderID. It will return an ArrayList containing all of the items. It no items is found, it will return null. |

### c. DishDAO class

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1 | getAllDishByHomeCook(int) | The function takes the “UserID” as the input parameter. Then, it will run the query in the database for all of the dishes related to the corresponding UserID. It will then return an ArrayList of Dish type to the user. If no record is found, it would return null |
| 2 | getAllDishByStatus(boolean) | The function takes a Boolean value as the input parameter. It will then query the database for all of the dishes with the corresponding Boolean value before return an ArrayList of Dish type. If there is no record found, it will return null |
| 3 | createDish(Dish) | The function takes a Dish type object as the input parameter. It will then create a new record of the dish in the database. If the operation success, it will return true, else, it will return false |
| 4 | updateDish(Dish) | The function takes a Dish type object as the input parameter. It will then update the corresponding record in the database with the input information. If the operation success, it will return true, else, it will return false |
| 5 | changeDishStatus(int, boolean) | The function takes the DishID and a Boolean value as the input parameters. It will the update the corresponding dish in the database with the new data input. If the operation succeed, it will return true, else, it will return false. |
| 6 | getAllDishByOrderID(int) | The function will take the OrderID as the input parameter. It will then query the database for all of the dishes with the corresponding orderID. It will return an ArrayList of Dish type. If there is no dish, it will return null. |

### d. HomeCookDAO class

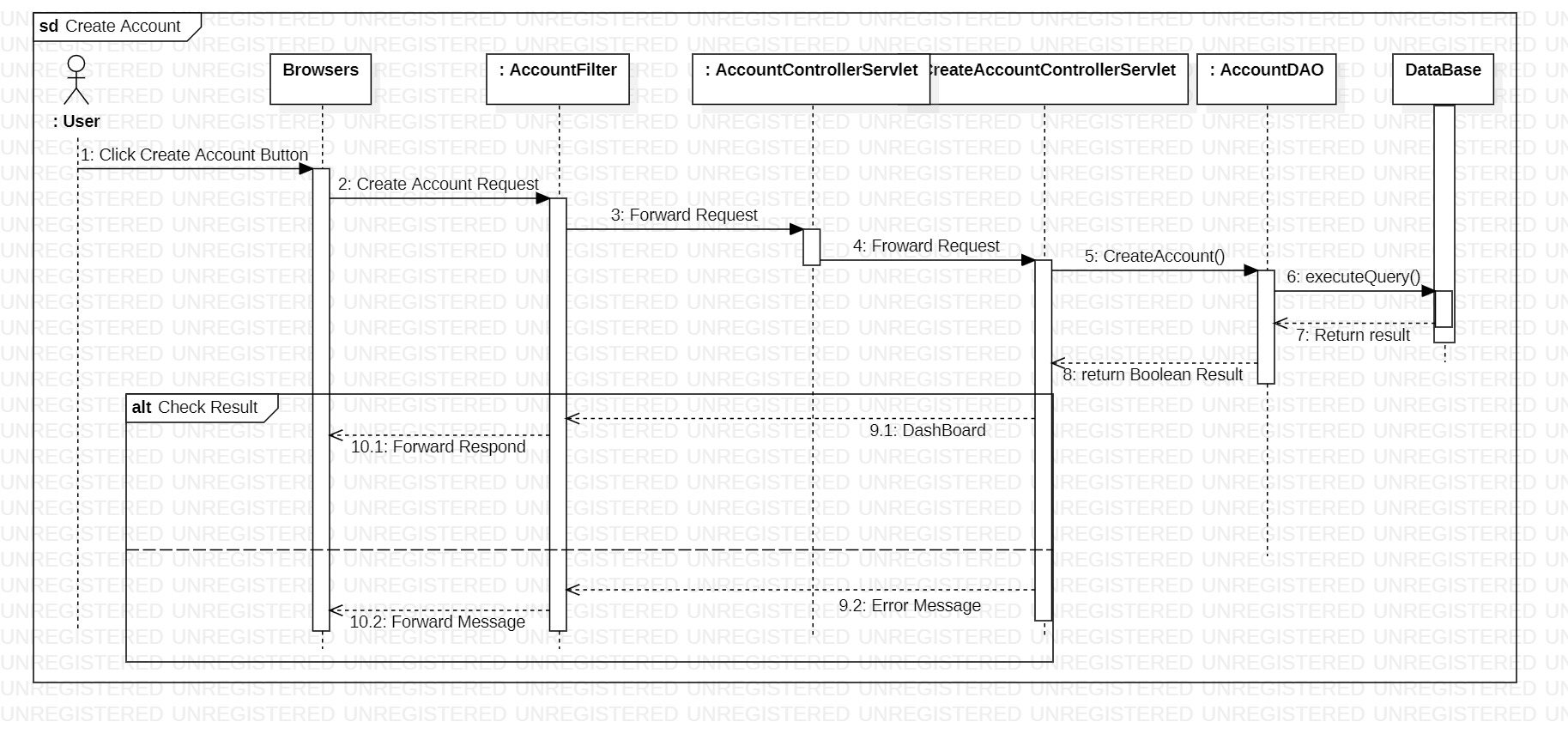
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1 | getAllHomeCook() | The function takes no input parameter. It will query the database for all of the existing records of home-cook. It will then return an ArrayList of home-cook if there are some record, else, it will return null |
| 2 | changeAccountStatus(int,Boolean) | The function takes UserID and a Boolean variable as the input parameter. It will then update the corresponding record with the input data. If the operation succeed, it will return true, otherwise, false. |
| 3 | creatHomeCookAccount(Account) | The function takes an Account type object as the input parameter. It will then create a new record in the database for that account. If the operation succeed, it will return true, else, it will return false |

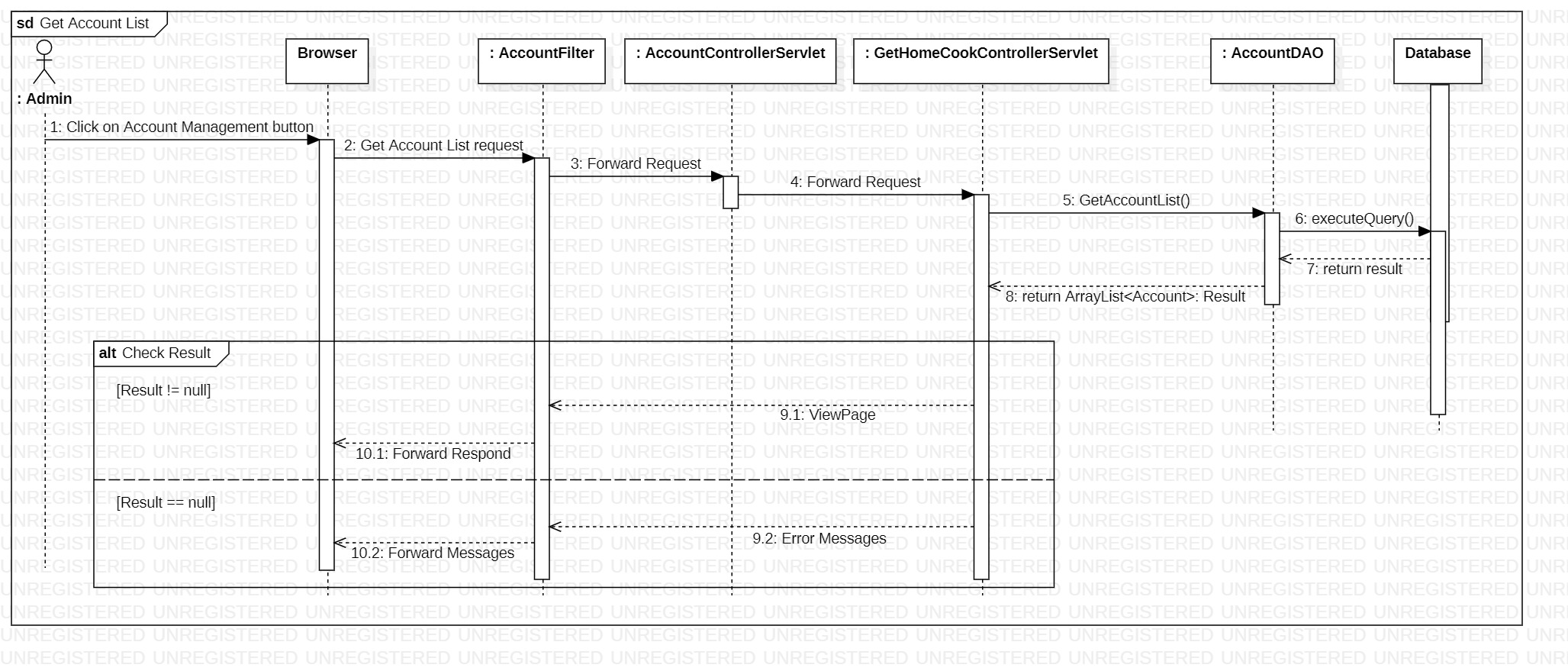
### e. MenuDAO class

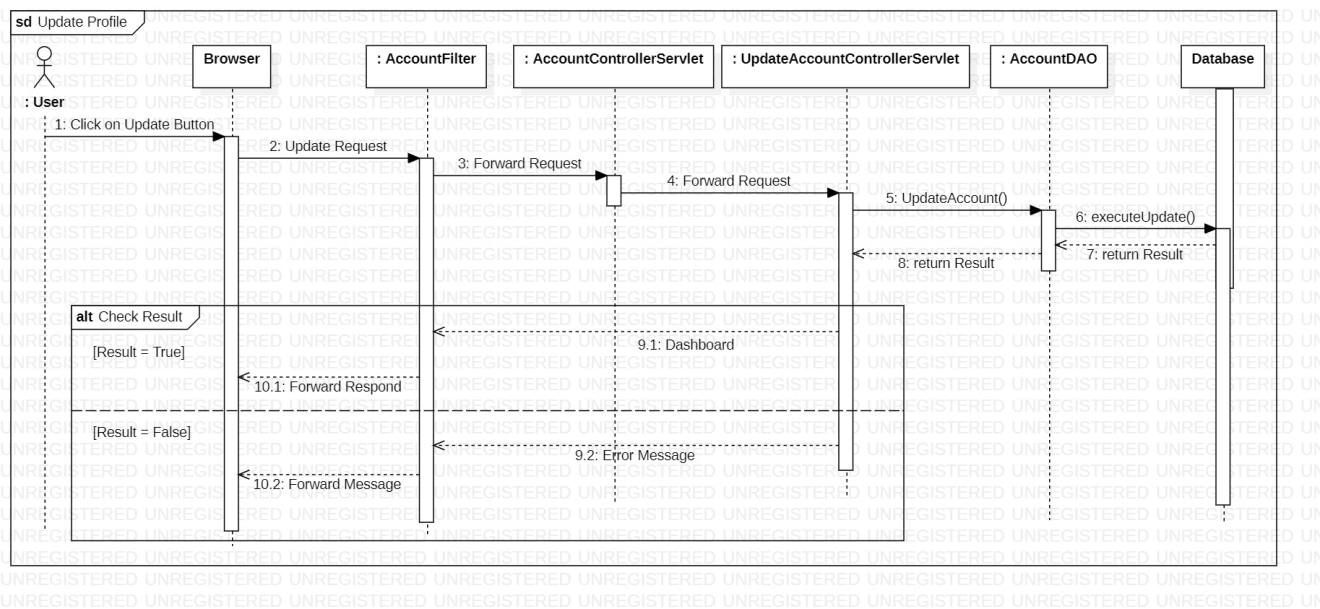
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1 | getAllMenuByHomeCook(int) | The function takes the “UserID” as the input parameter. Then, it will run the query in the database for all of the menus related to the corresponding UserID. It will then return an ArrayList of Menu type to the user. If no record is found, it would return null |
| 2 | createMenu(Menu) | The function takes a Menu type object as the input parameter. Then, it will create a new record of the menu with the input data in the database. If the operation succeed, it will return true, else, it will return false. |
| 3 | updateMenu(Menu) | The function takes the Menu object as the input parameter. It will then query the database and update the corresponding Menu with the input data. If the operation succeed, it will return true, else, it will return false. |
| 4 | getAllMenuByStatus(boolean) | The function takes the Boolean variable as the input parameter. It will then query the database for all of the order with the corresponding status. It will return an ArrayList of Menu type, which contains all of the data. If no order is found, it will return null |
| 5 | getOrderItemByID(int) | The function takes the OrderID as the input parameter. Then, it will query the database for all the order items with the corresponding orderID. It will return an ArrayList containing all of the items. It no items is found, it will return null. |

## Sequence Diagram(s)

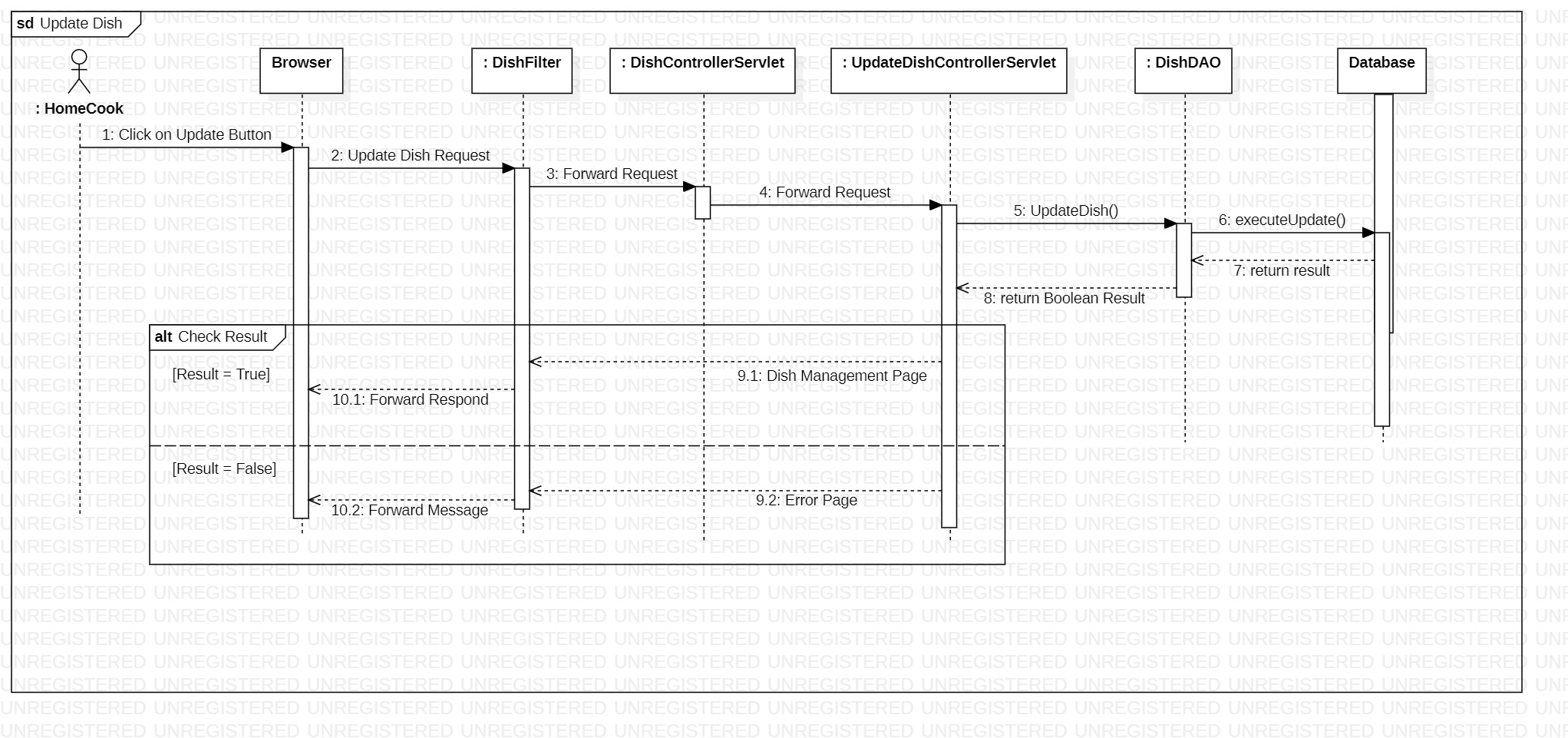
### Account Management

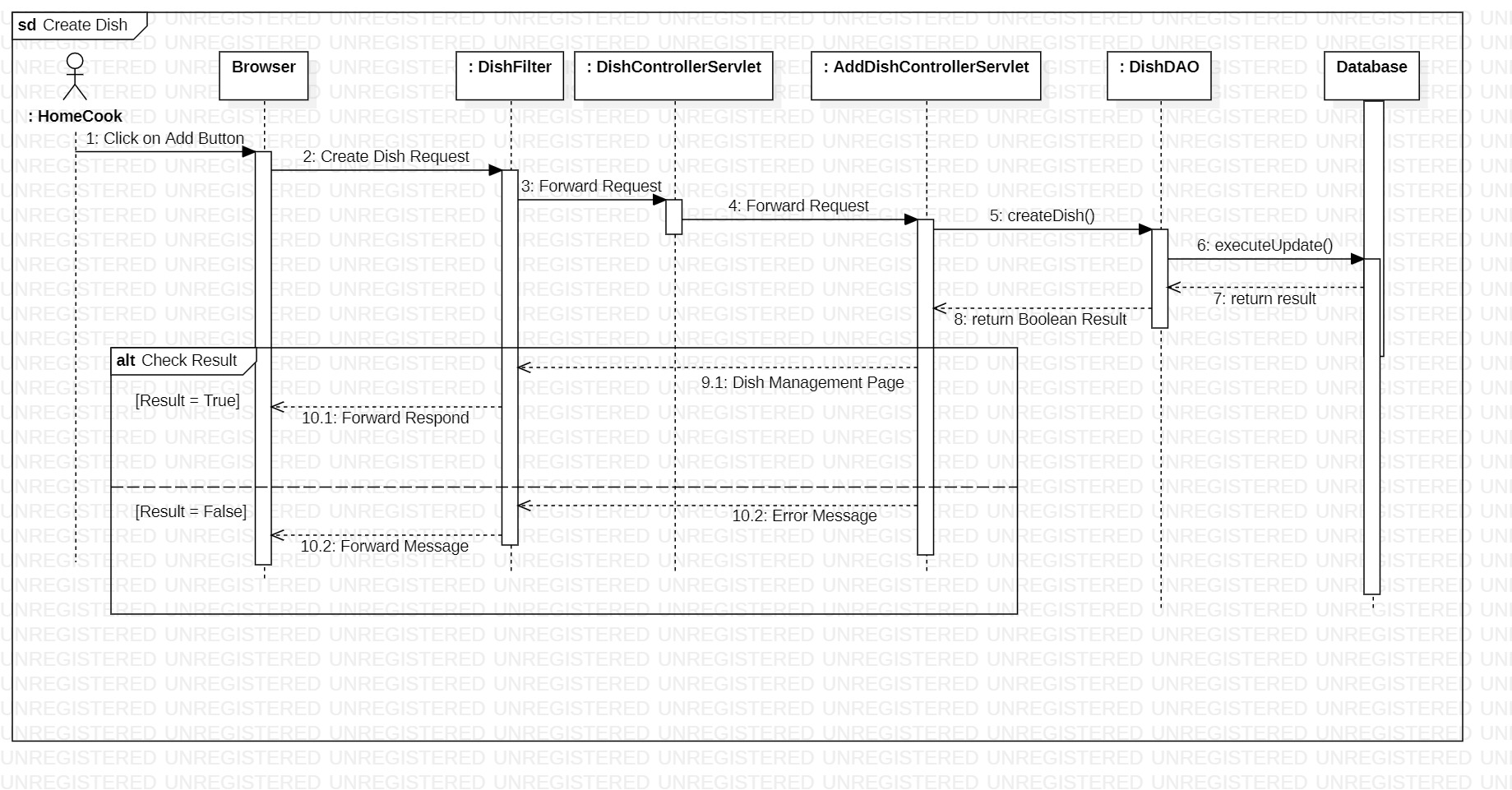


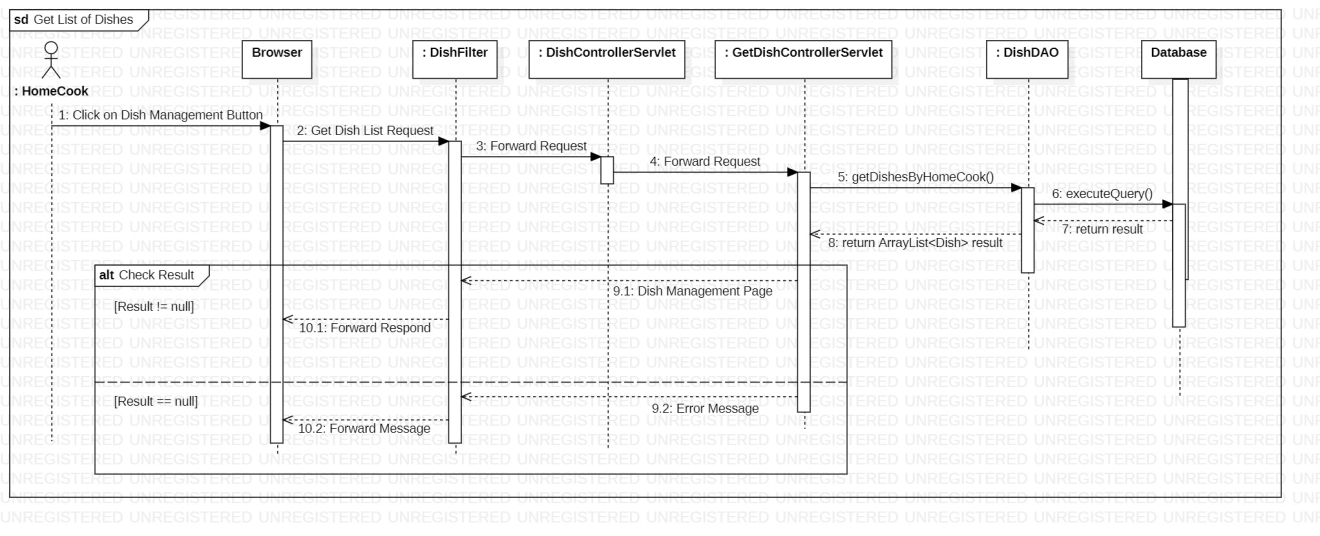




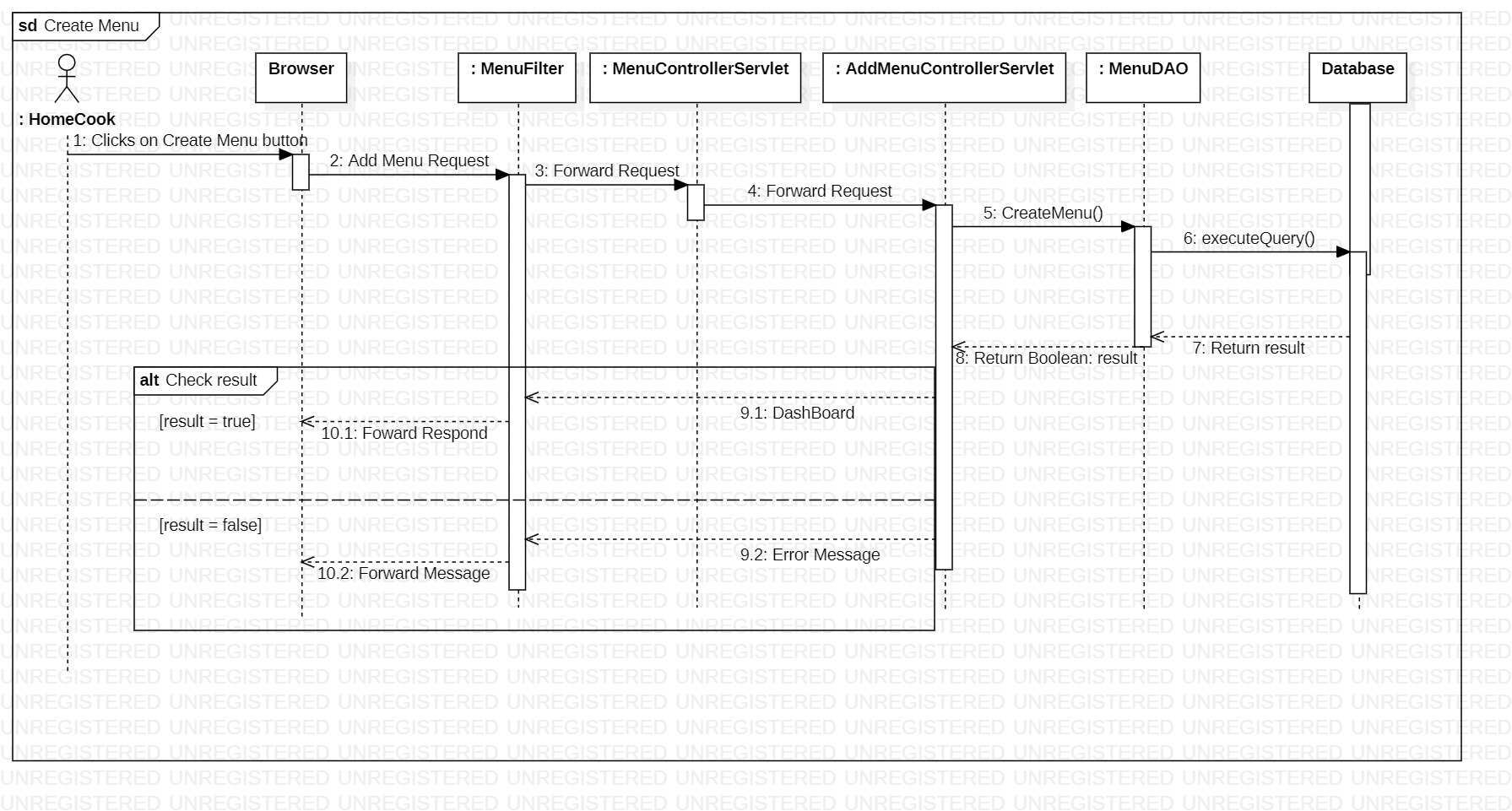
### Dish Management

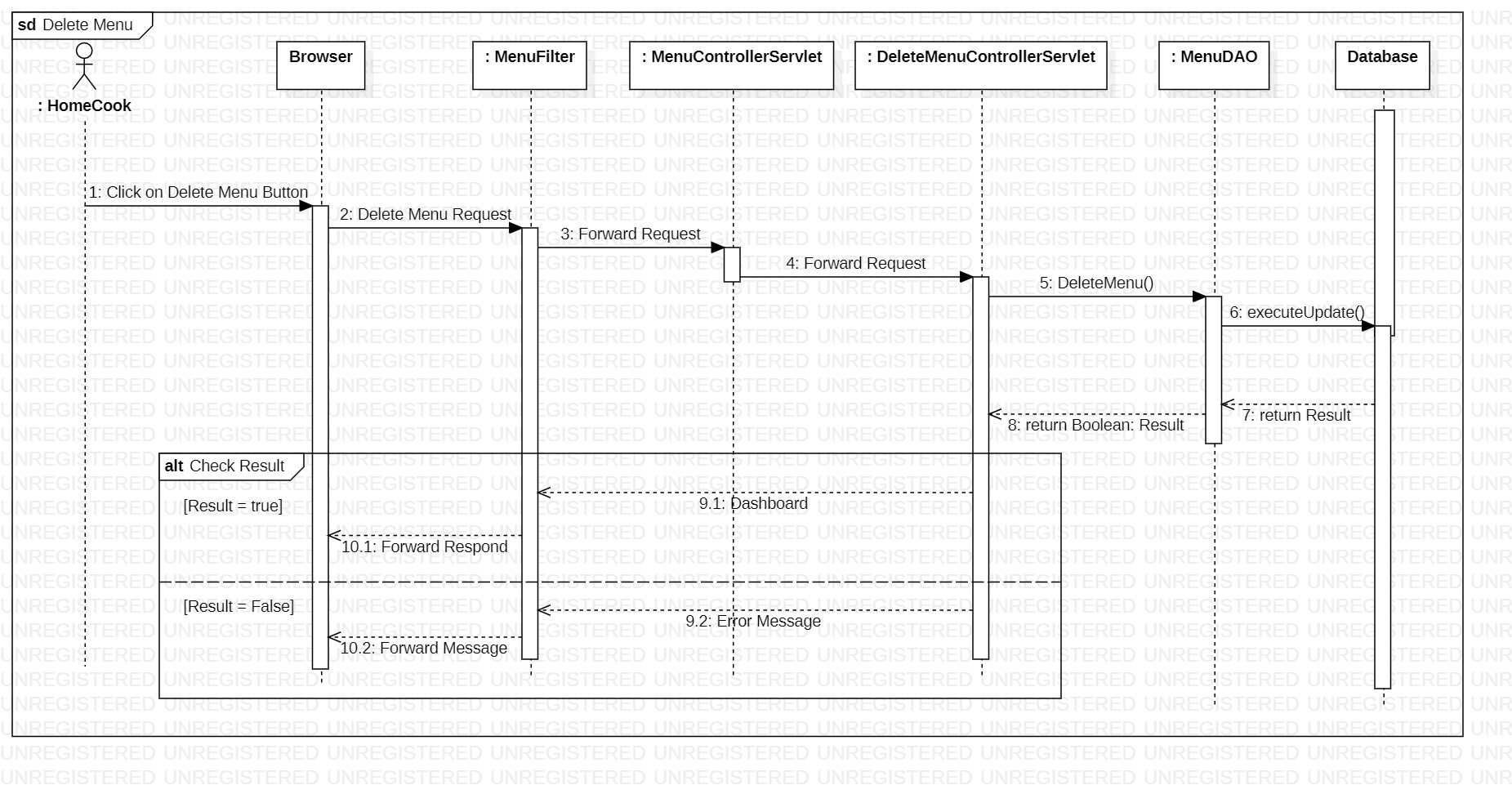


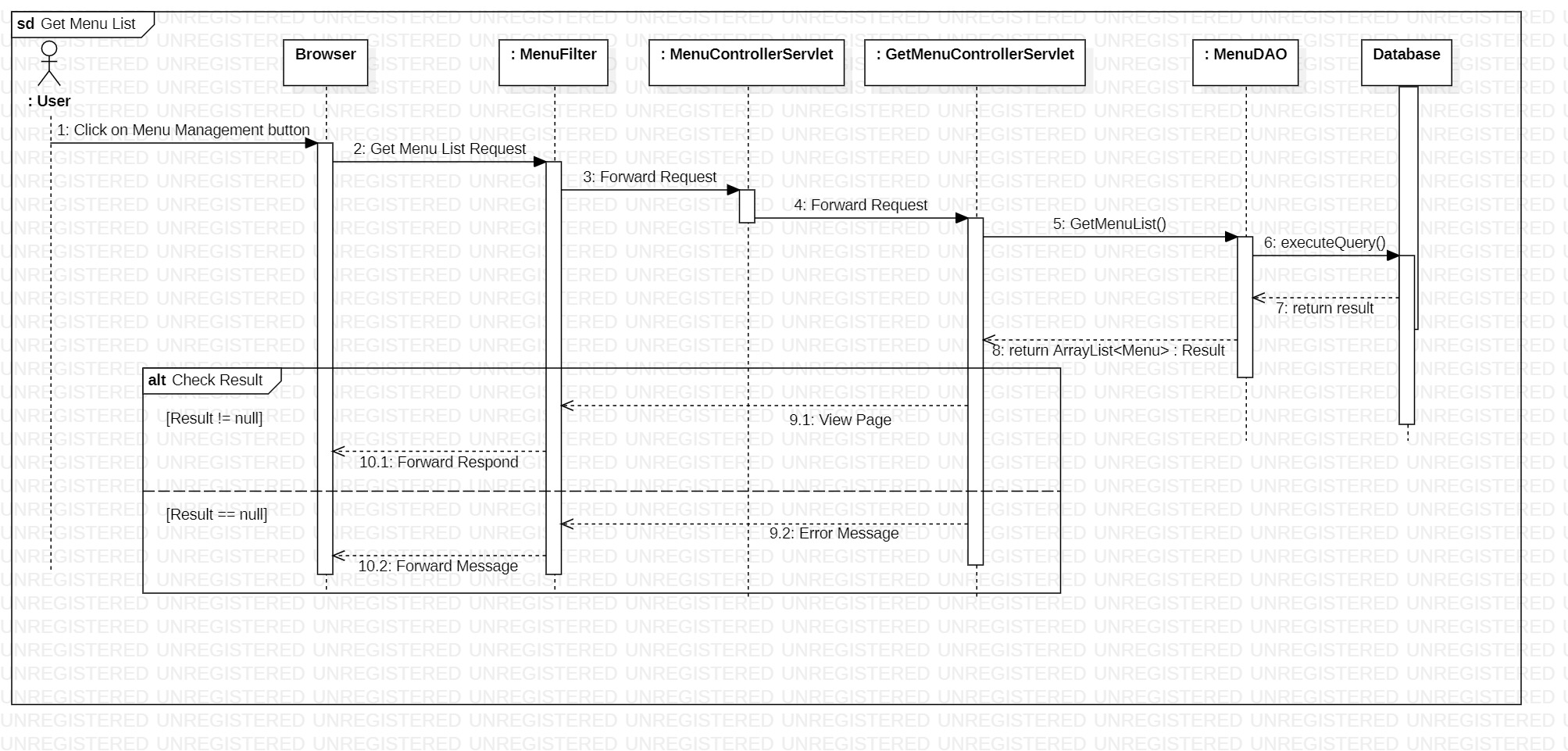


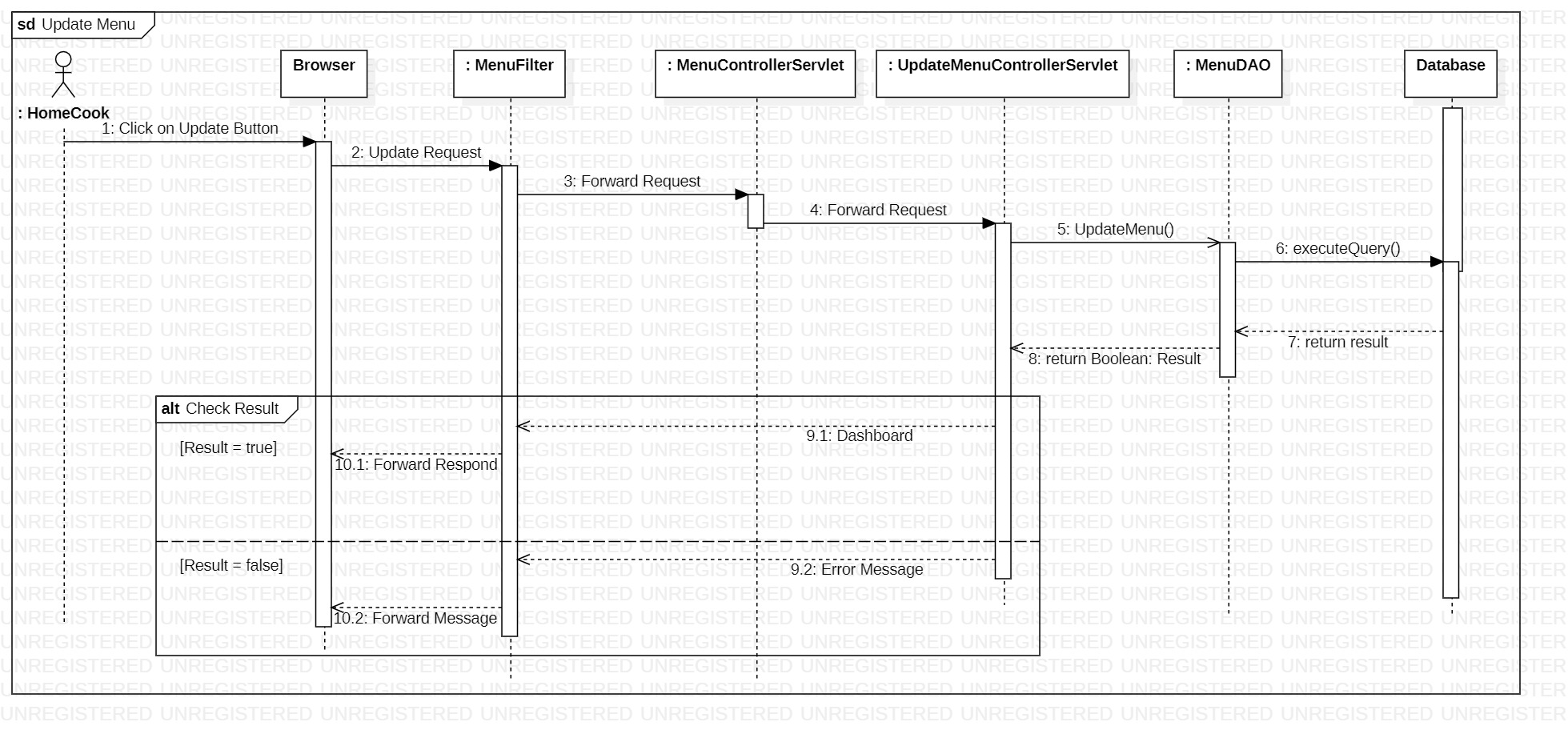


### Menu Management









### Order Management

## Database queries

### Account Management

#### createUserAccount

INSERT INTO Accounts VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

#### getAccountByID

SELECT \* FROM Accounts WHERE UserID = ?

#### getAccountByUNameandPassword

SELECT \* FROM Accounts WHERE Username = ? AND Password = ?

#### updateUserInfo

UPDATE