

Distribution and Integration Technologies

Restaurant Order and Accounting System

João Almeida | João Mendes



FEUP **FACULDADE DE ENGENHARIA**
UNIVERSIDADE DO PORTO

Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL

1. Introduction	4
2. Software Architecture	4
2.1. Products	5
2.1.1 Product	5
2.1.2. Order	6
2.2. Controllers	6
2.2.1. Dining Room Controller	6
2.2.2. Kitchen Bar Controller	7
2.2.3. Payment Zone Controller	7
2.3. Network	8
2.3.1. Payment Zone Node	8
2.3.1.1. Remotably Invoked Methods	8
2.3.1.2. Events	9
2.3.2. Event Repeater	9
3. User Interface	11
3.1. Dining Room Window	12
3.1.1 Product List and Current Order Frame	12
3.1.2 Orders Ready	12
3.1.3 Table Insertion	13
3.1.4 Undo Button	13
3.1.5 Apply Button	13
3.1.6 Miscellaneous Buttons	13
3.2 Kitchen or Bar Window	13
3.2.1. Not Picked and Preparing Frame	14
3.2.2. Miscellaneous Buttons	14
3.3. Order Detail Window	15
3.3.1 Back Button	15
3.4. Payment Zone Window	15
3.4.1. Table Selection Buttons	17
3.4.2. Payment Frame	17
3.4.3. Paid Button	17
3.4.4. Total Money Label	17
4. Conclusion	18

1. Introduction

A Restaurant needs to automate the dining room orders, allowing them to be quickly communicated to the kitchen and bar tenders, to be prepared as soon as possible. Also, the same system should maintain a complete record of all orders, compute the bills of each table, and maintain a record of the total amount received in the day.

In the dining room there are several terminals (client computers) where the waiters can make new orders (in any of the terminals), opening a new table or adding to the orders already assigned to a table. Those orders can have as destination the kitchen (dishes) or the bar (drinks). When the orders become prepared, cooks and bartenders should signal the event, letting the waiters know that they can fetch the orders and serve them to the tables. In the bar area and the kitchen there is also a terminal where the orders should appear, already divided (bar orders in the bar terminal and kitchen orders in the kitchen terminal).

When a free bar tender or cook picks one order to prepare it, he should signal that in the terminal changing its status from 'not picked' (the original status of any order) to 'in preparation', preventing other person to prepare the same order. When the order is prepared (drink or dish) the same person should go to the terminal (bar or kitchen) and change its status to 'ready', and that should be signaled or shown in the dining room terminals.

2. Software Architecture

The architecture we decided to use for this project is based on the Model-view-controller design pattern. This structure allows us to have lower coupling between the different modules as well as facilitating the process of multiple developers working simultaneously on the same project. Furthermore, although the project is finished as is, if in the future we intend to continue its development because of the separation of responsibilities this structure provides us, the further development and modification is easier.

Below is a simple representation of the Model-view-controller architecture we have used for the project.

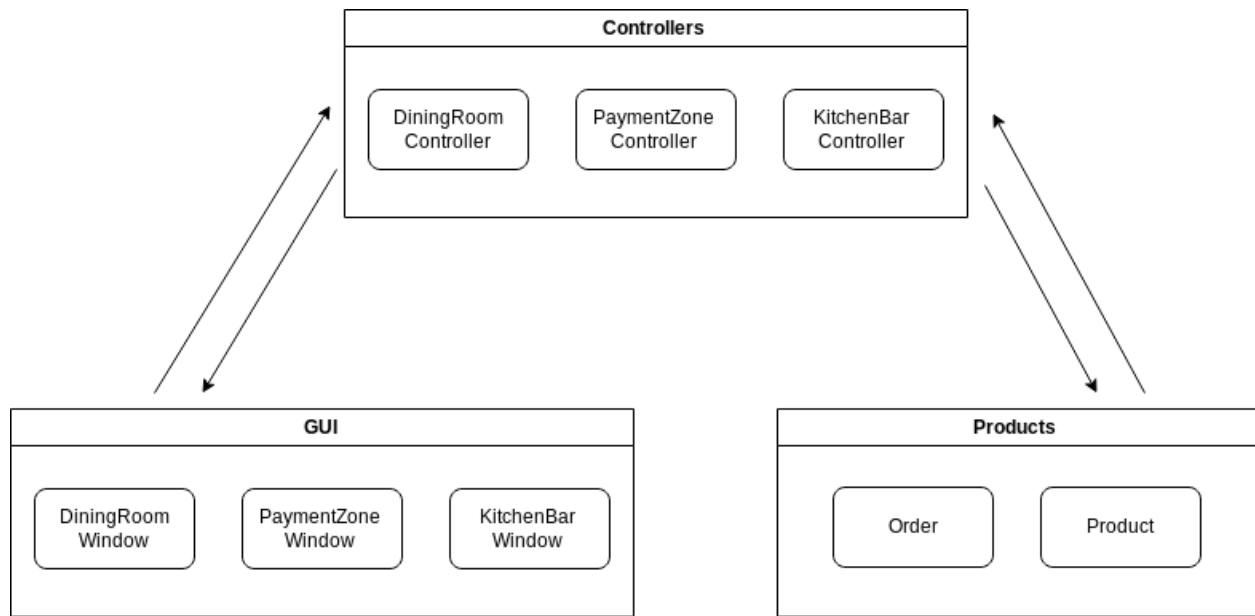


Fig. 1 - Architecture of the modules of the project

2.1. Products

Closer to the Model part of the Model-view-controller structure. In here we store information about the products available on the restaurant and the multiple orders that were either just placed or that were already paid.

2.1.1 Product

Represents a single product that the restaurant offers to its customers. A product can have 2 types, Dish or Drink. This affects the type of controller that will handle this product. Products have an associated name, price and a time of confection. The time of confection is a parameter that we do not use in our project, but we thought could be fun to play with in the future so we decided to add support for it.

The information of what products the restaurant offers, the cost and time of confection is all read from a .csv file present in the asset/products folder. This means that we can easily add more products to our restaurant. The files follow the following structure:

```
<product_name>;<product_price>;<product_time_of_confection>
```

2.1.2. Order

An order represents a single order placed from a table. The order has a unique identifier, and has an associated table. A single order can have multiple products associated, meaning that an order can have both dishes and drinks associated with it. Depending on the type of products the order has it can be of one of three types (Kitchen, Bar or Both), this information is used to know what controller is responsible for this order. Orders also have an associated state which can be one of four types (Not Picked, Preparing, Ready and Paid).

To facilitate the handling of simple orders that only have one type of product and composite orders that have multiple type of products, we have used the Composite design pattern, a simple overview can be seen below:

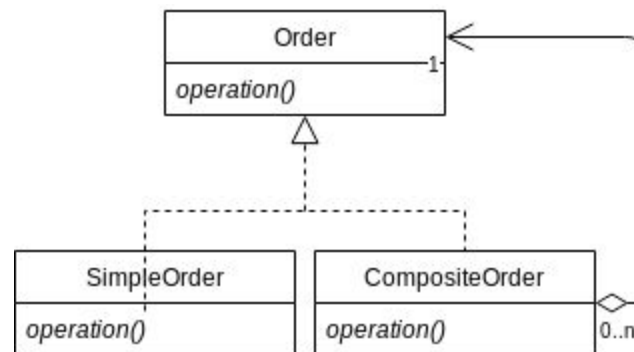


Fig. 2 - Order composite design pattern representation

2.2. Controllers

The controllers module is responsible for receiving the user input from the GUI and updating the Products module accordingly. Most of the project's logic is located in this module. The networking aspects of the project is also present in here, and every controller connects to the intranet.

2.2.1. Dining Room Controller

In the dining room controller, orders are built from a set of available products and a list of tables the orders are destined to. This controller would be the controller interacting with the waiters of the restaurant. After an order is built and sent into the network, the controller still waits for a confirmation that the order is ready to be picked up. Should that be the case, the waiter is signalled to deliver the order to the table.

This controller communicates only with the Payment Zone Controller, it has no connection whatsoever with the Kitchen Bar Controller.

2.2.2. Kitchen Bar Controller

The Kitchen Bar Controller is responsible for handling the orders sent into the kitchen and the bar. Independently of whether the application running is the Bar or the Kitchen it's functionality is the same, the only difference is the type of orders it receives. If the controller is registered as a Kitchen Controller, then it will only receive the products that are dishes, if it is a Bar Controller it will only receive drinks.

On network join, the controller communicates immediately with the Payment Zone Controller, to signal whether it is a Bar Controller or a Kitchen Controller. Similarly to the Dining Room Controller, this controller communicates only with the Payment Zone Controller.

2.2.3. Payment Zone Controller

Central controller, where most of the program's logic is located. It is responsible for sorting out the Orders received from the dining room, and send its respective products to the correspondent Kitchen or Bar Controller. Should an Order have multiple product types (dishes and drinks), then this controller splits the Order in two, and sends the split orders to the respective Kitchen or Bar. Once both the Kitchen and the Bar have prepared the orders, then the Dining Room is signalled that the Order is ready. When an order has only one type of products then as soon as it is ready it is signalled.

Once the Dining Room signals that the orders have been delivered to the table, then these are now available for payment. This controller merges multiple orders of the same table, and shows them in the terminal.

2.3. Network

Due to the nature of the architecture of the project, multiple computers will be running the program. To allow communication between the different nodes it is necessary for them to communicate through an established network.

Below is a simplified representation of how the communication between the different nodes is made:

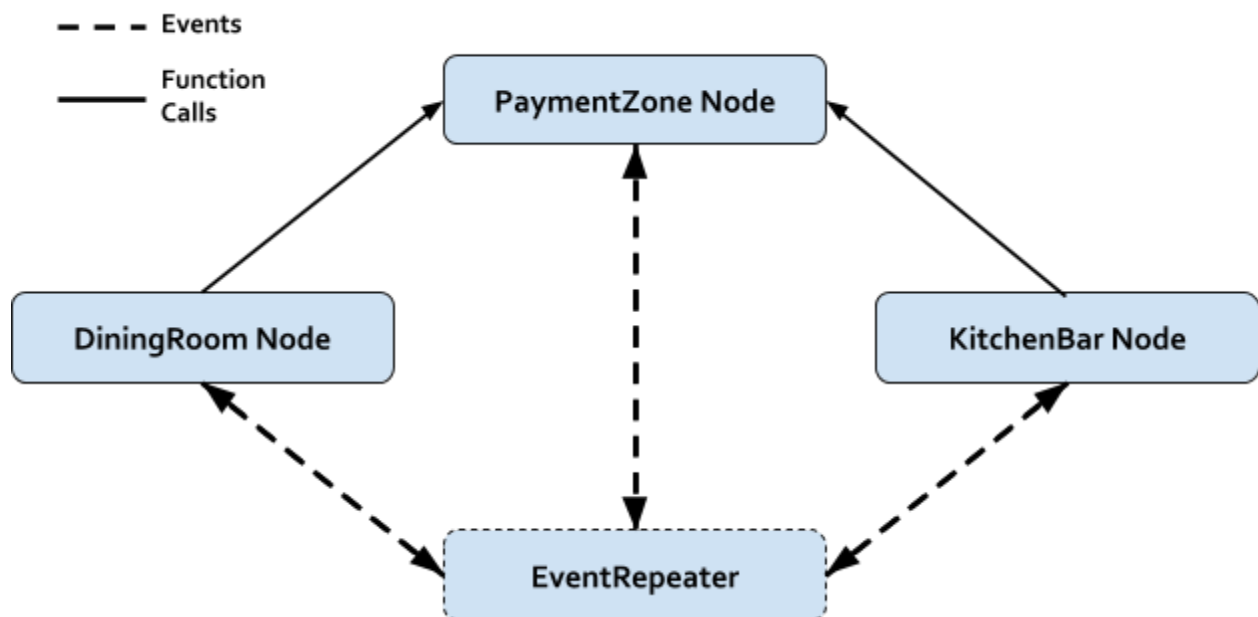


Fig. 3 - Representation of the network communication

2.3.1. Payment Zone Node

Central node which is responsible for storing all of the information about the restaurant. This means that it serves as a midpoint for the communication between the KitchenBar and the DiningRoom.

2.3.1.1. Remotably Invoked Methods

Methods invoked by either KitchenBar or DiningRoom to signal that a change in state of an order has occurred.

- `void NewOrder(Dictionary<string, uint> products, uint table_n)`

Called by DiningRoom when a new order is created. Contains the products name, respective amounts and the table number to assign the order.

- `void OrderReady(long order_id, uint table_n, bool from_kitchen)`
Called by KitchenBar when an order is ready. Contains the ID of the order, its table number and whether the callee was the Kitchen or the Bar. Internally the PaymentZoneController will check if all of the order is actually ready, or if it is only partly ready.
- `void OrderDelivered(long order_id)`
Called by the DiningRoom when an order is delivered to the respective table. Contains the ID of the order. When this function is called, the order is now ready to be paid by the customer.

2.3.1.2. Events

Events that can be subscribed by other nodes to signal that a change in an order has occurred. These events have associated delegate functions shown below:

- `delegate void OrderReadyEventHandler(long order_id, uint table_n)`
- `delegate void NewOrderEventHandler(Order order);`

The events that are public in the network are the following:

- `event OrderReadyEventHandler OrderReadyEvent`
Event subscribed by the DiningRoom to know when an order is fully ready, meaning both Kitchen and Bar have finished preparing.
- `event NewOrderEventHandler NewDishesOrderEvent`
Event subscribed by the Kitchen controller to know when a new order was created and must be prepared.
- `event NewOrderEventHandler NewDrinksOrderEvent`
Event subscribed by the Bar controller to know when a new order was created and must be prepared.

2.3.2. Event Repeater

Virtual node that merely serves to hide the DiningRoom and KitchenBar implementations from the PaymentZone, while still allowing these to subscribe to the PaymentZone events. This node is the one that contains the delegate functions that are called by the PaymentZone events. These functions sole purpose is to call the respective callback in either the DiningRoom or the KitchenBar.


```

public abstract class EventRepeaterDelegateObject: MarshalByRefObject {
    public void NewOrderCallback(Order order) {
        this.NewOrderCallbackCore(order);
    }
    public void OrderReadyCallback(long order_id, uint table_n) {
        this.OrderReadyCallbackCore(order_id, table_n);
    }
    protected abstract void NewOrderCallbackCore(Order order);
    protected abstract void OrderReadyCallbackCore(long order_id, uint table_n);
}

```

Class that is known by the PaymentZone. It merely serves as a callback to other functions that are to be implemented by subclasses. When PaymentZone fires its events, the respective *Callback* function is called.

```

public class EventRepeaterDelegate: EventRepeaterDelegateObject {
    OrderReadyEventHandler ready_handler;
    NewOrderEventHandler new_handler;

    public EventRepeaterDelegate (OrderReadyEventHandler ready_handler,
    NewOrderEventHandler new_handler) {
        this.ready_handler = ready_handler;
        this.new_handler = new_handler;
    }
    protected override void NewOrderCallbackCore(Order order) {
        if (this.new_handler != null) {
            this.new_handler(order);
        }
    }
    protected override void OrderReadyCallbackCore(long order_id, uint table_n) {
        if (this.ready_handler != null) {
            this.ready_handler(order_id, table_n);
        }
    }
}

```

Concrete implementation of the abstract class known by PaymentZone. KitchenBar or DiningRoom create a new object of this class and then subscribe the respective *Callback* method to the respective PaymentZone event. Afterwards, as seen above, the *Callback* method will simply call the respective *CallbackCore* method, which in turn will also simply call the respective handler passed by KitchenBar or DiningRoom.

3. User Interface

Module of classes associated with the direct interaction with the user. In this module there is an interface for each of the controllers. The GUI was developed using Gtk 2.0 and the Glade Interface Designer. By using cross-platform frameworks we ensure that our projects can run on multiple types of platforms.

3.1. Dining Room Window

Dining Room Window is the window that is mostly used by the waiters to create new orders for each table and know when a previously created order is now ready to be delivered. Below is a screenshot of this interface:



Fig. 3 - Interface of the Dining Room

3.1.1 Product List and Current Order Frame

Frame containing the list of the products the restaurant has to offer or the products the user has already inserted in its order. In case the list gets too long there there is a Gtk.ScrolledWindow inside the frame to allow scrolling, in order to view overflowed products. The list of products is divided between Dishes and Drinks, and each product has its name and price shown

3.1.2 Orders Ready

Frame containing a list of orders that were already sent into the Kitchen/Bar and are now ready to be picked up by the waiter and served to the respective table. Similarly to the above, it has a `Gtk.ScrolledWindow` inside the frame, to allow an overflow of orders.

3.1.3 Table Insertion

Box that allows the waiter to select the destination table of the order. Possible values are contained between [1, 9]. On order submit value is not reset!

3.1.4 Undo Button

Button that allows the waiter to undo the previous action. This action can either be adding a product to the current order or removing a product from the current order. Multiple undos are possible until the initial state is reached, an empty Current Order.

3.1.5 Apply Button

Button that submits the Current Order product list and table selection to the central computer. Once this button is clicked the order is submitted, and the current order list is cleared.

3.1.6 Miscellaneous Buttons

- 1.** Button to add another unit of the product. In the image if the user clicked button 1, another unit of 'Pizza' would be added to the current order list.
- 2.** Button to remove a unit of a product. In the image if the user clicked button 2, a unit of 'Pizza' would be removed from the current order.
- 3.** Button to signal that an order has been picked from the Kitchen/Bar and has been delivered to the respective table. After this, the order is now ready to be paid on the Payment Zone window.

3.2 Kitchen or Bar Window

Kitchen or Bar window is the window that is presented in either the kitchen or the bar of the restaurant. The windows are equal in both functionality and aspect, the only difference

between the two is the type of products that appear in them. Below is a screenshot of this interface:

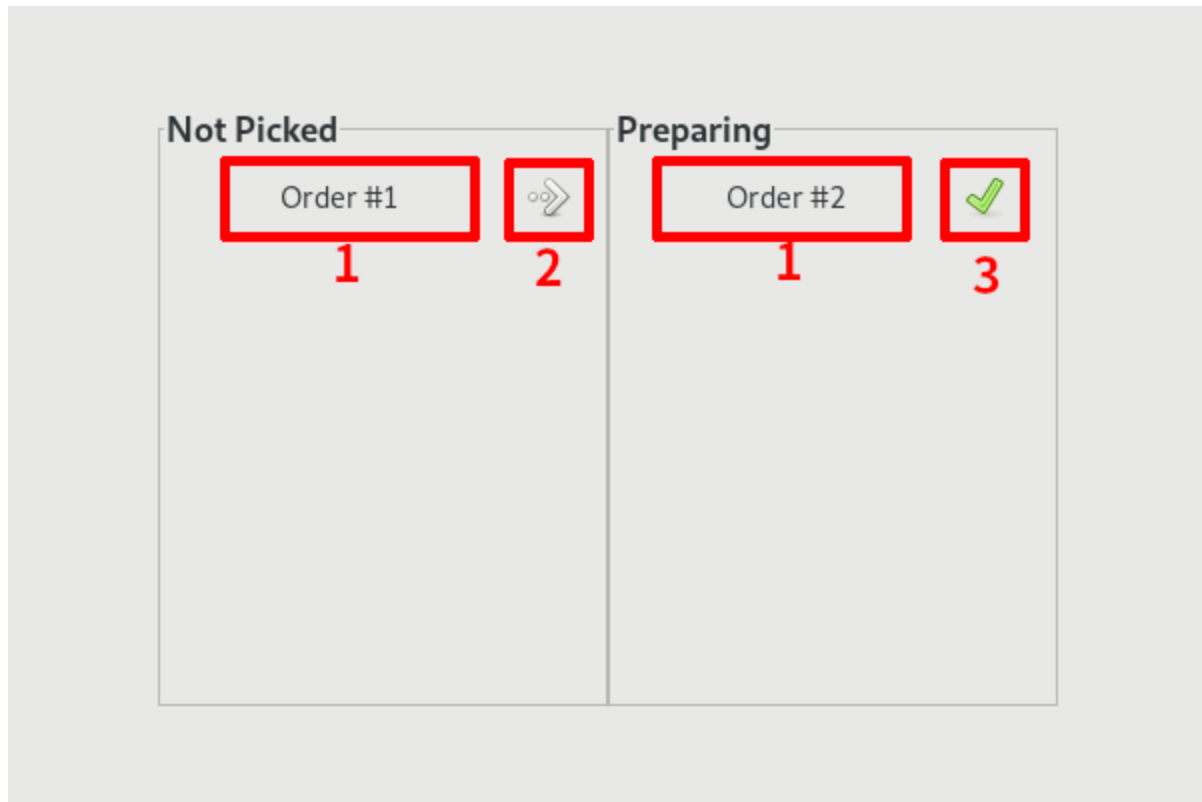


Fig. 4 - Interface of the Kitchen or Bar Window

3.2.1. Not Picked and Preparing Frame

These frames contain the orders that were not picked by a person and the orders that were already picked and are in preparation. Again, inside the frame is a `Gtk.ScrolledWindow` to allow for overflow of the list of orders.

3.2.2. Miscellaneous Buttons

- 1.** Clicking on the Order name label, will redirect the user to the Order Detail Window, where the user can see the actual products inside the order and the respective amount.
- 2.** Button to move order from the Not Picked state to the Preparing state.
- 3.** Button that signals that an order is ready and should be picked by a waiter.

3.3. Order Detail Window

Window that shows the details of the order selected by the user. In this interface the user can see the products of a given order. Remember that depending on the calling window (Kitchen or Bar), the products that the user will see will be different. If this window is called from the Kitchen Window then the user will only see the dishes. Below is a screenshot of the interface:

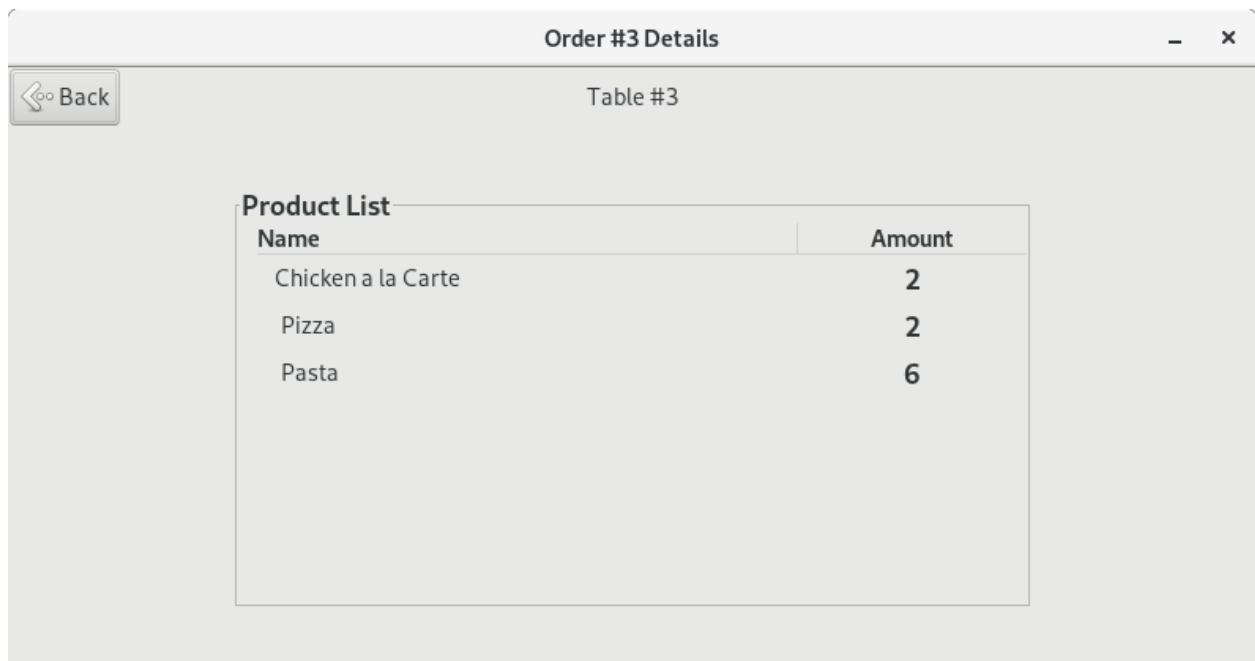


Fig. 5 - Interface of the Order Details Window

3.3.1 Back Button

Button that takes the user back to the origin window, be it Kitchen or Bar.

3.4. Payment Zone Window

Window representing the central computer of the restaurant. In here the user has access to all of the tables of the restaurant, and can see the delivered orders associated with each table. Below is a screenshot of the interface:

0.0 € Total Money

Table Selection

1	2	3
4	5	6
7	8	9

Payment

Name	Price	Amount	Total

0.0 € Order Total

See Statistics

Fig.6-Payment Zone Window on no table selected

0.0 € Total Money

Table Selection

1	2	3
4	5	6
7	8	9

Payment

Name	Price	Amount	Total
coca-cola	0.8	2	1.6€
margarita	1	5	5€
Chicken a la Carte	10.5	2	21€
Pizza	4	4	16€
Pasta	5.9	1	5.9€

49.5 € Order Total

See Statistics

Paid \$

Fig. 7 - Payment Zone Window on table #3 selected

3.4.1. Table Selection Buttons

Toggle buttons that allow the user to select which table information the user wants to see. When an Order is created for that table, a small animation will happen, where the border of the respective table will turn yellow. When an Order that was previously created for that table is now ready to be Paid, then the same animation will occur, but green instead. Toggling these buttons, changes the products seen in the Payment Frame.

3.4.2. Payment Frame

Frame that shows the products associated with a given table. This information is aggregated from all the orders of a table that were not yet paid, so there are products from multiple sources. In here we can see the name of the products, the individual price of each one, the amount of a single product that was requested, and the total price of the products. On the bottom of the frame we can see the total price of the order.

3.4.3. Paid Button

Button that is only present on non-empty table selection. Clicking on this button will signal the system that the products listed above were all paid. The system will in turn store this information, and the products of those orders will not be shown again to the user. Furthermore, the Total Money will increase with a little animation.

3.4.4. Total Money Label

Represents all of the money the restaurant has made. When this value is incremented a simple animation is played, where the textual representation of the money turns bigger and green for a while.

3.4.5. Statistics Button

Button that takes the user to the Statistics window, where it can see the global status of the Restaurant sold products.

3.5. Statistics Window

Window that contains all of the sold products. User can see most sold product, most profitable product and multiple other things.




Name  1	Price 2	Amount 3	Total 4
Pizza	4€	5	20 €
Pasta	5.9€	2	11.8 €
margarita	1€	3	3 €
coca-cola	0.8€	3	2.4 €
Chicken a la Carte	10.5€	3	31.5 €

Fig. 8 - Statistics Window with all sold products of the restaurant

3.5.1. Sorting Buttons

Buttons that enable the user to sort the respective information by ascending or descending order. Currently selected sorting is represented by the Up or Down arrow on the right of the button. Up arrow means descending order and Down arrow means ascending order. By clicking the buttons the currently selected order is toggled.

1. Name sorting toggle button
2. Price sorting toggle button
3. Amount sold sorting toggle button
4. Total money sold sorting toggle button

3.5.2 Back Button

Button that takes the user back into the PaymentZone Window.

4. Conclusion

We feel that we have developed a quite mature project that should be close to ready to be deployed. There is only a single bug that we have detected and are unable to fix, the steps to reproduce it are described below:

- Go to the DiningRoom Window
- Add at least 2 different products to the order
- Remove the product that is listed first (Product that was first added)
- Add a new product that was not already in the order
- Expected: Newly added product is added below the existing product
- Got: Newly added product is added overlapping the existing product

We do not know why this happens since we have debugged this thoroughly, and as far as we can detect in our code, we are appending the entry in the correct place, however it is not doing so. It may be an internal bug in the gtk-sharp-2.0 library (although unlikely). In any case, since the scope of this project is not to measure how well we work with the GUI, but the networking aspects, we decided to ignore it, since internally the product is added correctly anyway and does not affect the remainder of the program.

In terms of work done by each member, the work was divided evenly, so:

- João Almeida: 50%
- João Mendes: 50%