Conteúdo

[Título e Objetivo 1](#_Toc163124612)

[Autor, Data, Contexto de Trabalho 1](#_Toc163124613)

[Descrição do problema 1](#_Toc163124614)

[Ferramentas e versões 2](#_Toc163124615)

[Análise de dados 2](#_Toc163124616)

[Descrição de processos 2](#_Toc163124617)

[Dicionário 3](#_Toc163124618)

[String connection, instâncias, bases de dados 3](#_Toc163124619)

[Lista de tabelas, campos e tipos 3](#_Toc163124620)

[Namespaces 3](#_Toc163124621)

[Classes e métodos 3](#_Toc163124622)

[Classe LoginController 3](#_Toc163124623)

[Classe RegistoController 3](#_Toc163124624)

[Classe DjController 4](#_Toc163124625)

[Classe UserController 4](#_Toc163124626)

[Classe MusicasController 5](#_Toc163124627)

[Classe PlayListController 5](#_Toc163124628)

[Classe MusicasInPlayListController 6](#_Toc163124629)

[Classe PedidosController 6](#_Toc163124630)

[Classe ListagemController 7](#_Toc163124631)

[Classe PerfilController 7](#_Toc163124632)

[Classe ApplicationDbContext 7](#_Toc163124633)

[Outras informações 8](#_Toc163124634)

# Título e Objetivo

**Pay & Play: Escolha, Pague, Curta com Nosso DJ.**

**Permitir Utilizadores Escolher a sua Música Favorita no seu Ambiente Favorito.**

## Autor, Data, Contexto de Trabalho

Afonso Figueiredo, abril, UFCD 26.

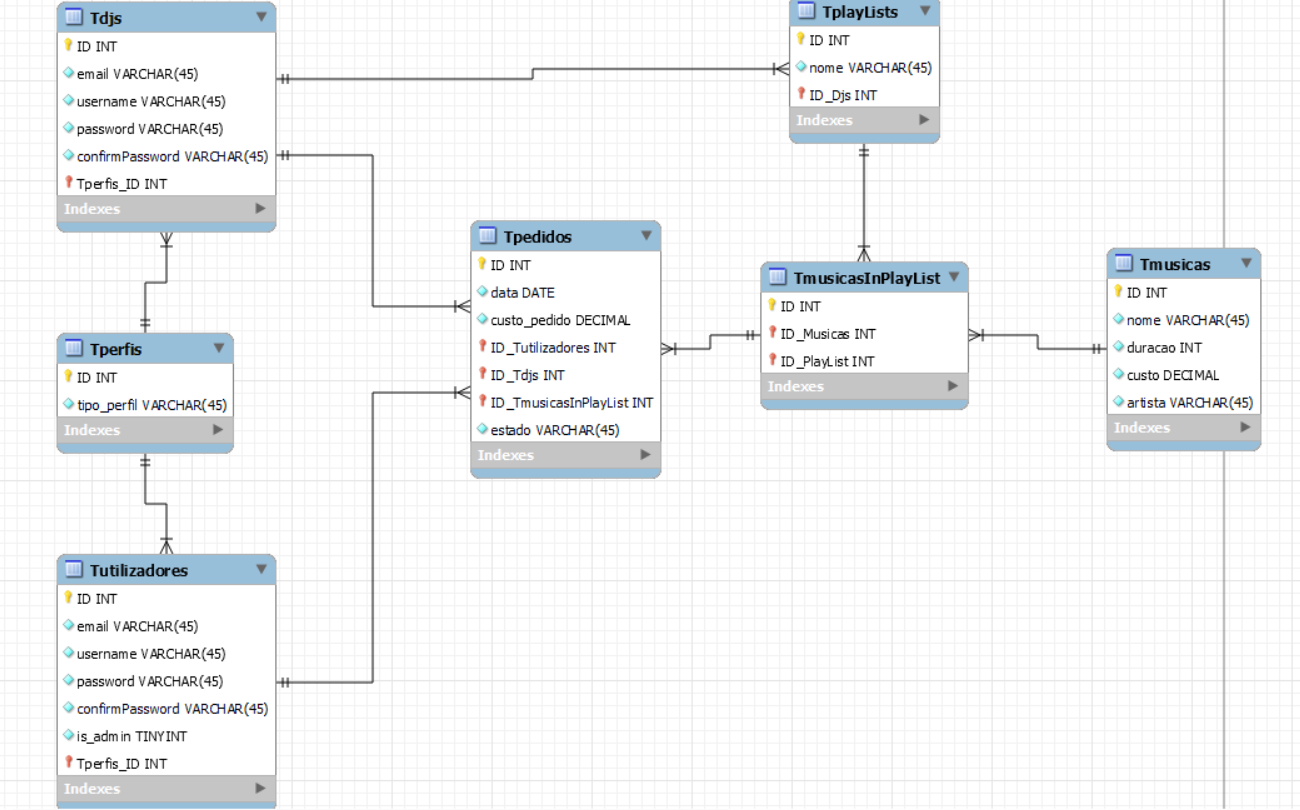
## Descrição do problema

Uma aplicação que permita a utilizadores fazer pedidos de músicas a Djs através de QrCode pagando por esse pedido, permitindo assim que os utilizadores possam disfrutar da sua música favorita. A aplicação deverá ter utilizadores do tipo Dj e utilizadores normais, os utilizadores do tipo Dj, teriam acesso a criar as suas playlists a partir de uma base de dados de músicas publica para todos os Djs, sendo-lhes possível também adicionar músicas a essa base, um QRCode que será usado pelos utilizadores normais acederem à sua playlist e escolherem a música que querem ouvir, esse QRCode pode ser gerado no início da noite (turno) e desativado quando o Dj terminar o seu tempo. Terá acesso também ao seu perfil onde terá acesso a todos os pedidos concluídos e pendentes e também ao total de dinheiro ganho nessa noite. Do lado do utilizador terá acesso a um mecanismo para ler o QRCode daquele Dj e será encaminhado para as suas playlists onde poderá escolher a música que quer ouvir, escolhendo a música que quer ouvir é redirecionado para o pagamento do pedido onde ficara pendente dependendo da aceitação do pedido e consequente passagem da música por parte do Dj, sendo confirmado a passagem da música por parte do Dj e do utilizador, o pagamento é concluído.

## Ferramentas e versões

* .NET 8.0
* ASP .NET CORE
* SQL Server Managment Studio 19
* Linq

# Análise de dados



# Descrição de processos

1. Processos Relativos a Dj:
   * + CRUD playlists;
     + Inserção novas músicas;
     + Gerar/Desativar QrCode;
     + Gerir Perfil;
     + Gerir Pedidos;
     + Levantar Dinheiro Ganho;

1. Processos Relativos a Utilizador Comum:
   * + Gerir Perfil;
     + Ler QrCode;
     + Escolher Música;
     + Efetuar Pagamento;
     + Gerir Pedidos;
2. Outros Processos:
   * + Mecanismo de Registo e Login;
     + Cálculo de Saldo Ganho de Dj;

(Listagens DJ)

* + - Listagem Ganhos por mês;
    - Listagem Ganhos por período específico;
    - Listagem Músicas mais/menos pedidas;
    - Listagem de Utilizadores com mais/menos pedidos.
    - Listagem de Utilizadores com mais/menos dinheiro gasto.

(Listagens Utilizador Comum)

* + - Listagem Gastos por mês;
    - Listagem Gastos por período específico;
    - Listagem Gastos por Dj;

# Dicionário

## String connection, instâncias, bases de dados

**String Connection:**

“Data Source = 62.28.39.135,62444;

Initial Catalog = PayAndPlay\_Afonso;

User ID = EFAafonso;

Password=Abc.123##;

Trust Server Certificate = True”

**Instância:**

Ip: 62.28.39.135

Porto: 62444

**Base de dados:** PayAndPlay\_Afonso

## Lista de tabelas, campos e tipos

Referenciado no Diagrama de Entidade-Relação acima.

## Namespaces

PayAndPlay

## Classes e métodos

### Classe LoginController

**Index():** Este método é acionado quando a página de login é acedida via HTTP GET. Retorna uma visualização para a página de login

**Index(string email, string password):** Este método é acionado quando os dados de login são submetidos via HTTP POST. Verifica se o email de utilizador e a palavra-passe foram fornecidos. Em seguida, verifica na base de dados se existe um utilizador correspondente às credenciais fornecidas. Se sim, armazena as informações do utilizador na sessão e redireciona para o perfil. Se não, exibe uma mensagem de erro na página de login.

**LogOut():** Este método é acionado quando o utilizador deseja sair do sistema. Limpa todos os dados da sessão e redireciona de volta para a página de login.

Essencialmente, a classe LoginController gere o processo de autenticação dos utilizadores, verificando as credenciais fornecidas, estabelecendo a sessão do utilizador após a autenticação bem-sucedida e fornecendo uma forma de sair do sistema através do método Logout().

### Classe RegistoController

**Index():** Este método é acionado quando a página de Registo é acedida via HTTP GET. Retorna uma visualização para a página de login

**Index(string email, string username, string password):** Este método é acionado quando os dados de registo são submetidos via HTTP POST. Regista o email de utilizador, o username e a palavra-passe que foram fornecidos. Em seguida, redireciona para o perfil do novo utilizador registado.

### Classe DjController

A classe DjController é responsável por gerir as operações relacionadas com os DJs, é acedida apenas por admins. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todos os DJs armazenados na base de dados.

**Details(int? id):** Este método retorna os detalhes de um DJ específico com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de novo Dj.

**Create([Bind("Id,Email,Username,Password")] Dj dj):** Método acionado quando os dados do novo DJ são submetidos via HTTP POST. Adiciona um novo DJ à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição de DJ existente com base no ID fornecido.

**Edit(int id, [Bind("Id,Email,Username,Password")] Dj dj):** Método acionado quando os dados de edição de DJ são submetidos via HTTP POST. Atualiza as informações do DJ na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão de DJ com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão de DJ é confirmada. Remove o DJ da base de dados.

**DjExists(int id):** Método auxiliar para verificar se um DJ com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para gerir os DJs no sistema.

### Classe UserController

A classe UserController é responsável por gerir as operações relacionadas com os utilizadores comuns, é acedida apenas por admins. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todos os utilizadores comuns armazenados na base de dados.

**Details(int? id):** Este método retorna os detalhes de um utilizador comum específico com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de novo utilizador comum.

**Create([Bind("Id,Email,Username,Password,is\_Admin")] Utilizador utilizador):** Método acionado quando os dados do novo utilizador comum são submetidos via HTTP POST. Adiciona um novo utilizador comum à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição de utilizador comum existente com base no ID fornecido.

**Edit(int id, [Bind"Id,Email,Username,Password,is\_Admin")] Utilizador utilizador):** Método acionado quando os dados de edição de utilizador comum são submetidos via HTTP POST. Atualiza as informações do utilizador comum na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão de utilizador comum com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão de utilizador comum é confirmada. Remove o utilizador comum da base de dados.

**UserExists(int id):** Método auxiliar para verificar se um utilizador comum com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para gerir os utilizadores comuns no sistema.

### Classe MusicasController

A classe MusicasController é responsável por gerir as operações relacionadas com as musicas, é acedida apenas por admins e Djs. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todas as musicas armazenadas na base de dados.

**Details(int? id):** Este método retorna os detalhes de uma musica específica com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de nova musica.

**Create[Bind"Id,Nome,Duracao,Custo")] Musica musica):** Método acionado quando os dados da nova musica são submetidos via HTTP POST. Adiciona uma nova musica à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição de utilizador comum existente com base no ID fornecido.

**Edit(int id, [Bind"Id,Nome,Duracao,Custo")] Musica musica):** Método acionado quando os dados de edição de musica são submetidos via HTTP POST. Atualiza as informações da musica na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão da musica com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão da musica é confirmada. Remove a musica da base de dados.

**MusicExists(int id):** Método auxiliar para verificar se uma musica com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para gerir as musicas no sistema.

### Classe PlayListController

A classe PlayListController é responsável por gerir as operações relacionadas com as playlists, é acedida apenas por admins e Djs. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todas as playlists armazenadas na base de dados.

**Details(int? id):** Este método retorna os detalhes de uma playlist específica com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de nova playlist.

**Create([Bind("Id,Nome,Dj\_Id")] PlayList playlist):** Método acionado quando os dados da playlist são submetidos via HTTP POST. Adiciona uma nova playlist à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição de playlist existente com base no ID fornecido.

**Edit(int id, [Bind("Id,Nome,Dj\_Id")] PlayList playlist):** Método acionado quando os dados de edição de playlist são submetidos via HTTP POST. Atualiza as informações da playlist na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão da playlist com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão da playlist é confirmada. Remove a playlist da base de dados.

**PlayListExists(int id):** Método auxiliar para verificar se uma playlist com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para gerir as playlists no sistema.

### Classe MusicasInPlayListController

A classe MusicasInPlayListController é responsável por gerir as operações relacionadas com as musicas associadas a playlists, é acedida apenas por admins e Djs. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todas as playlists e musicas a elas associadas armazenadas na base de dados.

**Details(int? id):** Este método retorna os detalhes de uma playlist e musica associada específica com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de nova musica na playlist.

**Create([Bind("Id,Musica\_Id,PlayList\_Id")] MusicaInPlayList musicainplaylist):** Método acionado quando os dados da musica na playlist são submetidos via HTTP POST. Adiciona uma nova musica na playlist à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição da musica na playlist existente com base no ID fornecido.

**Edit(int id, [Bind("Id,Musica\_Id,PlayList\_Id")] MusicaInPlayList musicainplaylist):** Método acionado quando os dados de edição da musica na playlist são submetidos via HTTP POST. Atualiza as informações da musica na playlist na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão da musica na playlist com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão da musica na playlist é confirmada. Remove a musica da playlist da base de dados.

**PlayListExists(int id):** Método auxiliar para verificar se uma musica na playlist com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para gerir as musicas nas playlists no sistema.

### Classe PedidosController

A classe PedidosController é responsável por gerir as operações relacionadas com os pedidos, é acedida apenas por admins. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index():** Este método devolve uma vista que lista todos os pedidos armazenadas na base de dados.

**Details(int? id):** Este método retorna os detalhes do pedido específico com base no ID fornecido.

**Create():** Método para exibir um formulário de criação de novo pedido.

**Create([Bind("Id,Utilizador\_Id,Dj\_Id,MusicasInPlaylist\_Id,Data,Custo\_Pedido”)] Pedido pedido):** Método acionado quando os dados do pedido são submetidos via HTTP POST. Adiciona um novo pedido à base de dados.

**Edit(int? id):** Método para exibir um formulário de edição do pedido com base no ID fornecido.

**Edit(int id, [Bind("Id,Utilizador\_Id,Dj\_Id,MusicasInPlaylist\_Id,Data,Custo\_Pedido”)] Pedido pedido):** Método acionado quando os dados de edição do pedido são submetidos via HTTP POST. Atualiza as informações do pedido na base de dados.

**Delete(int? id):** Método para exibir uma página de confirmação de exclusão do pedido com base no ID fornecido.

**DeleteConfirmed(int id):** Método acionado quando a exclusão do pedido é confirmada. Remove o pedido da base de dados.

**PedidoExists(int id):** Método auxiliar para verificar se pedido com o ID fornecido existe na base de dados.

Esta classe disponibiliza um conjunto completo de operações CRUD (Create, Read, Update, Delete) para os pedidos no sistema.

### Classe ListagemController

A classe ListagemController é responsável por gerir as operações de Listagem e Calculo de Ganhos e Gastos. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**ListarGanhosMes(DateOnly data, int Dj\_Id):** Este método retorna um decimal com o valor ganho no mês com base no ID fornecido e o mês especifico.

**ListarGanhosPeriodo(DateOnly data\_inicio, DateOnly data\_final, int Dj\_Id):** Este método retorna um decimal com o valor ganho no período de tempo com base no ID fornecido e o período especifico.

**ListarMusicasMaisPedidas(int Dj\_Id)**: Este método retorna a musica mais pedida a um Dj especifico.

**ListarMusicasMenosPedidas(int Dj\_Id)**: Este método retorna a musica menos pedida a um Dj especifico.

**ListarUtilizadorMaisPedidos(int Dj\_Id):** Este método retorna o utilizador com mais pedidos a um Dj especifico.

**ListarUtilizadorMenosPedidos(int Dj\_Id):** Este método retorna o utilizador com menos pedidos a um Dj especifico.

**ListarUtilizadorMaisGastos(int Dj\_Id):** Este método retorna o utilizador com mais gastos num Dj especifico.

**ListarUtilizadorMenosGastos(int Dj\_Id):** Este método retorna o utilizador com menos gastos num Dj especifico.

**ListarGastosMes(int Utilizador\_Id, DateOnly data):** Este método retorna o valor decimal dos gastos num mês especifico com base no ID fornecido e mês.

**ListarGastosPeriodo(int Utilizador\_Id, DateOnly data\_inicio, DateOnly data\_final):** Este método retorna o valor decimal dos gastos num período de tempo especifico com base no ID fornecido e o período de tempo.

**ListarGastosPorDj(int Utilizador\_Id, int Dj\_Id):** Este método retorna os gastos decimal por Dj.

**CalcularSaldoDj(int Dj\_Id):** Este método retorna o Saldo Total do Dj.

Esta classe disponibiliza um conjunto completo de operações de Listagem e Calculo do Saldo para os pedidos no sistema.

### Classe PerfilController

A classe PerfilController é responsável por gerir as operações de Perfil, apresentação de Listagens e Permite Gerar QrCodes para Dj e Ler QrCode para Utilizador. Aqui estão os métodos mais importantes:

**Construtor:** Recebe uma instância do contexto da base de dados através da injeção de dependência para permitir a interação com a base de dados.

**Index(int? Utilizador\_Id, int? Dj\_id):** Este método devolve uma vista do perfil dependendo do tipo de perfil do utilizador com as suas Listagens Especificas.

Esta classe disponibiliza uma vista do Perfil com as Listagens especificas a um utilizador ou Dj para os pedidos no sistema.

### Classe ApplicationDbContext

Esta classe representa o contexto da base de dados para a aplicação. Aqui estão os pontos principais:

**Construtores**: Esta classe possui dois construtores, um sem parâmetros e outro que recebe opções de configuração do contexto. O segundo construtor é utilizado para configurar as opções de acesso à base de dados.

**Propriedades DbSet**: Define propriedades virtuais para cada entidade do modelo de dados: Utilizador; Dj; PlayList; Musica; MusicaInPlayList; Pedido. Cada uma destas propriedades representa uma tabela na base de dados.

**OnConfiguring**: Subescreve o método OnConfiguring para configurar o provedor de base de dados. Neste caso, está a ser utilizado o provedor SQL Server com uma string de conexão especificada.

**OnModelCreating**: Sobrescreve o método OnModelCreating para configurar o modelo de dados através de Fluent API. Aqui são definidas as configurações de tabela, restrições e relações entre as entidades.

**Configuração das Entidades**: Utiliza o método Entity() para configurar cada entidade individualmente. Define o nome das tabelas, as propriedades e respetivos tipos de dados, índices e relações de chave estrangeira.

Esta classe é essencial para a interação com a base de dados e define a estrutura e comportamento das entidades no contexto da aplicação de gestão do hotel.

## Outras informações