FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Distribution and Integration Technologies
4th year of the Integrated Masters in Informatics and Computing Engineering (MIEIC)

Intranet Distributed Application(using .NET Remoting) Restaurant order and account system

Group elements:

Diogo Luis Rey Torres - 201506428 - up201506428@fe.up.pt
Rui Emanuel Cabral de Almeida Quaresma - 201503005 - up201503005@fe.up.pt

April 26, 2019

Index

1.Introduction

The developed project consists of an intranet distributed application capable of automatize the restaurant needs. It was developed using the C# Language more specifically .NET Remoting and WinForms was used in the GUI.

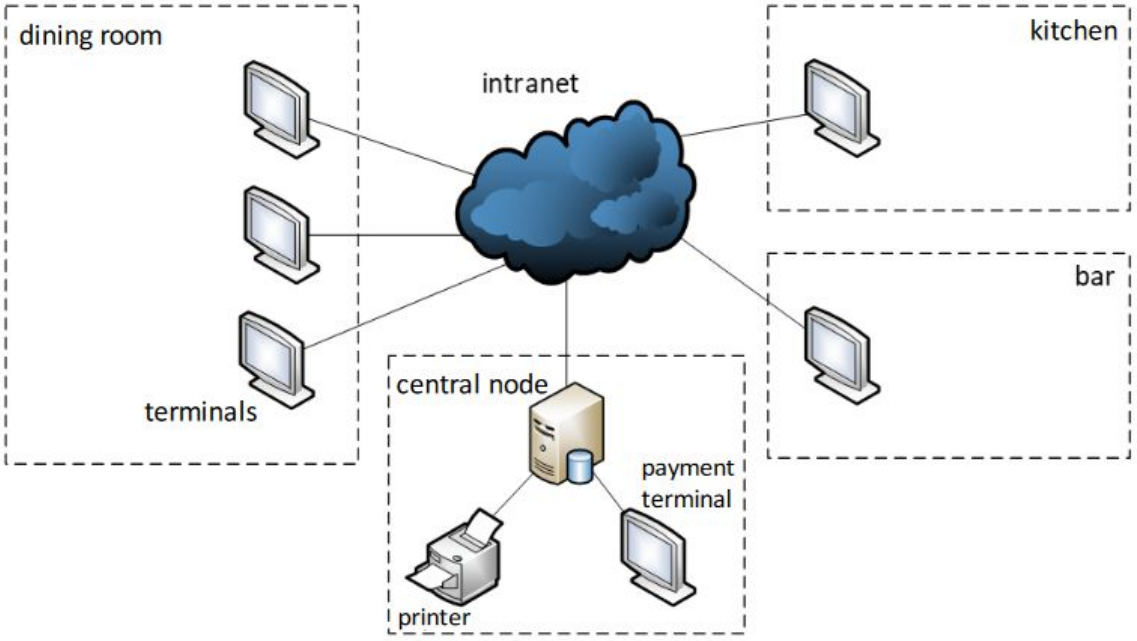
A restaurant needs to automatize the dining room orders, allowing them to be quickly communicated to the kitchen and bar tends, to be prepared as soon as possible. Also, it was necessary to maintain a complete record of all orders, compute the bills of each table and maintain a record of the total amount received in the day. With this in mind, the application implements all of these features in an intuitive and pleasant way for the final user.

In the next sections, the application will be described in finer detail, namely its architecture, implementation details and relevant issues. This report concludes with a small reflection about the work done and possible future enhancements.

2. Architecture

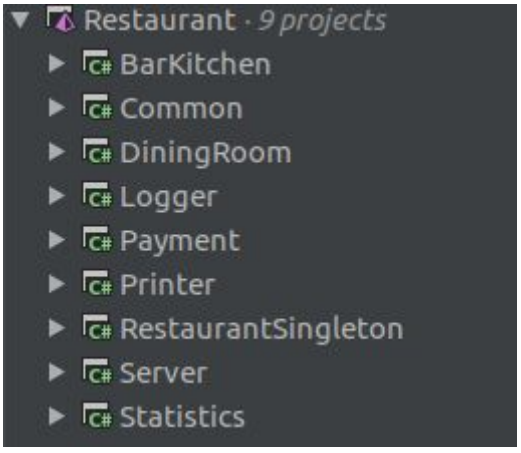
2.1 Description

This application is based on a Client-Server architecture that allows clients and servers communicate over a computer network, where the server shares their resources with clients. The sharing of information between server and clients is made through events where the clients subscribe to the events they are interested in, and the server propagates those changes. Also, the clients are able to make requests to the server through an interface given by the host.



Subtitle 1: System Architecture

The used architecture consists of five components representing the main services offered by the application, of which: a payment area (which has 3 applications - Payment App, Printer App and Statistics App), a Dining Room (DiningRoom App), a preparation area (which has a BarKitchen App), a component dedicated to register the main actions in the system (Logger) and the server itself.

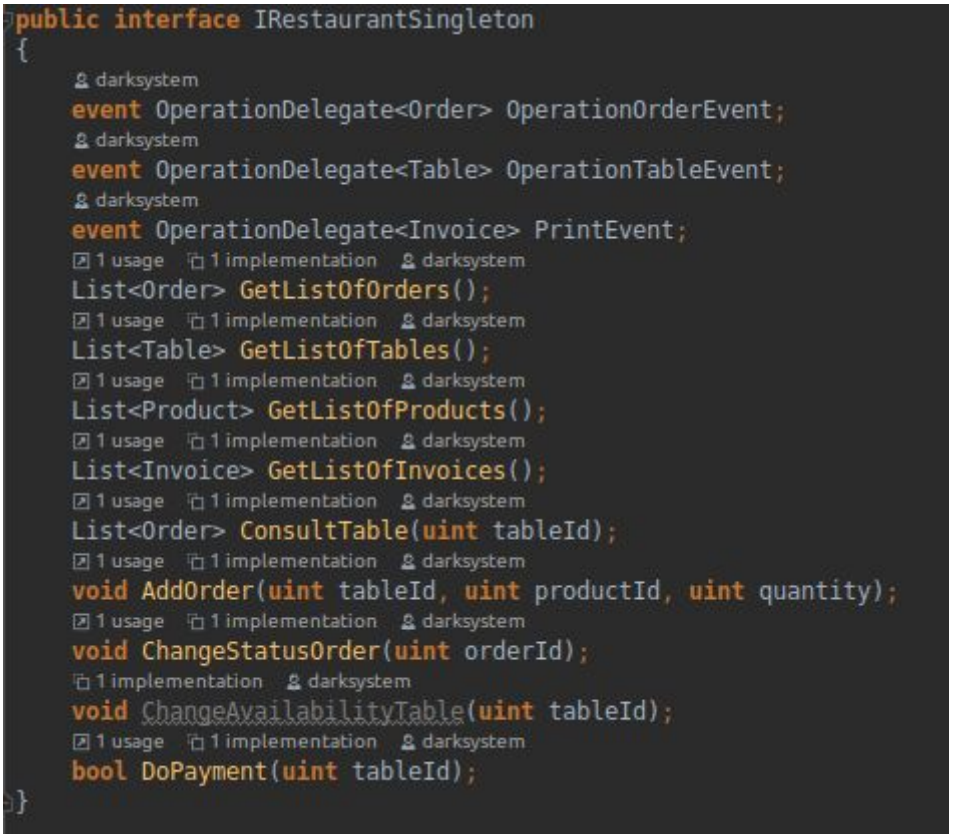


Subtitle 2: Solution Overview

2.2 Remote Objects

The remote object in this architecture is called *RestaurantSingleton* that implements methods to handle the orders, tables and products but also trigger the correct events to maintain the information updated, in each terminal.

This remote object provides an interface for the client to interact with the server.



Subtitle 3: Remote Object Interface

2.2.1 Methods

The server keeps a variable with the next order Id to maintain an incremental and unique id's. As well as maintaining four lists, one for orders, one for tables, one for the products and other for the invoices. Also, the remote object is responsible for initialize each variable with the correct values.

As described in the image of the previous section, the remote object provides a lot of methods to the client but the most important are: *AddOrder*, *ChangeStatusOrder* and *DoPayment* methods.

The *AddOrder* function adds the order to the List of Orders, sets the table as occupied, if the table is available, and notifies all the clients about the new order and whether the availability of the table has changed.

The *ChangeStatusOrder* function updates the order to the next state (*NotPicked, InPreparation, Ready, Delivered, Paid*).

Finally, the *DoPayment* function receives a table Id and check if all orders have already been delivered, if so, for each order change the state to paid and notify the clients. Then changes the state of the table to available, sends a notification to the printer with the invoice and remove the paid orders from the orders list.

2.2.2 Events

Events in .NET are based on the delegate model. So, a delegate was created in order to pass methods as arguments to other methods. This allows the events handlers, which are methods, to be invoked through these delegates.

```
public delegate void OperationDelegate<in T>(Operation op, T obj);
```

Subtitle 4: Generic Delegate

The *OperationDelegate* receives an object of the type T, which in this case can be an order, a table or an invoice, but also receives an operation that allows the subscriber to know if the object is new, has been changed, or has been removed.

```
darksystem
public event OperationDelegate<Order> OperationOrderEvent;
darksystem
public event OperationDelegate<Table> OperationTableEvent;
darksystem
public event OperationDelegate<Invoice> PrintEvent;
```

Subtitle 5: Events used

The *OperationOrderEvent* is responsible for announcing whenever new orders are created or they are changed.

The *OperationTableEvent* is responsible for announcing when the availability of a table changes.

The *PrintEvent* is responsible for transmitting the invoice to the printer when a table is closed (all orders are paid) and to update the statistics.

Finally, to notify the clients is called the *NotifyClients* function that receives an event of the Type T, the operation performed on the object and the object itself. This function invokes for each subscriber the corresponding handler and if an error is returned, the subscriber is removed from that event.

```
5 usages  darksystem
private static void NotifyClients<T>(OperationDelegate<T> operationDelegate, Operation op, T obj)
{
    if (operationDelegate == null) return;

    Delegate[] invkList = operationDelegate.GetInvocationList();

    foreach (OperationDelegate<T> handler in invkList)
    {
        new Thread( start: () =>
        {
            try
            {
                handler(op, obj);
                Console.WriteLine( value: "Invoking event handler");
            }
            catch (Exception)
            {
                operationDelegate -= handler;
                Console.WriteLine( value: "Exception: Removed an event handler");
            }
        }).Start();
    }
}
```

Subtitle 6: NotifyClients method

2.2.3 Subscribers

Each client connects to the remote object knowing only its interface, using the *RemoteNew*.*New* method that invokes the *RemotingServices.Connect* function to make that connection as shown in the images below.

```
RemotingConfiguration.Configure(remotingConfiguration,  ensureSecurity: false);
_restaurantServer = (IRestaurantSingleton) RemoteNew.New( type: typeof(IRestaurantSingleton));
```

Subtitle 7: Remoting configuration and client connects to the server

```
public static class RemoteNew
{
    private static Hashtable _types;

    1 usage   darksystem
    private static void InitTypeTable()
    {
        _types = new Hashtable();
        foreach (WellKnownClientTypeEntry entry in RemotingConfiguration.GetRegisteredWellKnownClientTypes())
            _types.Add(entry.ObjectType, entry);
    }

    1 usage   darksystem
    public static object New(Type type)
    {
        if (_types == null)
            InitTypeTable();
        WellKnownClientTypeEntry entry = (WellKnownClientTypeEntry) _types[type];
        if (entry == null)
            throw new RemotingException( message: "Type not found!");
        return RemotingServices.Connect(type, entry.ObjectUrl);
    }
}
```

Subtitle 8: RemoteNew Class

After this connection, each subscriber creates an *OperationEventRepeater* object of the type T and adds a method to the *OperationEvent* that will process the received notifications in the appropriate way. The subscriber then subscribes to the desired events of the Remote Object by passing the *Repeater* function, which will allow the Remote Object to invoke the function added to the *OperationEvent* when notifying the clients of this event.

Also, each client can invoke directly the remote object methods (e.g. Create an order) which will trigger an event that will be broadcasted to all the clients.

```
26 usages   darksystem
public class OperationEventRepeater<T> : MarshalByRefObject
{
    darksystem
    public event OperationDelegate<T> OperationEvent;

    darksystem
    public override object InitializeLifetimeService()
    {
        return null;
    }

    25 usages   darksystem
    public void Repeater(Operation op, T obj)
    {
        OperationEvent?.Invoke(op, obj);
    }
}
```

Subtitle 9: OperationEventRepeater Class

2.2.3.1 BarKitchen

This application only subscribes the *OperationOrderEvent* event in order to receive the update of the new orders. And, calls the remote method *ChangeStatusOrder* to update the orders between the *NotPicked, InPreparation* and *Ready* State.

2.2.3.2 DiningRoom

This application subscribes the *OperationOrderEvent* event and the *OperationTableEvent* event in order to receive the update of the state of the orders and the availability of the tables.

And, calls two remote methods the *ConsultTable* and the *AddOrder.* The first one, receives from the Remote Object the orders of a specific table that have not yet been paid. The other adds an order in the system and broadcast that information.

2.2.3.3 Logger

This application subscribes the *OperationOrderEvent* event and the *OperationTableEvent* event in order to receive the information about new orders, the updates of the state of the orders and the availability of the tables. Then, print them to the console.

2.2.3.4 Payment

This application subscribes the *OperationOrderEvent* event and the *OperationTableEvent* event in order to receive the new orders, the updates of the state of the orders and the availability of the tables.

And, call the remote method *DoPayment* to pay the bill of a specific table, print an invoice and make the table available (broadcasting that information to all the clients).

2.2.3.5 Printer

This application only subscribes the *PrintEvent* event in order to receive an invoice and print it to the console.

2.2.3.6 Statistics

This application only subscribes the *PrintEvent* event in order to receives the invoices created. When an invoice is received, some statistics is calculated like total number of invoices, total sum of the day and the quantity of each product sold.

Note: In subscribers who have a GUI there are also events to update the information in the graphical interface in real time.

3. Libraries

3.1. Common

The common library contains all the data structure used by the main areas of the restaurant. This library contains the following data classes: Table, Order, Product and Invoice. As well, it contains several enums related with the product type or the events. And finally, it also contains the *OperationDelegate* delegate, the interface of the remote object and class *OperationEventRepeater*.

3.2. Abstract Controller

The abstract controller was created to avoid code repetition on clients, thus containing the most common methods and variables used by them and inherited when the client controllers are created.

4.Functionalities included

- Keep a record of all orders
- Compute the bills of each table
- Keep a record of the total amount received in the day
- Each table has an Available label, that is removed once an order is added to it. When those orders are paid the label is added again
- Create new orders in a dining room table
- Open a new table
- Add to the orders already assigned to a table
- When an order is created it is displayed in the terminal of the kitchen or bar (respectively if the order is dish or drink)
- Make the orders pass from the state “Not picked” to the state “In Preparation” in the kitchen or in the bar
- Make the orders pass from the state “In Preparation” to the state “Ready” in the kitchen or in the bar
- After an order becomes “Ready” it is available to be delivered
- In a client table which has orders, each order has its state being shown and updated live, as well as colors corresponding to each state (Light Salmon - “Not Picked”, Yellow - “In Preparation”, Light Green - “Ready” and Cyan - “Delivered”).
- Once all the orders of a table are in the state “Delivered”, the waiter may press the button “Pay and Print” in the Payment terminal for that table. That table’s orders become “Paid” and the table becomes “Available”. An invoice is printed in the Printer. The Statistics terminal is updated.
- The Statistics terminal displays the total number of invoices generated so far, the total amount paid in each day and a list of each dish/drink ordered and the number of times it has been ordered.

Extra functionalities:

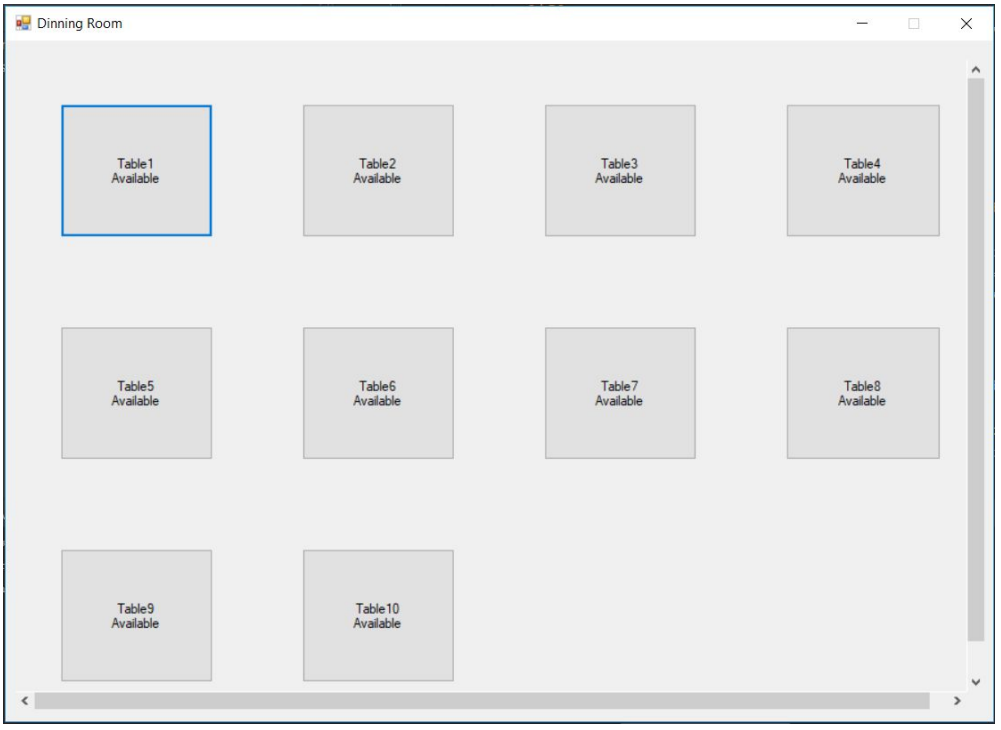
- When a table has orders in the state “Ready” a label “Deliverables” is added to it, signaling the waiters that there are orders to be delivered in that table.

- There is a Logger which writes to the console the events related with orders and tables that happen.

All requested features have been implemented as well as some extra features.

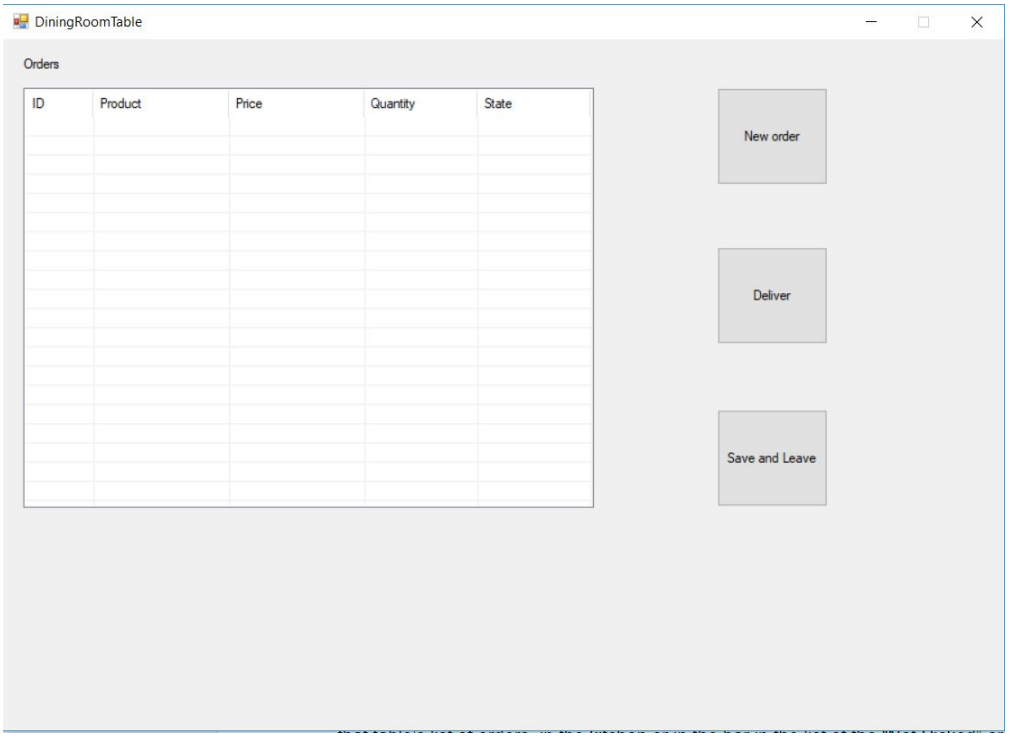
5.Screen Captures illustrating the main sequences of use

The App flow starts by running the server and then executing all the existing Apps (*BarKitchen, DiningRoom, Payment, Logger, Printer and Statistics*). After that the user may start working.



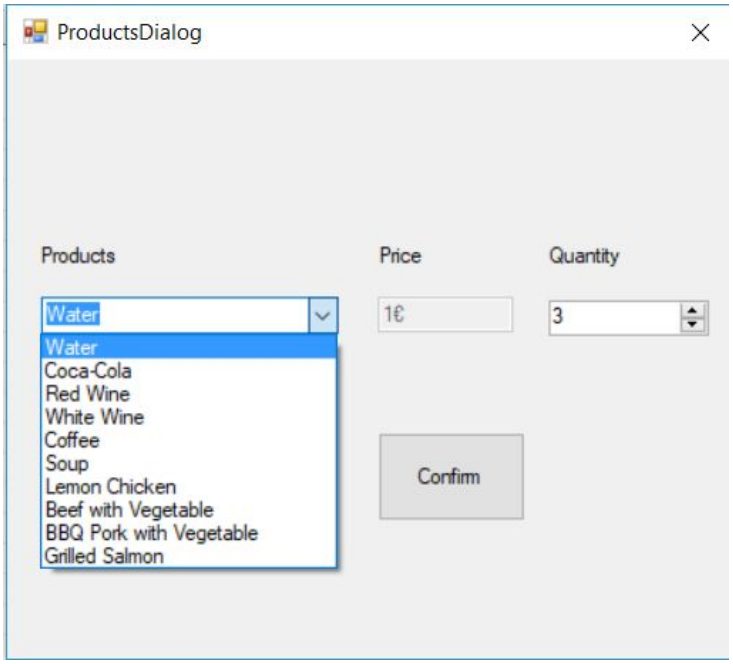
Subtitle 10: DiningRoom App

In the *DiningRoom* window the existing tables are shown as buttons.



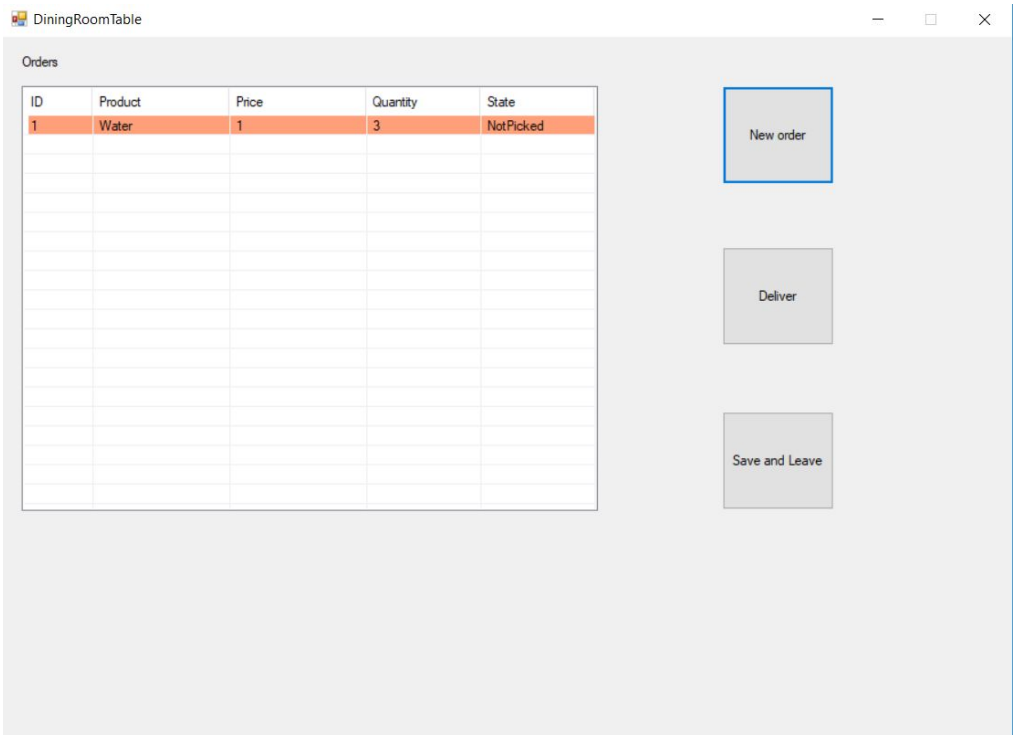
Subtitle 11: DiningRoomTable (DiningRoom App)

Clicking on one of the buttons opens that *DiningRoomTable* window. In this the waiter clicks on “New Order”, to create an order which will have the “Not Picked” state.

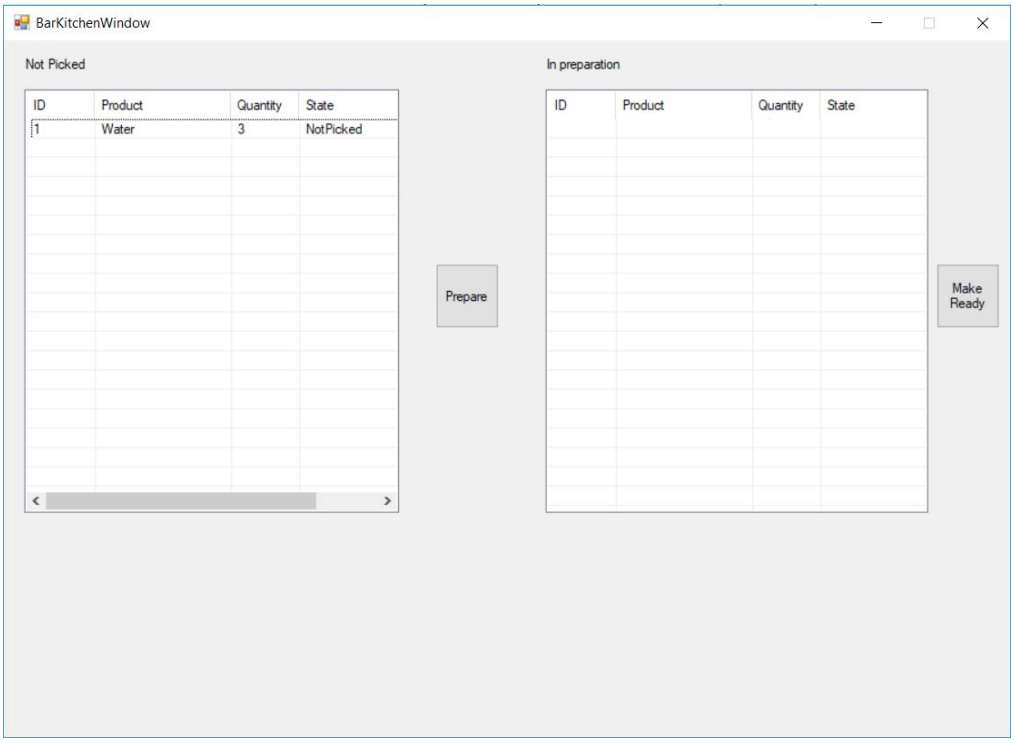


Subtitle 12: ProductsDialog (DiningRoom App)

The *ProductsDialog* window is shown, containing a list with all the products available. The waiter selects a product and sets a quantity and then presses “Confirm”.

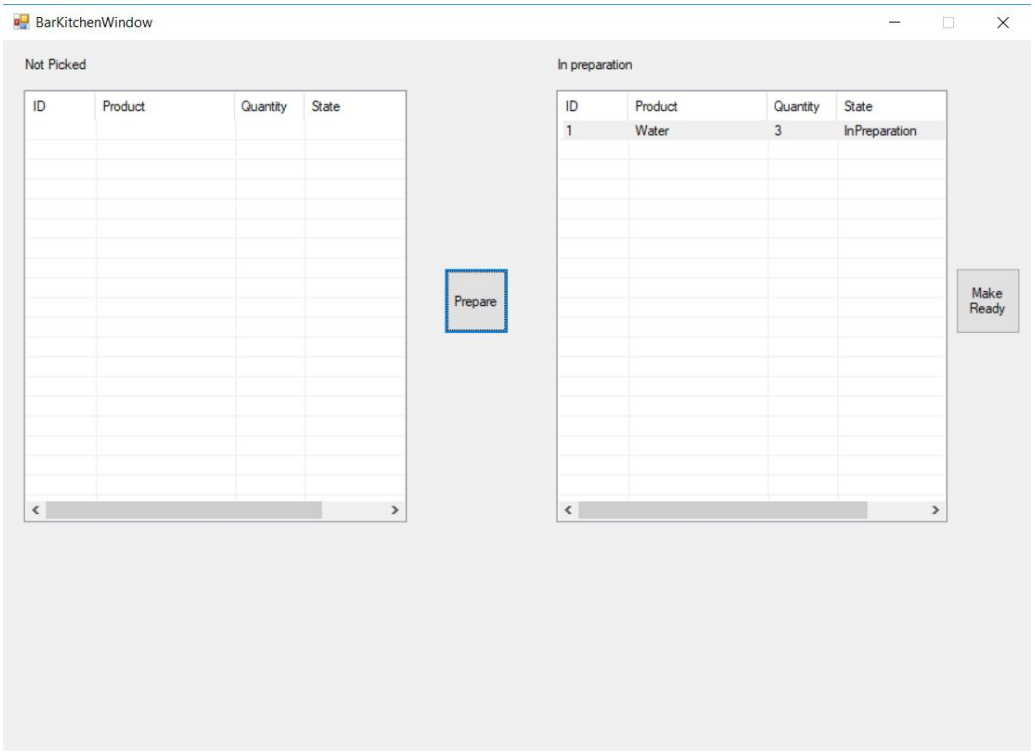


Subtitle 13: DiningRoomTable (DiningRoom App)

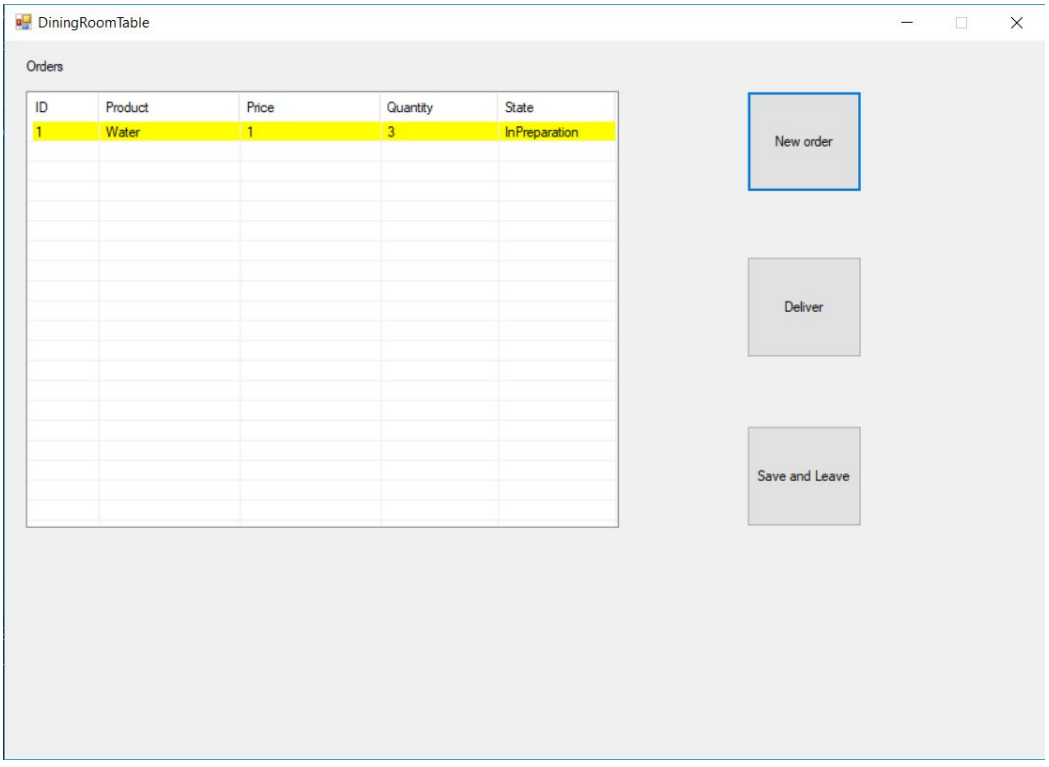


Subtitle 14: BarKitchenWindow (BarKitchen App)

This makes the order appear in that table's list of orders, in the kitchen or in the bar in the list of the "Not Picked" orders.

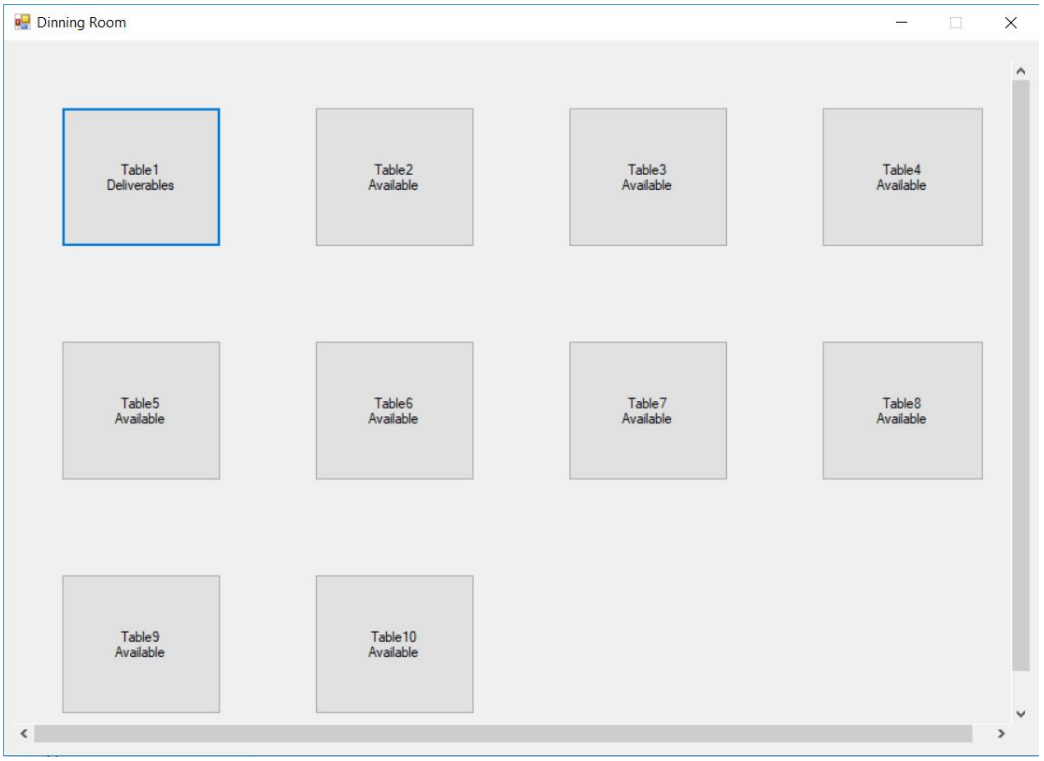


Subtitle 15: BarKitchenWindow (BarKitchen App)



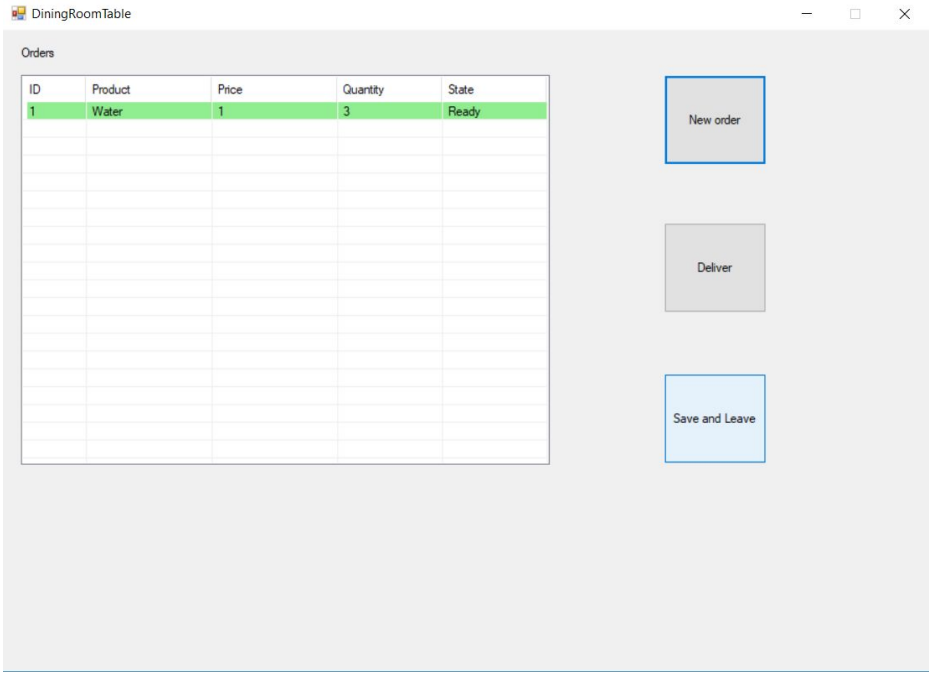
Subtitle 16: DiningRoomTable (DiningRoom App)

In the kitchen/bar the cook/bartender can pass a “Not Picked” order to “In Preparation” by selecting an order and pressing “Prepare”. After that the cook/bartender can pass an “In Preparation” order to “Ready” by selecting an order and pressing “Make Ready”.



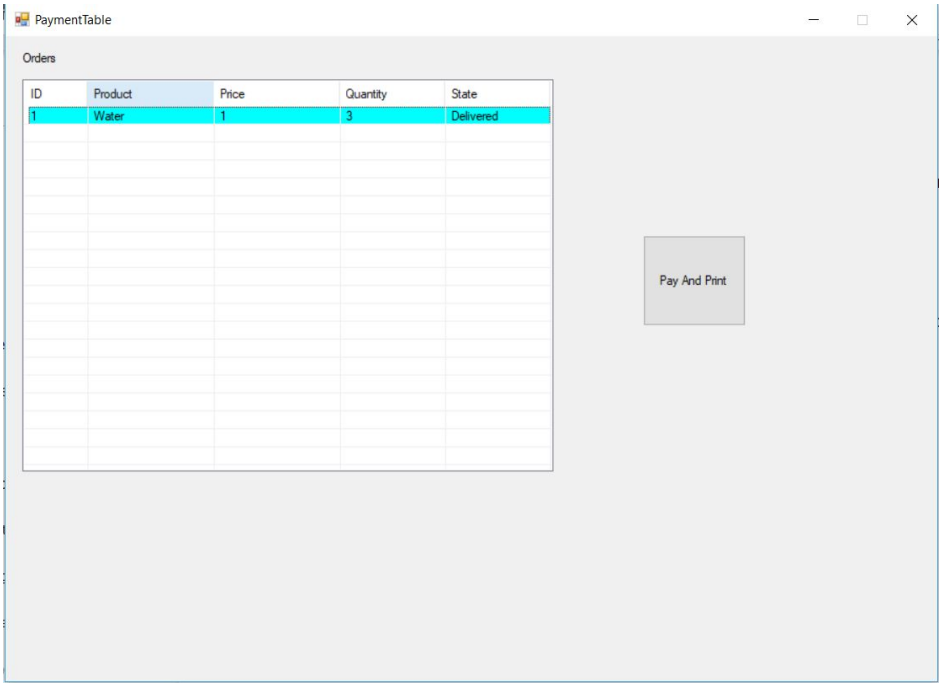
Subtitle 17: DiningRoom App

This makes the table have a “Deliverables” label in the *DiningRoom* window.



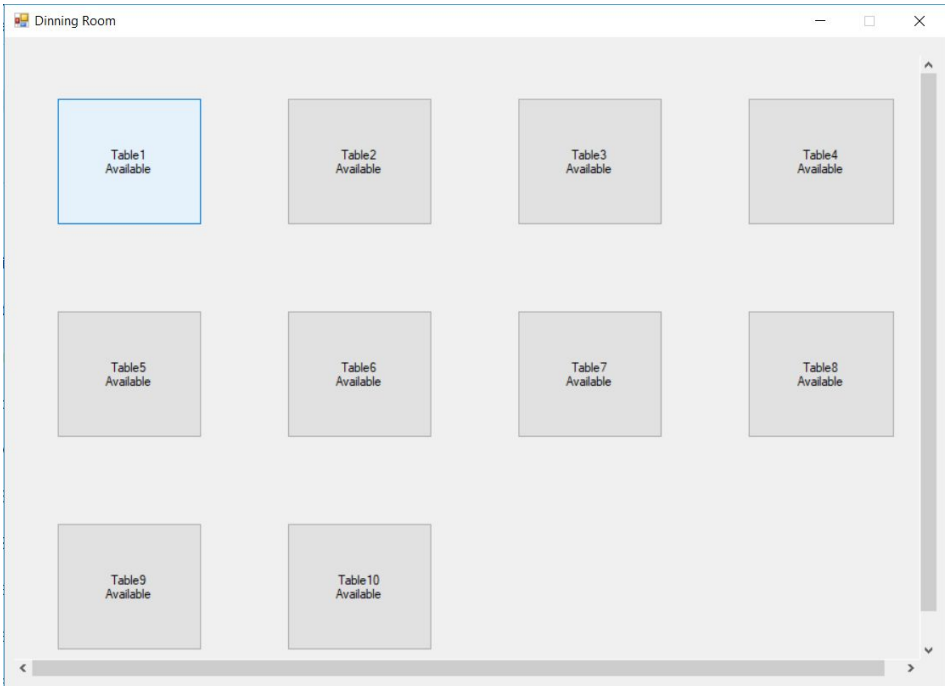
Subtitle 18: DiningRoomTable (BarKitchen App)

The waiter may select, in the *DiningRoomTable* window, the orders that are “Ready” and make them “Delivered” by pressing Deliver.



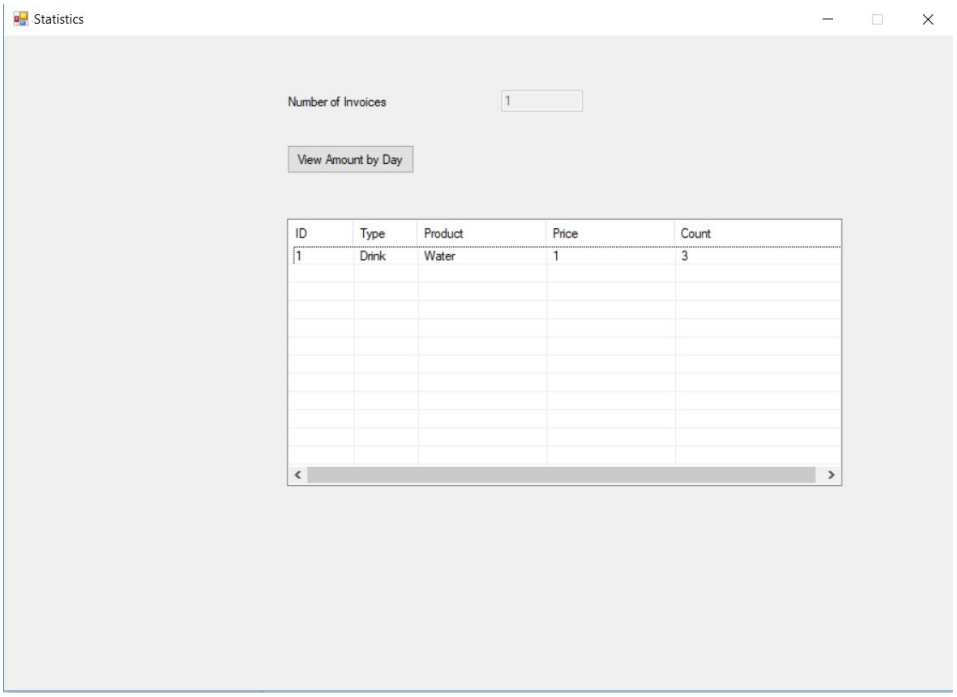
Subtitle 19: PaymentTable (Payment App)

Once all the orders are in the “Delivered” staten and the client wants to pay. the waiter goes to the *PaymentTable* window corresponding to that table and presses “Pay and Print”.

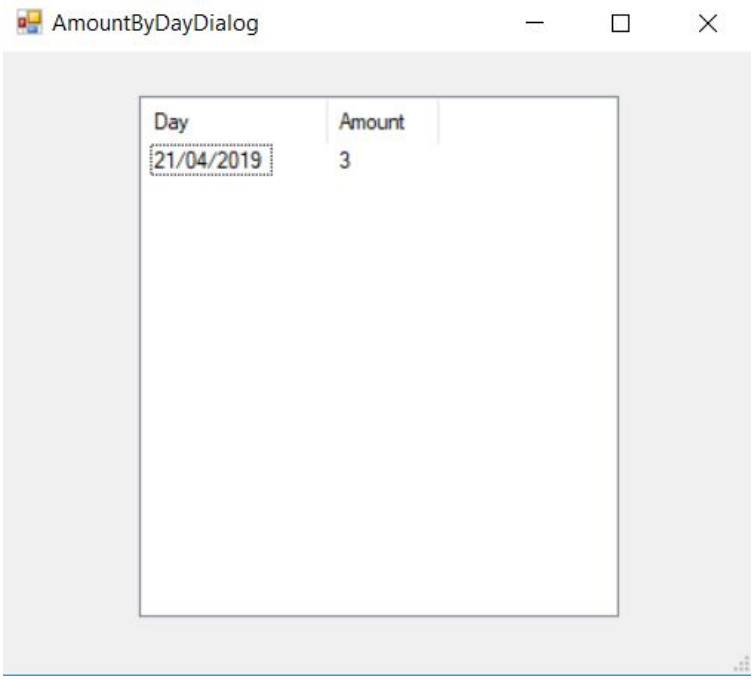


Subtitle 20: DiningRoom App

The table becomes “Available” and the *Statistics* window is updated.



Subtitle 21: Statistics App



Subtitle 22: AmountByDayDialog (Statistics App)

Checking the *Statistics* window the user can see the total invoices generated, the products quantity and open a dialog that shows the user the total amount by day.

Also in console mode the user can check the logger.

6.Conclusion

With this work, the group realized the importance of a well structured architecture in the development process in order to achieve a good final result. As we learn the basic operation of applications developed by .NET Remoting framework, as well as the main problems associated with them (subscription events, registration and instantiation of remote objects).

The final project provides a set of applications that can interact with each other in order to create an intranet distributed application capable of automatize the restaurant needs. Despite this, as always, there can be enhancements done so as to further elevate its quality and performance. In this case, we can develop automated tests to verify that all functionalities are operational. As well as allowing the restaurant customer to pay only a few orders of a given table, allowing split the bill between the elements.

7.Bibliography

- .NET Documentation (events)
  - https://docs.microsoft.com/en-us/dotnet/standard/events/
- .NET Documentation (delegates)
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/
- C# Documentation
  - https://docs.microsoft.com/en-us/dotnet/csharp/
- Client-Server Architecture
  - https://en.wikipedia.org/wiki/Client%E2%80%93server_model
- Event-driven Architecture
  - https://en.wikipedia.org/wiki/Event-driven_architecture
- Page of the curricular unit - TDIN
  - https://paginas.fe.up.pt/~apm/TDIN/

8. Resources

8.1. Software Used

- *Rider* provided by *Jetbrains*
- *Visual Studio 2019* provided by *Microsoft*