

## Projeto de DA

# Relatório do Projeto de Desenvolvimento de Aplicações

---

**Grupo:** PL1/2/3-X      **Docente:** Alexandre Rosário

---

**Nº** 2230240      Diogo Gouveia

**Nº** 2232357      Rafael Coelho

**Nº** 2231432      Rafael Reis

---

# Índice

<b>1. Models (Modelos)</b> .....	4
<b>2. Controllers (Controladores)</b> .....	10
<b>2.1. Controllers Details:</b> .....	11
<b>2.2. Controllers Lists:</b> .....	12
<b>3. Views</b> .....	15
<b>3.1. Views Details</b> .....	16
<b>3.2. Views Lists</b> .....	20

## Índice de Figuras

Figura 1: User Model .....	4
Figura 2: Dish Model .....	5
Figura 3: Employee Model .....	5
Figura 4: Extra Model.....	6
Figura 5: Invoice Model .....	6
Figura 6: InvoiceLine Model .....	7
Figura 7: Menu Model.....	7
Figura 8: Penalty Model .....	8
Figura 9: Professor Model.....	8
Figura 10: Reservation Model .....	9
Figura 11: Student Model .....	9
Figura 12: Client Model .....	10
Figura 13: Login .....	15
Figura 14: Main Page .....	15
Figura 15: Employee Details .....	16
Figura 16: Client Details .....	16
Figura 17: Dish Details .....	17
Figura 18: Extras Details .....	18
Figura 19: Canteen Menu Details .....	19
Figura 20: Create Reservation Details .....	19

Figura 21: List of Employees .....	20
Figura 22: List of Clients .....	21
Figura 23: List of Dishes .....	21
Figura 24: List of Extras .....	22
Figura 25: List Canteen Menus .....	23
Figura 26: List Reservations .....	24
Figura 27: List of Penalties .....	25

## 1. Models (Modelos)

O sistema é composto por um conjunto de modelos que são classes representativas de objetos. Cada modelo está associado a uma tabela da base de dados, o que facilita os processos de armazenamento, atualização, recuperação e eliminação de dados.

Para além disso, os modelos permitem estabelecer relações entre as diferentes entidades, ou seja, as várias tabelas da base de dados estão interligadas por relacionamentos através das chaves primárias e estrangeiras, como exemplo temos um cliente que pode ter diversas reservas.

De seguida, serão apresentados os diversos modelos que compõe o projeto.

- **User Model:**

*Figura 1: User Model*

```
public class User
{
    [Key]
    14 references
    public int Id { get; set; }
    27 references
    public string Name { get; set; }
    33 references
    public string NIF { get; set; }

    0 references
    public User()
    {
    }

    2 references
    public User(string name, string nif)
    {
        Name = name;
        NIF = nif;
    }

    4 references
    public override string ToString()
    {
        return $"Name: {Name}, NIF: {NIF}";
    }
}
```

- **Dish Model:**

*Figura 2: Dish Model*

```

22 references
public class Dish
{
    [Key]
    8 references
    public int Id { get; set; }
    9 references
    public DishTypeEnum Type { get; set; }
    6 references
    public string Description { get; set; }
    8 references
    public bool Active { get; set; }
    0 references
    public ICollection<Menu> Menus { get; set; }
    0 references
    public ICollection<Reservation> Reservations { get; set; }

    1 reference
    public Dish()
    {
    }

    0 references
    public Dish(DishTypeEnum dishType, bool active)
    {
        Active = active;
        Type = dishType;
    }

    0 references
    public override string ToString()
    {
        return Type + " - " + Description;
    }
}

```

- **Employee Model:**

*Figura 3: Employee Model*

```

23 references
public class Employee : User
{
    6 references
    public string Username { get; set; }

    1 reference
    public Employee()
    {
    }

    0 references
    public Employee(string name, string nif, string username) : base(name, nif)
    {
        Username = username;
    }

    1 reference
    public override string ToString()
    {
        return Name + " - " + NIF + " - " + Username;
    }
}

```

- **Extra Model:**

Figura 4: Extra Model

```
public class Extra
{
    [Key]
    4 references
    public int Id { get; set; }
    7 references
    public string Description { get; set; }
    8 references
    public float Price { get; set; }
    7 references
    public bool Active { get; set; }
    1 reference
    public ICollection<Menu> Menus { get; set; }
    2 references
    public ICollection<Reservation> Reservations { get; set; }

    2 references
    public Extra()
    {
        Menus = new List<Menu>();
        Reservations = new List<Reservation>();
        Reservations = new List<Reservation>();
    }

    0 references
    public Extra(string description, float price, bool isActive) : this()
    {
        Description = description;
        Price = price;
        Active = isActive;
    }

    0 references
    public override string ToString()
    {
        return Description + " - " + Price + " €" + (Active ? "" : " (inactive)");
    }
}
```

- **Invoice Model:**

Figura 5: Invoice Model

```
public class Invoice
{
    [Key]
    1 reference
    public int Id { get; set; }
    3 references
    public DateTime Date { get; set; }
    3 references
    public float Total { get; set; }
    4 references
    public Client Client { get; set; }
    7 references
    public ICollection<InvoiceLine> InvoiceLines { get; set; }
    0 references
    public Menu Menu { get; set; }

    1 reference
    public Invoice()
    {
        InvoiceLines = new List<InvoiceLine>();
    }

    0 references
    public Invoice(DateTime date, float total)
    {
        Date = date;
        Total = total;
        InvoiceLines = new List<InvoiceLine>();
    }

    1 reference
    public override string ToString()
    {
        return "Invoice: " + Id + " - " + Client.Name + " - " + Date + " - " + Total + " €";
    }
}
```

- **Invoice Line Model:**

Figura 6: InvoiceLine Model

```

11 references
public class InvoiceLine
{
    [Key]
    0 references
    public int Id { get; set; }
    8 references
    public float Price { get; set; }
    8 references
    public string Description { get; set; }

    2 references
    public InvoiceLine() { }

    0 references
    public InvoiceLine(int quantity, float price, string description)
    {
        Price = price;
        Description = description;
    }

    0 references
    public override string ToString()
    {
        return Description + " - " + Price + "€";
    }
}

```

- **Menu Model:**

Figura 7: Menu Model

```

28 references
public class Menu
{
    [Key]
    3 references
    public int Id { get; set; }
    19 references
    public DateTime Date { get; set; }
    9 references
    public Dish Dish { get; set; }
    7 references
    public ICollection<Extra> Extras { get; set; }
    6 references
    public int QuantityAvailable { get; set; }
    5 references
    public float PriceStudent { get; set; }
    5 references
    public float PriceProfessor { get; set; }

    2 references
    public Menu()
    {
        Extras = new List<Extra>();
    }

    0 references
    public override string ToString()
    {
        return $"{Dish.Description} - {Dish.Type} - {Date.ToString("dd/MM/yyyy HH:mm")}";
    }
}

```

- **Penalty Model:**

*Figura 8: Penalty Model*

```
✓ namespace iCantina.models
{
  16 references
  ✓ public class Penalty
  {
    [Key]
    3 references
    public int Id { get; set; }
    8 references
    public float Amount { get; set; }
    7 references
    public int Hours { get; set; }

    1 reference
    ✓ public Penalty()
    {
    }

    1 reference
    ✓ public override string ToString()
    {
      return $"Penalty: {Amount} € for {Hours} hours";
    }
  }
}
```

- **Professor Model:**

*Figura 9: Professor Model*

```
23 references
public class Professor : Client
{
  6 references
  public string Email { get; set; }

  1 reference
  public Professor()
  {
  }

  0 references
  public Professor(string name, string nif, float balance, string email) : base(name, nif, balance)
  {
    Email = email;
  }

  2 references
  public override string ToString()
  {
    return NIF + " - " + Name + " - " + Email;
  }
}
```



- **Reservation Model:**

Figura 10: Reservation Model

```

14 references
public class Reservation
{
    [Key]
    1 reference
    public int Id { get; set; }
    13 references
    public Client Client { get; set; }
    4 references
    public DateTime Date { get; set; }
    4 references
    public Dish Dish { get; set; }
    4 references
    public ICollection<Extra> Extras { get; set; }
    16 references
    public Menu Menu { get; set; }
    6 references
    public Penalty Penalty { get; set; }
    3 references
    public bool Served { get; set; }

    1 reference
    public Reservation()
    {
        Extras = new List<Extra>();
    }

    4 references
    public float GetTotal()
    {
        float total = 0;
        foreach (Extra extra in Extras)
        {
            total += extra.Price;
        }
        if (this.Client is Student)
            total += this.Menu.PriceStudent;
        else
            total += this.Menu.PriceProfessor;

        if (Penalty != null)
            total += Penalty.Amount;

        return total;
    }

    1 reference
    public override string ToString()
    {
        return $"Reservation: {(this.Client)} - {Date} - {Menu} - {Penalty}";
    }
}

```

- **Student Model:**

Figura 11: Student Model

```

namespace iCantina.models
{
    23 references
    public class Student : Client
    {
        5 references
        public int StudentNumber { get; set; }

        1 reference
        public Student() { }

        0 references
        public Student(string name, string nif, float balance, int studentNumber) : base(name, nif, balance)
        {
            StudentNumber = studentNumber;
        }

        2 references
        public override string ToString()
        {
            return NIF + " - " + Name + " - " + StudentNumber;
        }
    }
}

```

- **Client Model:**

Figura 12: Client Model

```

34 references
public class Client : User
{
    13 references
    public float Balance { get; set; }
    3 references
    public ICollection<Invoice> Invoices { get; set; }

    0 references
    public Client() {
        Invoices = new List<Invoice>();
        Balance = 0;
    }

    2 references
    public Client(string name, string nif, float balance) : base(name, nif)
    {
        Invoices = new List<Invoice>();
        Balance = balance;
    }

    3 references
    public override string ToString()
    {
        return NIF + " - " + Name + " - " + Balance;
    }
}

```

## 2. Controllers (Controladores)

O “Controller” é a base para todos os controladores que compõem o sistema. Este controlador, inicializa o context da base de dados (ICanteenContext), e reúne um conjunto de métodos que permitem salvar alterações e libertar recursos, permitindo que os acessos e operações à base de dados é efetuada de uma simples e consistente.

O “LoginController” possui um construtor destinado a exibir uma mensagem de inicialização e conta com o método “Authenticate” que procede à autenticação do funcionário através do username do funcionário que se encontra registado na base de dados. Caso a procura pelo username seja bem sucedido, retorna o objeto “employee”, caso contrário, retorna null.

Os restantes controladores encontram-se divididos em dois tipos: os controladores responsáveis por operações como criação, atualização e eliminação de registos (controllers details), e os controladores para os forms de listagem (controllers list), que permitem listar e apresentar os múltiplos registos realizados no respetivo form details.

## **2.1. Controllers Details:**

- **CanteenMenuDetailsController:**

Controlador que realiza a gestão de operações inerentes aos menus da cantina, tais como a criação, atualização e eliminação de menus e a inclusão de pratos e extras. Através deste controlador, é ainda possível filtrar os tipos de pratos e garante que não existem menus duplicados numa determinada data;

- **ClientDetailsController:**

Controlador com métodos CRUD que permitem realizar diversas operações aos detalhes dos clientes, podendo ser estudantes ou professores. Neste sentido, os métodos possibilitam a inclusão, atualização, exclusão e a listagem de múltiplos registos provenientes, tanto de estudantes, como também de professores. Para além disso, ainda efetua a gestão do saldo de cada cliente;

- **DishDetailsControllers:**

Constituído pelos métodos CRUD, este controlador diz respeito aos pratos da cantina, permitindo a criação, a atualização e eliminação dos registos de pratos, assegurando que os dados relativos à descrição dos pratos, tipo e estado (ativo ou inativo) estão devidamente seguros;

- **EmployeesDetailsController:**

Este controlador, é constituído pelos métodos CRUD e recorre à base de dados para adicionar, alterar ou remover os registos de funcionários presentes na base de dados.

- **ExtraDetailsController:**

O presente controlador possui diversos métodos para criar, editar ou eliminar (métodos CRUD) itens extra da base de dados. Neste sentido, o método

“CreateExtra” permite adicionar um novo item extra, já o “EditExtra” permite efetuar a atualização de um item criado anteriormente, e, por fim, o “DeleteExtra” remove um extra da base de dados através do seu id.

- **PenaltyDetailsController:**

É o controlador responsável pela gestão de operações das penalidades atribuídas aos clientes recorrendo aos métodos CRUD. Neste sentido, o método “CreatePenalty” adiciona uma nova penalidade com o valor e horas de registo.

Já o “ReadPenalty” permite que o funcionário leia uma determinada penalidade encontrada pelo seu id. Por sua vez, o “Updatepenalty”, está relacionado com a edição da penalidade, ou seja, quando é editada este método vai atualizar os valores existentes. O “DeletePenalty” remove uma penalidade por via do seu id.

- **ReservationDetailsController**

É o controlador responsável pelas operações nas reservas dos menus da cantina. Para além possibilitar a criação, atualização, eliminação e apresentar a listagem de reservas, também é nele que são calculadas as penalidades. Importa ainda frisar que também concentra a gestão de faturas e as verificações do tipo de cliente, aluno ou estudante, com vista a assegurar a devida ligação entre reservas a clientes e menus disponíveis.

## **2.2. Controllers Lists:**

- **ListEmployeesController:**

O controlador faz a gestão da listagem dos vários funcionários inseridos no sistema. Posto isto, o método “GetEmployees(string search = ””)” efetua o retorno de uma lista de todos os funcionários se não for inserido nenhum critério (“search” vazio) ou então faz a filtragem dos funcionários com recurso ao nome específico do funcionário ou o seu NIF.

- **ListClientsController:**

O controlador “ListClientsController” é responsável pela gestão da listagem dos clientes, alunos ou professores. O método “GetStudents(string search = “”)” retorna a listagem de todos os alunos que se encontram registados na base de dados, ou efetua uma filtragem para um aluno específico, tendo por base o seu nome ou NIF. O mesmo se processa com o método “GetProfessors(string search = “”), mas com a diferença de que um professor pode se procurado pelo seu email.

- **ListDishesController :**

O controlador é responsável pela listagem dos pratos disponíveis no sistema. Neste sentido, está presente o método “GetDishes(string search = “”)” que efetua o retorno de uma lista de todos os pratos caso não seja aplicado nenhum critério em string search = “”, ou então filtra os pratos tendo por base a descrição dos mesmos.

- **ListExtrasController:**

O controlador é responsável pela listagem dos extras disponíveis no sistema. Para tal, é usado o método “GetExtras(string search = “”)” que efetua o retorno de uma lista de todos os extras se eventualmente não for aplicado nenhum critério (“search” vazio), ou então filtra os extras tendo por base a descrição dos mesmos, tal como se sucede com os pratos.

- **ListCanteenMenuController:**

Responsável pela gestão dos dados dos menus da cantina. Contém o método “GetWeeksofYear” que retorna uma lista de semanas para um determinado ano em específico. O “GetMenus” é método responsável por retornar uma lista de menus para uma data em particular, através de um processo de filtração dos menus pela data correspondente à semana no calendário. Por último, o “GetYears” é o método que irá retornar uma lista de anos em que há menus registados no sistema.

- **ListReservationsController:**

Trata-se de um controlador que lida diretamente com as reservas dos clientes, bem como as respetivas faturas. Neste sentido, o método “GetClients(string term=“”)” faz o retorno de uma lista de clientes, que poderão ser professores ou estudantes. Caso sejam

aplicados filtros de procura como nome ou nif, ele vai selecionar o cliente específico, senão retorna uma lista com todos os clientes.

Posteriormente, encontramos o método “GetFutureReservations(string clientNif)” que efetua o retorno de uma lista de reservas futuras de um determinado cliente, sendo identificado pelo seu NIF. A par do nome do cliente, ele retorna o tipo de prato, o menu e qualquer penalidade que tenha sido associada ao cliente.

Após o método “GetFutureReservations(string clientNif)”, é usado o método “GetServedReservations(string clientNif)” que é utilizado para sinalizar uma certa reserva como tendo sido servida. Para além de efetuar esta sinalização, é através deste método que é criada uma fatura da reserva que calcula o valor total a ser cobrado e inclui as linhas da fatura(“InvoiceLine”) para o menu que foi reservado e os extras que eventualmente tenham sido reservados. Para além disso, o método gera um arquivo PDF da fatura do cliente com os dados do respetivo cliente e o valor total da mesma.

- **ListInvoicesController:**

O controlador faz a gestão da listagem faturas inseridas no sistema. Posto isto, o método “GetInvoices(string search = ””)” efetua o retorno de uma lista com todas as faturas se não for inserido nenhum critério (“search” vazio). Caso contrário, faz a filtragem faturas presentes no sistema através da sua data de criação.

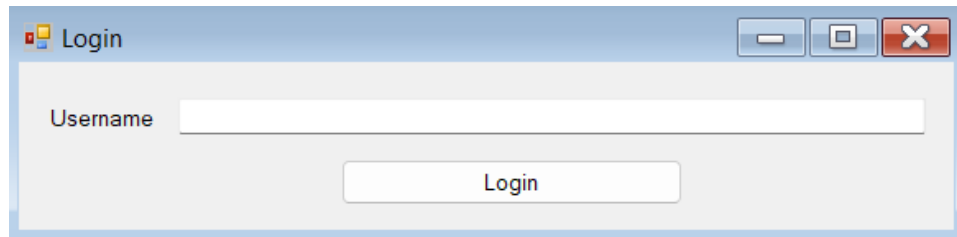
- **ListPenaltiesController:**

O controlador gere as multas que são inseridas no sistema. Posto isto, o método “GetPenalties(string search = ””)” efetua o retorno de uma lista com todas as multas aplicadas caso não tenha sido especificado nenhum critério, ou seja, (“search” vazio). Senão, faz a filtragem das penalidades presentes no sistema com recurso no seu id, valor, ou horas de atraso.

### 3. Views

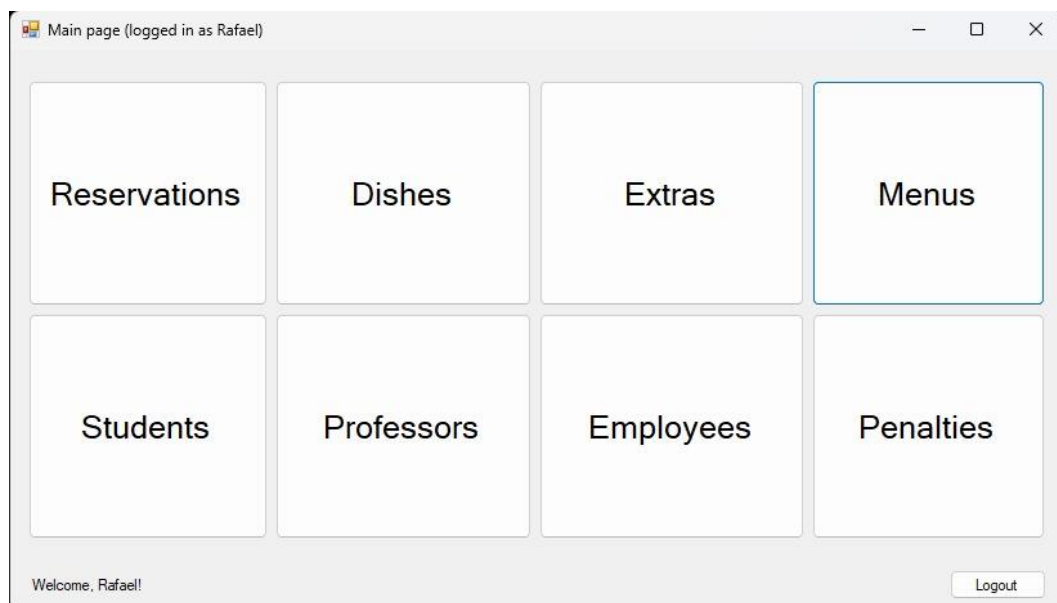
O projeto conta com uma view “Login”, figura 13, onde o funcionário se pode autenticar no sistema, caso já se encontra registado na base de dados.

Figura 13: Login

A screenshot of a Windows-style window titled "Login". It features a text input field labeled "Username" and a "Login" button below it. The window has standard minimize, maximize, and close buttons in the top right corner.

Após efetuar o login, o funcionário visualiza a view “Main Page” que apresenta um painel com redireccionamentos para as diversas funcionalidades que integram o sistema. Para além disso, o funcionário poderá sair do sistema clicando no botão “Logout”.

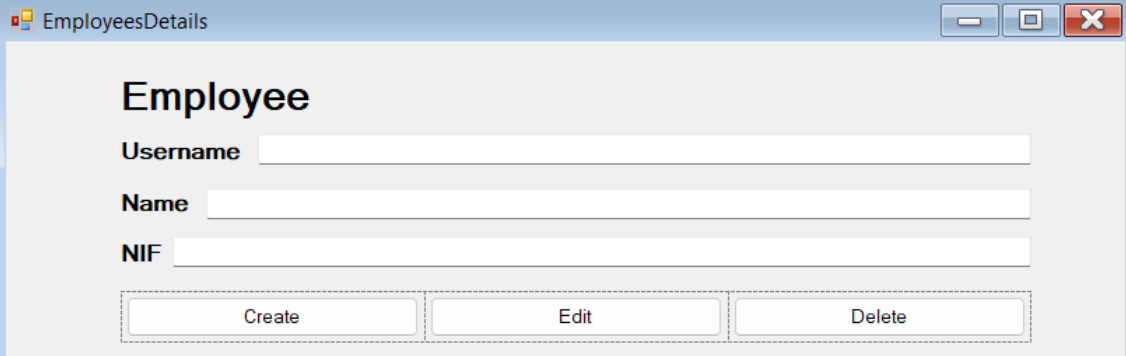
Figura 14: Main Page



As views do projeto encontram-se divididas em dois tipos distintos, as “Views Details” onde são criados, editados e eliminados os múltiplos registos do projeto, podendo ser clientes, tipos de pratos, menus, faturas, entre outros. Já as “Views Lists” são responsáveis por listar os registos criados as diversas view details.

### 3.1. Views Details

Figura 15: Employee Details

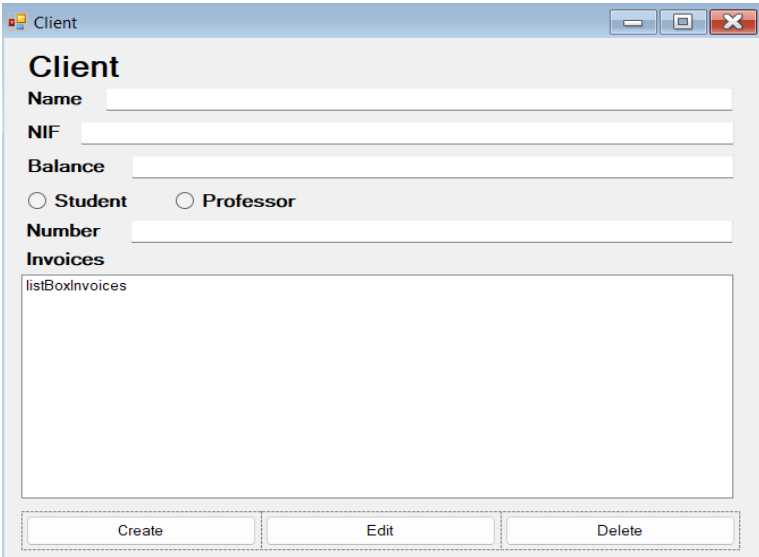


The screenshot shows a window titled "EmployeeDetails". Inside, there is a section titled "Employee". Below this title are three text input fields labeled "Username", "Name", and "NIF". At the bottom of the window, there are three buttons: "Create", "Edit", and "Delete".

A view “Employee Details ” tem como principal função adicionar funcionários ao sistema. A par disso, também permite editar os dados daqueles que já se encontram registados e ainda eliminá-los do sistema. Para a criação de um funcionário é obrigatório inserir três tipos dados, o “Username”, o “Name” e o “Nif”.

Importa frisar que, quando o sistema não detém nenhum registo na base de dados, antes de surgir a view login, surge esta view de modo a ser criado um utilizador e ficar registado na base de dados, só depois surge então a view do login. Depois, o processo de inserir mais funcionários é realizado através do botão “Employees” presente na main page.

Figura 16: Client Details



The screenshot shows a window titled "Client". Inside, there is a section titled "Client". Below this title are five text input fields labeled "Name", "NIF", "Balance", "Number", and "Invoices". Below the "Invoices" field, there are two radio buttons labeled "Student" and "Professor". At the bottom of the window, there are three buttons: "Create", "Edit", and "Delete".



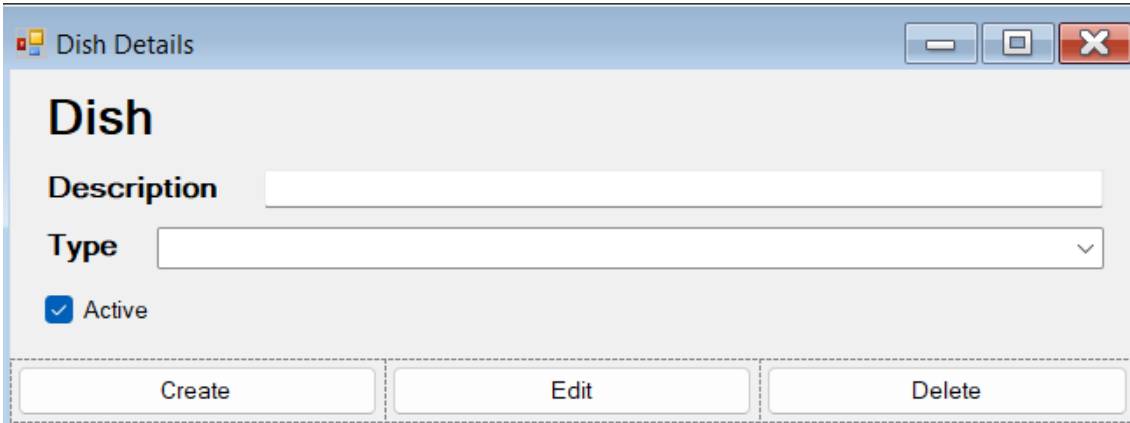
A view “Client Details”, permite efetuar a gestão dos detalhes dos clientes (professores ou estudantes) da cantina. Começa-se por escolher o tipo de cliente, podendo assumir-se como "Student" ou "Professor", sendo que a interface é ajustada tendo por base a opção escolhida. Depois, o funcionário deve redigir o nome e o NIF do cliente, seguindo-se o “Balance” do cliente, isto é, o seu saldo.

Tal como foi referido anteriormente, a seleção do tipo de cliente faz com que haja uma diferença no tipo de dados solicitados. Neste sentido, para os estudantes, é solicitado o "Number"/número de estudante, já na opção “Professor/”professores, é pedido o "Email".

A validação dos campos é realizada no preciso momento, destacando qualquer tipo de infração a vermelho. Posto isto, quando todos os dados são inseridos devidamente, o botão “Create” é desbloqueado, permitindo a criação do cliente ao ser clicado. Caso um cliente (estudante ou professor) for transposto para a view, seus dados são carregados automaticamente nos campos correspondentes.

Deste modo, é possível alterar os dados inseridos anteriormente no processo de “create”. Após a edição dos campos, o funcionário pode clicar no botão “Edit” para salvar as alterações. Esta view também contém a opção de eliminar o cliente atual clicando no botão “Delete”. Para finalizar, apresenta uma lista na qual são exibidas as faturas associadas ao cliente.

*Figura 17: Dish Details*

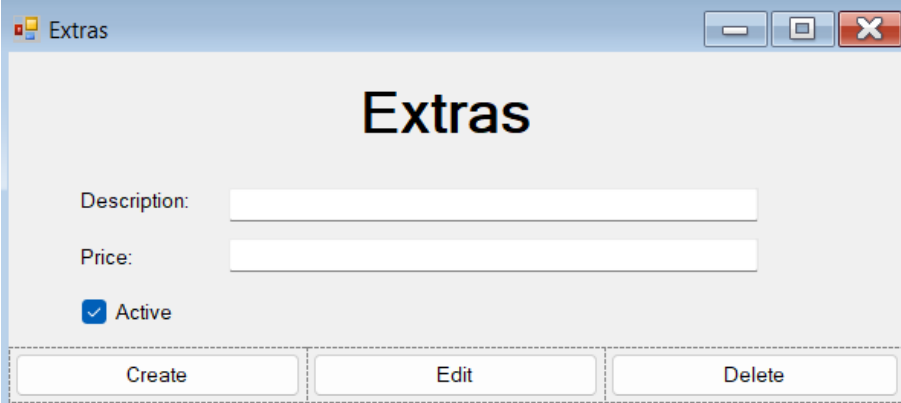


The screenshot shows a window titled "Dish Details" with standard Windows window controls (minimize, maximize, close). The main content area has a title "Dish" in large bold font. Below it are two labels: "Description" followed by a text input field, and "Type" followed by a dropdown menu. Below these is a checkbox labeled "Active" which is checked. At the bottom of the window, there are three buttons: "Create", "Edit", and "Delete".

Através da view “Dish Details”, é possível criar pratos, editá-los, ou ainda eliminá-los. Inicia-se com um campo para a descrição do prato, seguindo-se uma caixa de seleção para se selecionar o tipo de prato que se encontra enumerado e ainda uma caixa de verificação que permite marcar se o prato está ativo ou não.

A validação dos campos é realizada no momento sendo, que a descrição deve conter somente letras, números e espaços; caso seja válida, o campo é destacado em branco, caso contrário, em vermelho. Ao ser preenchido devidamente, os botões “Create”, “Edit” e “Delete” são habilitados consoante a ação. Em suma, a view “Dish Details” tem como objetivo estabelecer uma gestão eficiente dos vários pratos, agilizando os processos de criação, edição e eliminação dos registos.

*Figura 18: Extras Details*



The screenshot shows a window titled "Extras" with a standard Windows-style title bar. The main content area has a light gray background. At the top, the word "Extras" is displayed in a large, bold, black font. Below this, there are three input fields: "Description:" followed by a text box, "Price:" followed by a text box, and a checked checkbox labeled "Active". At the bottom of the window, there is a horizontal bar containing three buttons: "Create", "Edit", and "Delete". The window has standard minimize, maximize, and close buttons in the top right corner.

Semelhante ao “Dish Details”, a view “Extras Details”, permite gerir os extras alimentares que irão compor os menus, juntamente com os pratos. Também se inicia com um campo para a descrição do prato, uma textbox para ser incluído o preço e uma caixa de seleção para se assinalar se o extra está ativo ou não.

Quanto aos botões de funcionalidades, o de “Create” permite criar um extra com base nos detalhes fornecidos. Já o botão “Edit” permite atualizar os detalhes de um extra existente. O botão “Delete” permite remover o extra do sistema.

Figura 19: Canteen Menu Details

Designada por “Canteen Menu”, esta view permite gerir os detalhes do menu para os clientes. Deste modo, inicia-se com a escolha do tipo de refeição “Lunch” ou “Dinner”, depois a escolha do tipo de prato criado na view “Dish Details”. Após isto, é inserido o valor do menu para os estudantes e professores e são adicionados os extras que irão compor o menu. De seguida, é selecionado o dia para o qual pretendemos criar este menu.

Por fim, para ser criado o menu basta selecionar o botão “Create”. Para além disso, o funcionário pode ainda editar o menu criado, ou então eliminá-lo através dos outros dois botões, respetivamente.

Figura 20: Create Reservation Details

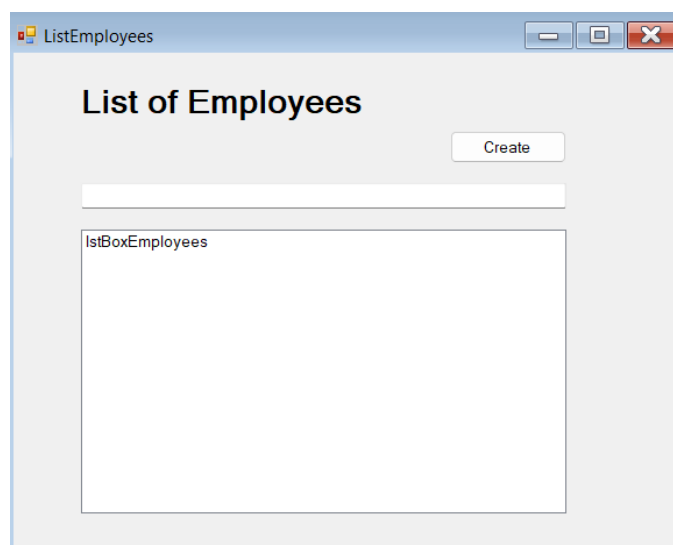
A view “Reservation Details” permite que o funcionário efetue a gestão das reservas na cantina. Ao ser selecionada esta view, o funcionário seleciona o nome do cliente que pretende efetuar a reserva e que já se encontra registado na base de dados. Depois, seleciona o ano e a semana na qual o cliente pretende fazer a reserva.

De seguida, são listados os vários menus na “listboxMenus”, que foram anteriormente criados na view “Canteen Menu”. O mesmo se sucede com os extras.

Na parte final da view, encontra-se o saldo do cliente e o custo total do menu, sendo este calculado com base no tipo de cliente (estudante ou professor) e nos extras selecionados. Ao ser clicado no “Create”, automaticamente há uma verificação do saldo existente, se o cliente tiver saldo suficiente então é realizada a reserva, caso contrário o sistema não permitirá efetuar a reserva. Para além disso, quando é acionado o botão “Create”, é criado um arquivo .txt onde estão presentes diversas informações sobre o cliente e o menu escolhido na reserva.

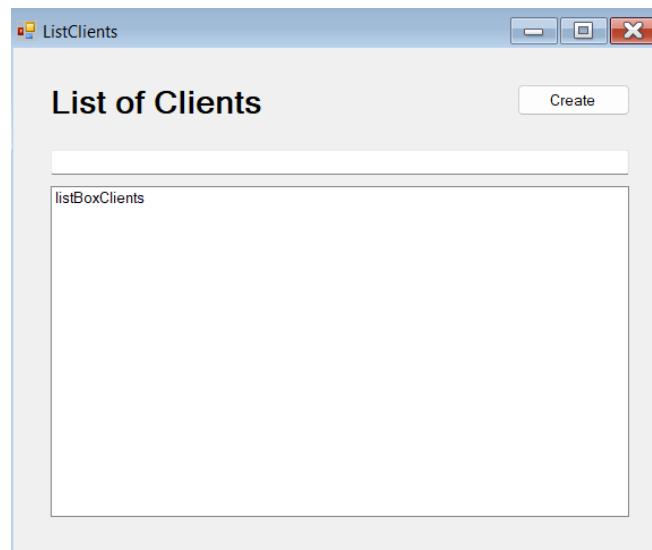
### 3.2. Views Lists

*Figura 21: List of Employees*



A view “List of Employees” permite listar os funcionários registados no sistema, após serem criados na view “Employees Details”. Quando o funcionário efetua um clique duplo no registo do funcionário listado, irá retornar à view “Employees Details” e poderá editar os dados do registo, ou eliminá-los.

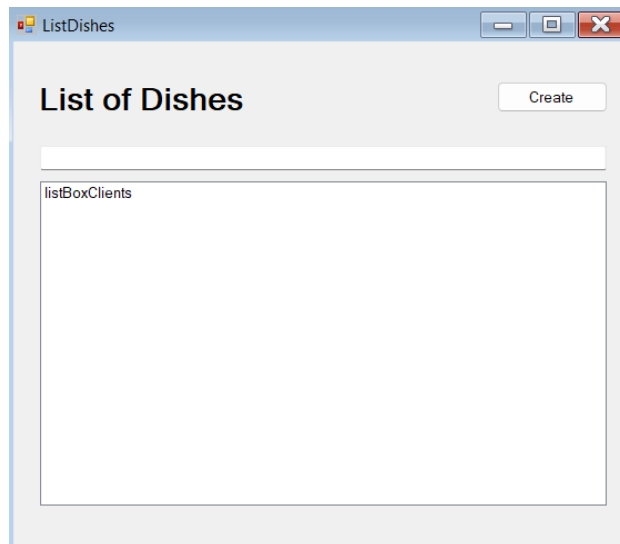
Figura 22: List of Clients



A view “List of Clients”, conta com a presença do botão “Create” que permite ao funcionário aceder à view “Client Details”, onde poderá criar o registo de um novo cliente.

Para além disso, conta com uma listbox que lista todos os clientes registados, sendo que o funcionário, ao efetuar um duplo clique no nome do cliente listado, acederá diretamente à view “Client Details” e poderá editar ou eliminar os dados desse mesmo cliente.

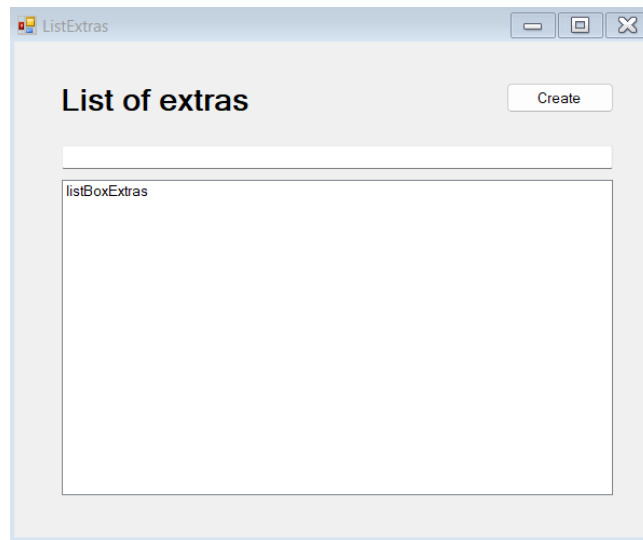
Figura 23: List of Dishes



A view “List of Dishes”, contém o botão “Create” que permite ao funcionário aceder à view “Dishes Details”, onde poderá criar o registo de um novo prato.

Está também presente uma listbox que lista todos os pratos registados, sendo que o funcionário, ao efetuar um duplo clique no nome do prato listado, acederá diretamente à view “Dishes Details” e poderá editar esse mesmo prato em específico ou eliminá-lo.

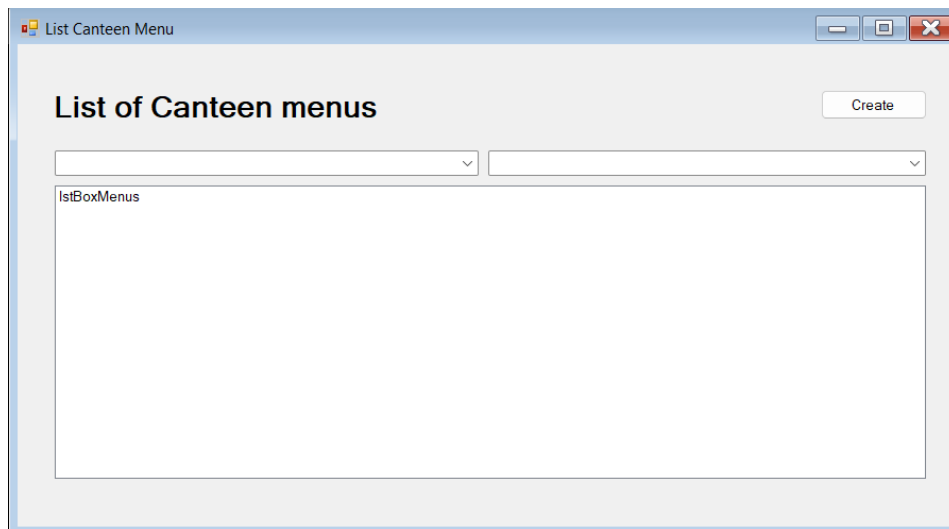
*Figura 24: List of Extras*



A view “List of Extras” é muito semelhante à anterior, contém o botão “Create” que permite ao funcionário aceder à view “Extras Details”, onde poderá criar o registo de um novo prato.

Conta com uma listbox que lista todos os extras registados, sendo que o funcionário, ao efetuar um duplo clique no nome do extra listado, acederá diretamente à view “Extras Details”, onde poderá editar esse mesmo extra em específico ou eliminá-lo.

Figura 25: List Canteen Menus



Na view “List of Canteen Menus”, está presente um botão, o “Create”, que redireciona o funcionário para a view “Canteen Menu Details” para ser criado um menu.

Depois, há uma listbox onde são listados os menus criados na view “Canteen Menu Details”. Para além de apresentar o nome dos mesmos, é visível a data para o qual foram criados. Esta view conta ainda com a possibilidade de selecionar o ano e a semana em que queremos verificar se existem menus listados.

Importa ainda frisar que um duplo clique no menu listado, faz com que surja a view “Canteen Menu Details” com intuito de editar ou eliminar esse menu em específico.

Figura 26: List Reservations

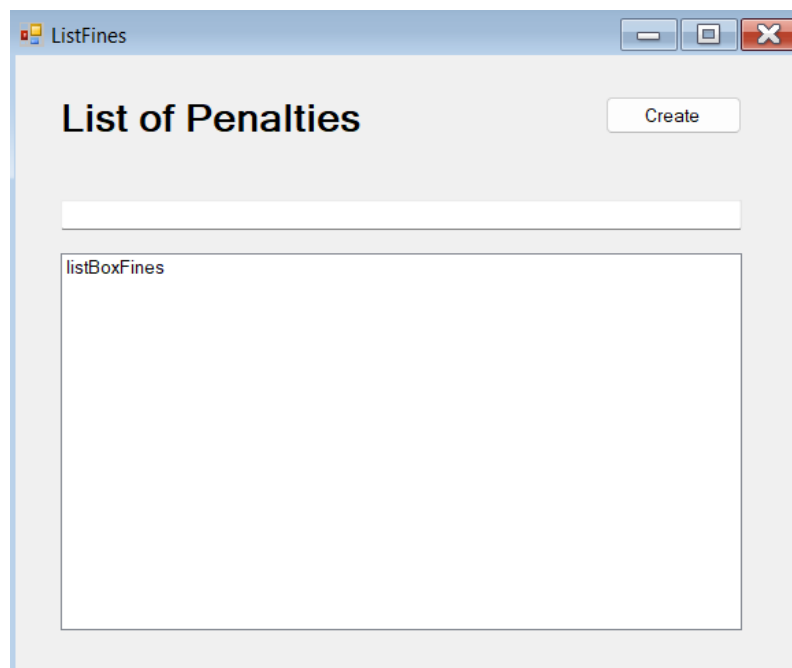
The screenshot shows a Windows application window titled "ListReservations". The window contains a form with the following elements:

- A title "Reservations" centered at the top of the form area.
- A "Client" label followed by a dropdown menu.
- A large empty list box labeled "listBoxReservations" in its top-left corner.
- Two buttons: "Create reservation" and "Mark as served", positioned side-by-side below the first list box.
- A second empty list box labeled "listBoxPastReservations" in its top-left corner, located below the buttons.

A view “ List of Reservations” permite ao funcionário criar as reservas ao clicar no botão “Create reservation ”. Após serem criadas, é listado o nome do cliente e surge na ”listboxreservations” as reservas efetuadas. Quando o botão “Mark as served” é pressionado, a reserva que estava na “listBoxReservations” passa para a “listBoxPastReservations” e é gerado um pdf com os dados dessa mesma reserva. Para o pdf ser gerado, foi instalada a biblioteca PDFsharp que processa arquivos em pdf.



Figura 27: List of Penalties



A view “List of Penalties” contém o botão “Create” que permite ao funcionário aceder à view “Penalties Details”, onde poderá criar o registo de uma nova multa. Após ser criada, é listada na listBoxFines, deste modo, o funcionário consegue visualizar todas as multas instauradas aos clientes.