



Aplicação Distribuída em Intranet (.NET Remoting) para Sistema de Restauração de pedidos e pagamentos

Relatório 1º Projeto

Tecnologias de Distribuição e Integração

2018/2019

Mestrado Integrado em Engenharia Informática e
Computação

Francisco Maria Fernandes Machado Santos - up201607928@fe.up.pt
Leonardo Manuel Gomes Teixeira - up201502848@fe.up.pt

27 de Abril de 2019

Conteúdo

1	Introdução	3
1.1	Descrição informal do sistema	3
1.2	Cenários e Casos de Uso	3
1.3	Arquitetura do Sistema	4
2	Instâncias da Solução	5
2.1	Aplicações com GUI Windows Forms	5
2.2	Aplicações de Consola	5
2.3	Namespaces de suporte	5
3	Detalhes de Implementação	6
4	Interface Gráfica	9
4.1	Pedido	9
4.2	Concepção	10
4.3	Pagamento	11
4.4	Servidor	13

1 Introdução

1.1 Descrição informal do sistema

Este trabalho consiste no desenvolvimento de um sistema distribuído para a simulação de um fluxo básico de restauração, gerindo os pedidos passando pela confecção, entrega e pagamento dos mesmos. O sistema é capaz de guardar registo de todas as ordens efetuadas, assim como registar a conta de cada mesa e a receita total ao final de um dia.

1.2 Cenários e Casos de Uso

A aplicação consiste em vários programas que interagem remotamente entre si. Em qualquer das instâncias de *DiningRoomTerminal* é possível efetuar novos pedidos, abrir uma nova mesa ou adicionar pedidos a uma mesa já aberta. Estes pedidos são direcionados ao *KitchenTerminal* ou *BarTerminal*, e este será preparado por um bartender ou cozinheiro (dependendo do tipo de pedido) que esteja livre, alterando o estado do pedido para "em preparação", a fim de evitar que este seja preparado mais do que uma vez. Quando um pedido está pronto, é alterado o estado do pedido para "pronto" e os *DiningRoomTerminal* são notificados (*Event*) para que o mesmo possa ser levado à mesa. Quando se pretende emitir a conta para uma mesa, é no *PaymentTerminal* que se fecha a mesa e que se procede o pagamento dos pedidos efetuados. Este terminal tem um *Printer* associado, no qual deve ser impressa uma fatura contendo a lista de pedidos para essa tabela e cada respectivo preço e quantidade, e o valor total a pagar.

1.3 Arquitetura do Sistema

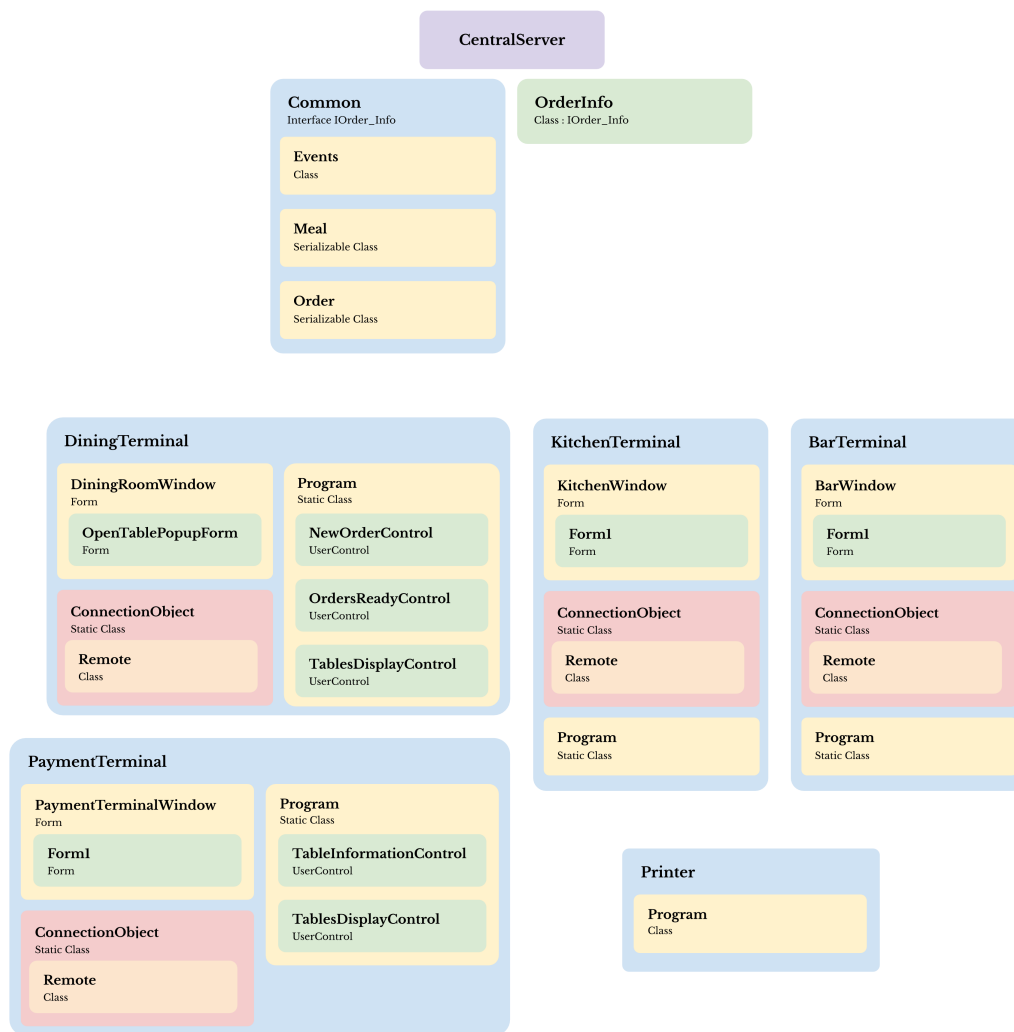


Figura 1: Arquitetura do sistema

2 Instâncias da Solução

2.1 Aplicações com GUI Windows Forms

- DiningRoomTerminal:
Terminal usado para abrir novas mesas, efetuar novos pedidos, verificar o estado dos mesmo e encaminhá-los quando estiverem prontos.
- KitchenTerminal:
Terminal usado para receber novos pedidos do restaurante, proceder à sua concepção e alterar o estado para 'pronto' quando estiverem prontos.
- BarTerminal:
Terminal usado para receber novos pedidos do bar, proceder à sua concepção e alterar o estado para 'pronto' quando estiverem prontos.
- PaymentTerminal:
Terminal usado para registrar o pagamento de um pedido e fechar a mesa correspondente.

2.2 Aplicações de Consola

- CentralServer:
Este programa é responsável por conter toda a informação do sistema, e fazer a conexão entre os Terminais
- Printer:
Este programa é responsável por apresentar a fatura de pagamento do pedido completo.

2.3 Namespaces de suporte

- Common:
Este *namespace* define a Interface de comunicação entre o *CentralServer* e os Terminais. São declarados os Eventos lançados pelos Terminais, assim como as Estruturas de Dados a serem enviadas (*Meal* e *Order*)
- OrderInfo:
Este *namespace* implementa os métodos Interface de comunicação entre o *CentralServer* e os Terminais. Aqui acontece grande parte da lógica do sistema e é onde o estado do mesmo é modificado.

3 Detalhes de Implementação

Como já foi mencionado, o Sistema segue o paradigma de Arquitetura Cliente-Servidor. A Solução é dividida em aplicações de Cliente (DiningRoomTerminal, KitchenTerminal, etc.) e a aplicação de Servidor (*CentralServer*). Toda a comunicação é efetuada através de Eventos e do acesso a objetos remotos comuns. O namespace Common é responsável por definir a interface `IOrder_Info` que declara os métodos a ser chamados pelos Terminais e que são implementados na classe `OrderInfo`:

```
1 public interface IOrder_Info
2 {
3     event UpdateActiveTablesDelegate updateActiveTablesEvent;
4
5     event UpdateOrdersReadyDelegate updateOrdersReadyEvent;
6
7     event SendOrderToKitchenDelegate sendOrderToKitchenEvent;
8
9     event SendOrderToBarDelegate sendOrderToBarEvent;
10
11     event PrintMealInvoiceDelegate printMealInvoiceEvent;
12
13     void OpenTable(int tableID);
14
15     void CloseTable(int tableID);
16
17     void AddNewOrder(int tableID, Order newOrder);
18
19     Meal GetMealInformation(int tableID);
20
21     List<int> GetActiveTables();
22
23     void SetMealAsPaid(int tableID);
24
25     void UpdateKitchenOrderState(int tableID, int orderID, Order.
        ORDER_STATE state);
26
27     void UpdateBarOrderState(int tableID, int orderID, Order.
        ORDER_STATE state);
28
29
30 }
```

Common é também responsável por definir os Eventos usados na comunicação Cliente-Servidor e definir as Estruturas de Dados usadas na comunicação (Order, Meal), estas que são sincronizadas na serialização através do uso de *lock()*.

```
1 [Serializable]
2 public class Meal
3 {
4
5     private static int IDCounter = 0;
6     private static object locker = new object();
7
8     private int ID;
9     private int tableID;
10    private DateTime start;
11    private DateTime? end;
12    private List<Order> orders;
13    private double totalPrice;
14
15    public Meal(int tableID)
16    {
17        lock (locker)
18        {
19            IDCounter++;
20            ID = IDCounter;
21        }
22        this.tableID = tableID;
23        start = DateTime.Now;
24        end = null;
25        orders = new List<Order>();
26        totalPrice = 0;
27    }
28
29    // (...)
30 }

1 [Serializable]
2 public class Order
3 {
4     private int ID;
5     private string description;
6     private int destinationTable;
7     private double price;
8     private ORDER_STATE state;
9     private ORDER_TYPE type;
10
11    public enum ORDER_STATE { NOT_PICKED, IN_PREPARATION, READY,
12        PAID };
13
14    public enum ORDER_TYPE { KITCHEN, BAR }
15
16    // (...)
17
18    public Order(string description, int destinationTable, double
19        price, ORDER_TYPE type)
20    {
21        this.description = description;
22        this.destinationTable = destinationTable;
23        this.price = price;
24        state = ORDER_STATE.NOT_PICKED;
25        this.type = type;
26    }
27
28    // (...)
29 }
```

O Armazenamento encontra-se centralizado no Servidor (*CentralServer*) sendo que todas as operações de comunicação entre quaisquer elementos passam pelo mesmo. Em cada Terminal há um *CommunicationObject* responsável por efetuar a comunicação com o Servidor, sendo este um objeto estático que possibilita abstrair toda a lógica de comunicação da GUI. Segue a implementação deste objeto do BarTerminal.

```

1 namespace BarTerminal
2 {
3     static class ConnectionObject
4     {
5         static IOrder_Info centralServer;
6         static SendOrderToBarRepeater sendOrderToBarRepeater;
7
8         static ConnectionObject() {
9             Console.WriteLine("CONNECTION OBJECT CALLED");
10             RemotingConfiguration.Configure("BarTerminal.exe.config", false);
11             centralServer = (IOrder_Info)RemoteNew.New(typeof(IOrder_Info));
12         }
13
14         public static void InitConnection() {
15             Console.WriteLine("Object was Initialized");
16         }
17
18         public static void ReceiveNewOrder(Action<Order> ReceiveOrderFunc) {
19             // (...)
20         }
21
22         public static void UpdateOrderState(int tableID, int orderID, Order.ORDER_STATE state) {
23             // (...)
24         }
25     }
26 }
27
28
29 /* Mechanism for instanciating a remote object through its interface, using the config file */
30 class RemoteNew
31 {
32     private static Hashtable types = null;
33
34     private static void InitTypeTable()
35     {
36         types = new Hashtable();
37         foreach (WellKnownClientTypeEntry entry in RemotingConfiguration.GetRegisteredWellKnownClientTypes())
38             types.Add(entry.ObjectType, entry);
39     }
40
41     public static object New(Type type)
42     {
43         if (types == null)
44             InitTypeTable();
45         WellKnownClientTypeEntry entry = (WellKnownClientTypeEntry)types[type];
46         if (entry == null)
47             throw new RemotingException("Type not found!");
48         return RemotingServices.Connect(type, entry.ObjectUrl);
49     }
50 }

```


4 Interface Gráfica

4.1 Pedido

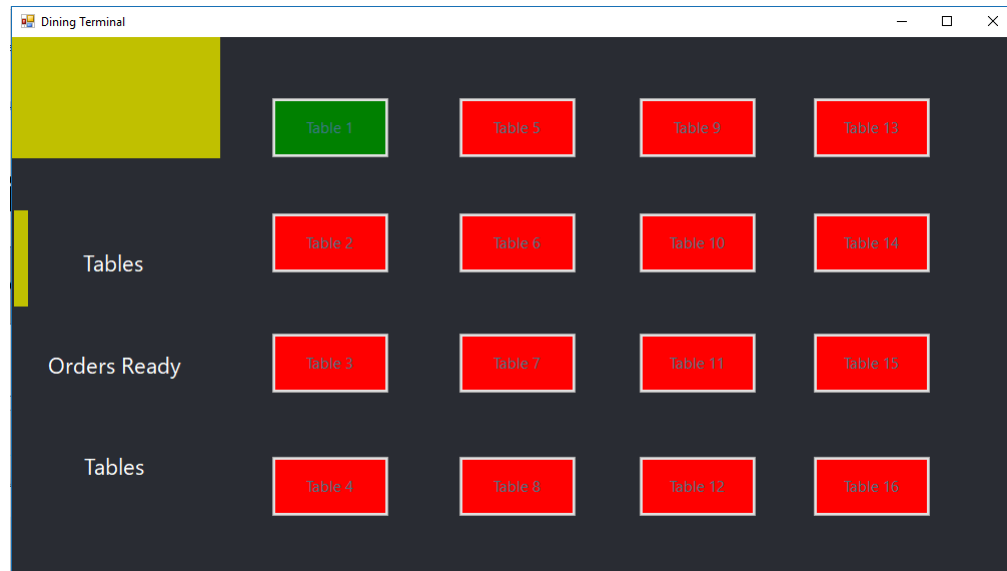


Figura 2: *DiningRoomTerminal* com a Mesa 1 aberta

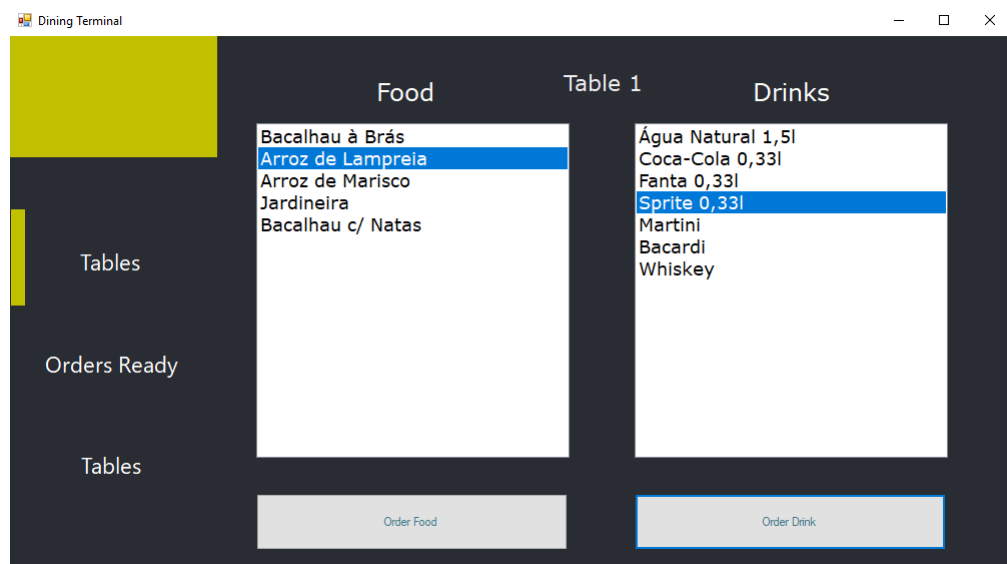


Figura 3: *DiningRoomTerminal* com a emissão dos pedidos de Arroz de Lampreia e Sprite 0.33L.

4.2 Concepção

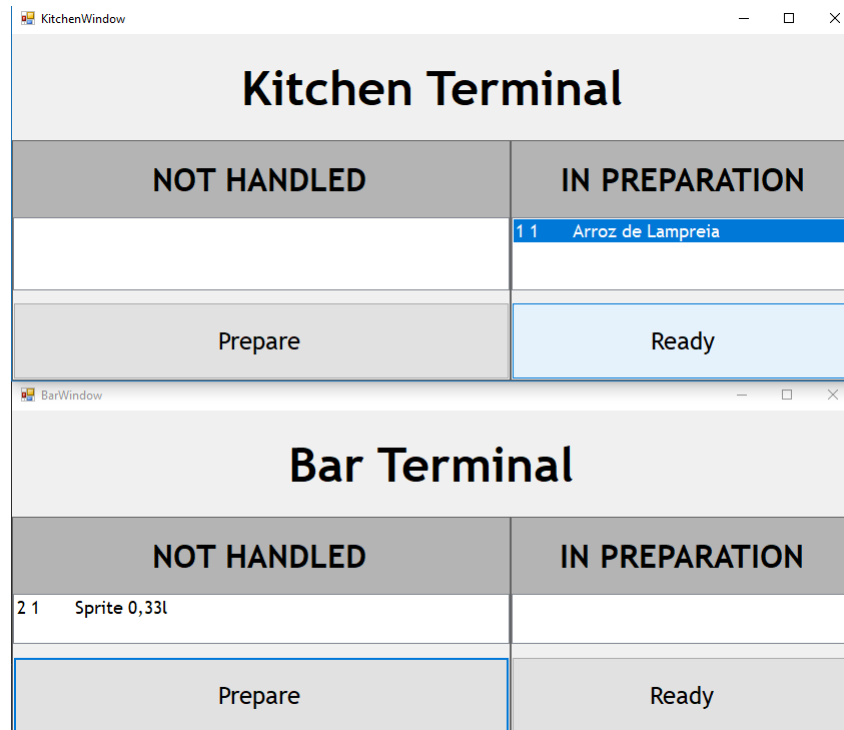


Figura 4: *KitchenTerminal* e *BarTerminal* com os pedidos de Arroz de Lampreia e Sprite 0.33L, respectivamente, no seu terminal.

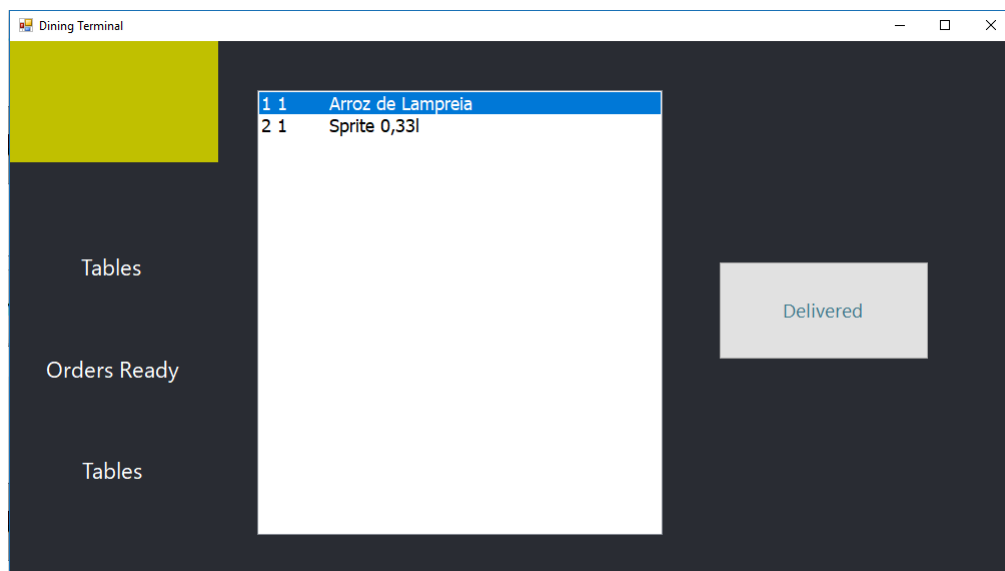


Figura 5: *DiningRoomTerminal* com a confirmação de que os pedidos estão prontos, e a possibilidade de confirmar a entrega do pedido.

4.3 Pagamento

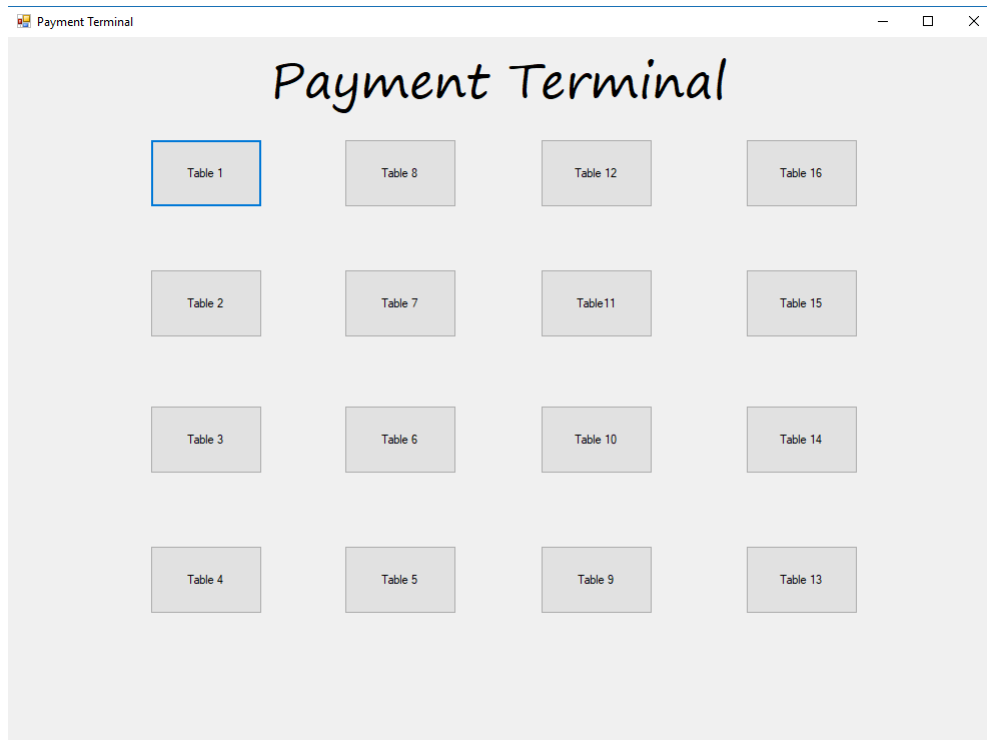


Figura 6: *PaymentTerminal* com a possibilidade de escolher a mesa a efetuar o pagamento.

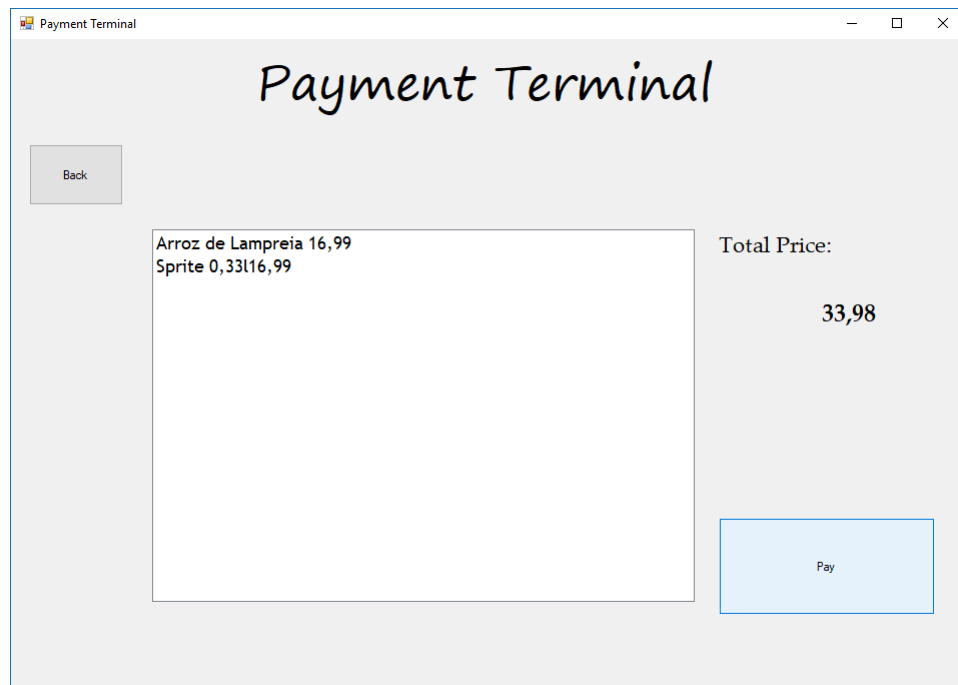


Figura 7: *PaymentTerminal* com o registo final do pedido, o valor final associado e a possibilidade de confirmar o pagamento.

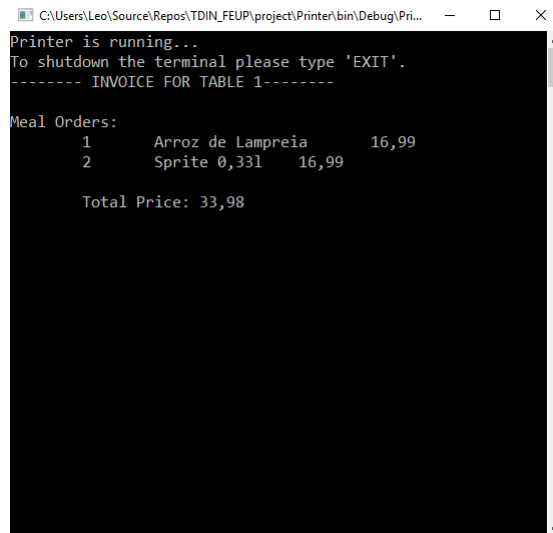
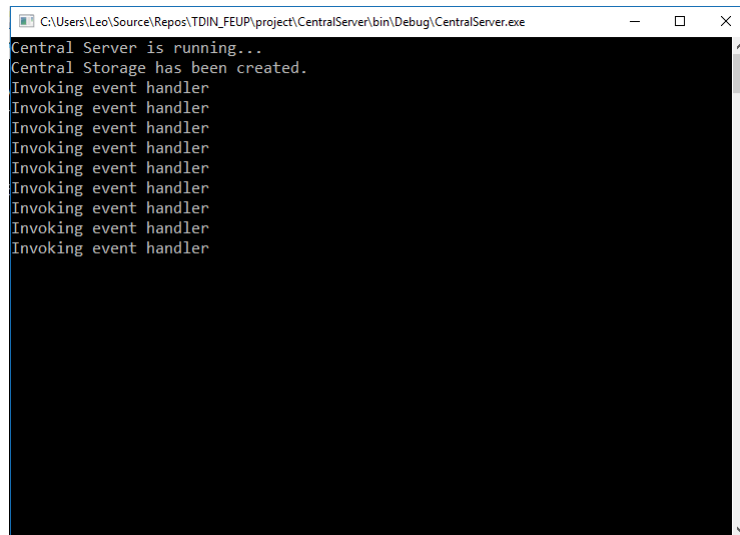


Figura 8: *Printer* após o pagamento do pedido.

4.4 Servidor



```
C:\Users\Leo\Source\Repos\TDIN_FEUP\project\CentralServer\bin\Debug\CentralServer.exe
Central Server is running...
Central Storage has been created.
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
Invoking event handler
```

Figura 9: *CentralServer* no final de todo o processamento