**CENTRO UNIVERSITÁRIO SENAC**

**ARQUITETURA DE SOFTWARE**

Iago Guimarães

Caue Ghetti

Felipe Tadeu

Luis Henrique Lima

Natalia Farias

Contents

[1. Introdução 3](#_Toc500024421)

[2. Estrutura de Pastas 3](#_Toc500024422)

[3. Pasta APP 4](#_Toc500024423)

[3.1 Controller 5](#_Toc500024424)

[4. Pasta App\_Start 6](#_Toc500024425)

[5. Pasta Controllers 7](#_Toc500024426)

[6. Pasta Daos 9](#_Toc500024427)

[7. Pasta Models 11](#_Toc500024428)

[8. Pasta SQL Scripts 12](#_Toc500024429)

[9. Pasta Views 14](#_Toc500024430)

[10. Arquivos Soltos 16](#_Toc500024431)

[11. Console APP 17](#_Toc500024432)

[12. Demonstração do Sistema 18](#_Toc500024433)

# Introdução

Este trabalho consiste em apresentar um projeto de API, utilizando as práticas para suporte a arquitetura MVC proposto pelo professor de Arquitetura de Software do 4° semestre do curso Tecnologia em Analise de Desenvolvimento de Softaware.

Para que este trabalho fosse realizado, foram determinadas as seguintes exigencias:

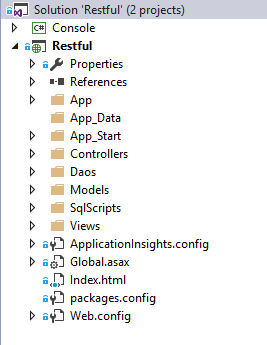
* Um modelo, representado através de um banco de dados relacional (mySQL, SQLite, Oracle, SQLServer, etc) com no mínimo três tabelas. O banco de dados pode ser aproveitado do projeto de Projeto Integrador do semestre.
* Um webservice RESTful no qual, para cada um das tabelas, possa permitir métodos GET, POST, PUT e DELETE. Este webservice pode ser desenvolvido em JSP, ASP, PHP, Python ou qualquer outra plataforma que desejar.
* Duas diferentes plataformas de visualização que consumam o webservice criado. Sugestão: um Web App (HTML/CSS/Javascript) e Java. Deve ser desenvolvida estas interfaces para acessar cada uma das tabelas

Decidimos desenvolver a API este projeto utilizando a linguagem de programação C# e os recursos Web API da plataforma .NET devido a sua facilidade de desenvolvimento.No front-end utilizamos o AngularJS, também visando a facilidade de desenvolvimento nas interações com o usuário. Para aplicar um pouco de estilização nas páginas do sistema, utilizamos o Bootstrap. O banco de dados escolhido foi o MySQL por ser fácil de instalar, gratuíto e possuir integração com o C#, sendo necessário só usar uma biblioteca que pode ser baixada e instalada pela propria IDE.

O projeto desenvolvido foi publicado no Github e pode ser acessado através do seguinte endreço: <https://github.com/iagoguimaraes/restful>

A seguir vamos apresentar como foi desenvolvido o projeto, ilustrando passo a passo da arquitetura MVC.

# Estrutura de Pastas

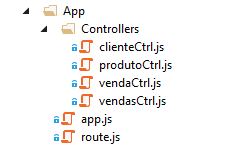


A solução é composta pela seguinte estruta de pastas (conforme ilustrado acima):

* **Console**: Um projeto a parte para testar as funcionalidades das operações do sistema
* **App**: Pasta que contém os scripts utilizado no front-end da aplicação
* **App\_Start**: Pasta contém alguns arquivos de configuração para o funcionamento da API Restful
* **Controllers**: Pasta que contém os arquivos da camada controladora que tem por objetivo o transporte dos dados da camada de dados para a camada de visualização
* **Daos**: Pasta que contém os arquivos da camada de dados, que faz comunicação com o banco de dados
* **Models**: Pasta que contém os arquivos da camada de modelos de entidades que são utilizados na aplicação para estruturar os dados
* **SqlScripts**: Pasta que contém os scripts de criação da estrutura do banco de dados para o funcionamento das operações do sistema
* **Views**: Pasta que contém os arquivos da camada de visualização. São as páginas do sistema
* **Arquivos soltos**: alguns arquivos soltos são gerados automaticamente na criação da solução que servem para configurar a aplicação web.

A seguir vamos aprofundar em cada uma das pastas citados, explicando mais detalhamente sobre os arquivos que cada uma deve possuir.

# Pasta APP



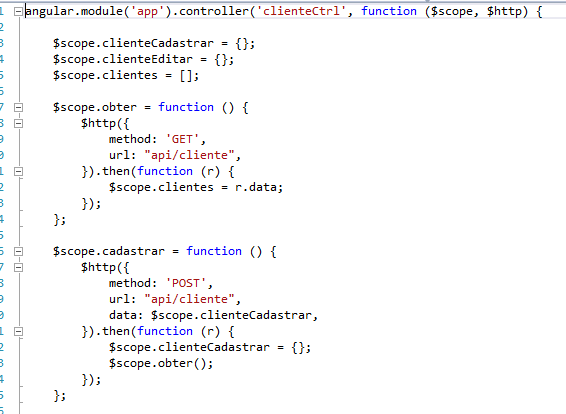
A pasta App contém os scripts que são utilizados no front-end para promover as interações com o usuário. Neste projeto utilizamos o AngularJS para desenvolver os scripts do front-end, devido a sua facilidade. Fizemos uma aplicação bem simples então só utilizamos controllers do AngularJS e o sistema de Rotas. Não foi necessários utilização de serviços e diretivas.

Abaixo uma ilustração de como foi programado o sistema de rotas do AngularJS que consiste em direcionar o usuário para a página que ele deseja acessar sem que seja necessário a atualização de página do navegador.



# Controller

Abaixo uma ilustração de como foram programados os controllers do AngularJS. Neste código foi ilustrado o controller da pagina do cadastro de clientes, mas todos foram feitos da mesma forma.





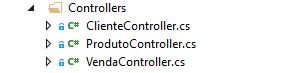
# Pasta App\_Start



Esta pasta contém arquivos de configuração do Web API. Abaixo uma ilustração do código que configura como as requisições são feitas para a API

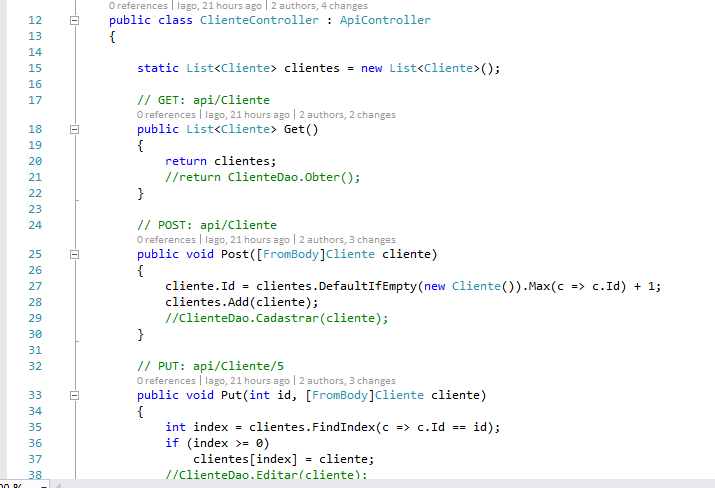


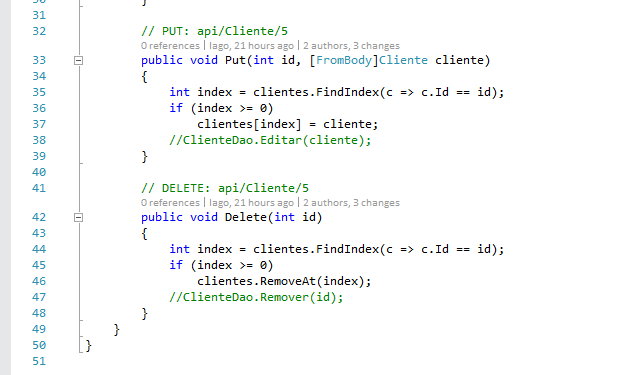
# Pasta Controllers



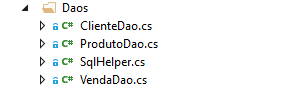
Essa pasta que contém os arquivos da camada controladora que tem por objetivo o transporte dos dados da camada de dados para a camada de visualização.

Abaixo vamos apresentar o código do arquivo ClienteController para apresentar como são programadas as operações do sistema que passam pela controladora. O conceito utilizado no ClienteController é usamos da mesma forma nos outros arquivos.



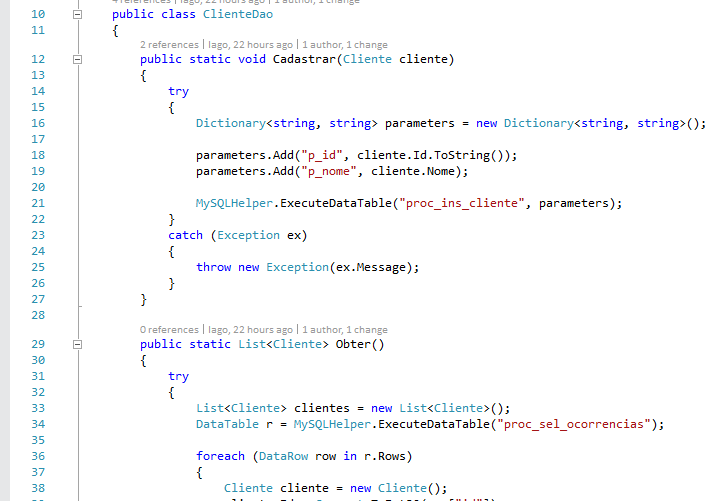


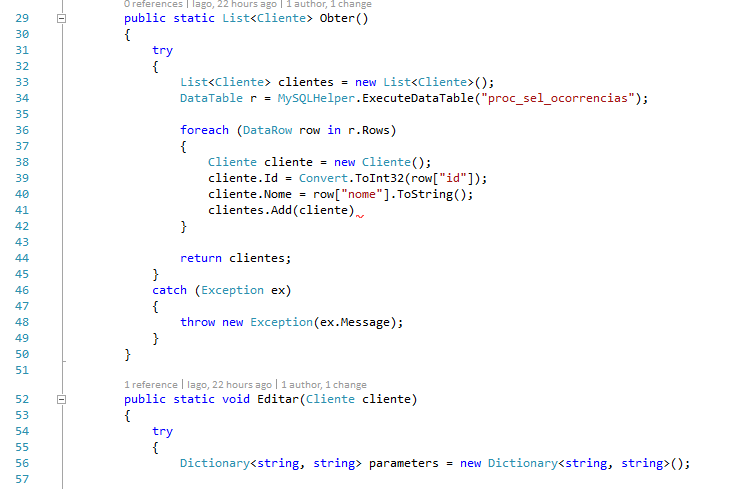
# Pasta Daos

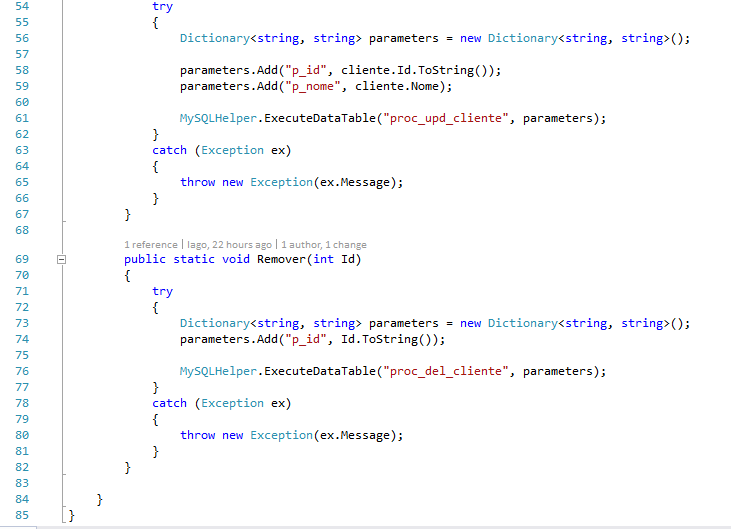


Pasta que contém os arquivos da camada de dados, que faz comunicação com o banco de dados.

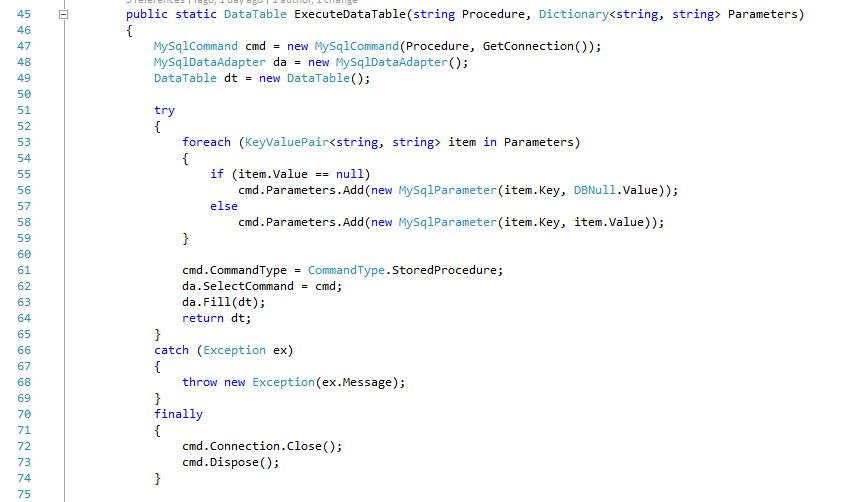
Abaixo uma ilustração de como foi desenvolvido o código do arquivo ClienteDao. Como foram feitas as operações do sistema se comunicando com o banco de dados, neste caso MySQL.



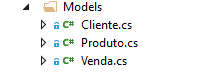




Abaixo também vamos ilustrar a classe MySQLHelper que foi utilizada nas Daos que serve para executar comandosno banco de dados e neste caso pudemos fazer o reaproveitamento de código.

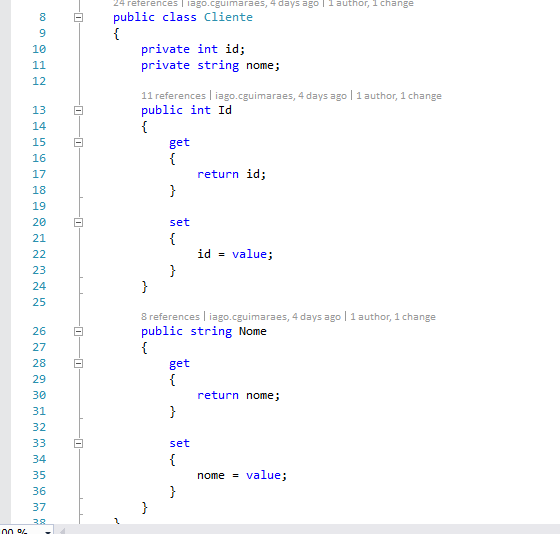


# Pasta Models



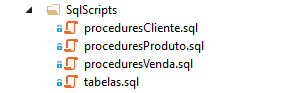
Pasta que contém os arquivos da camada de modelos de entidades que são utilizados na aplicação para estruturar os dados.

Abaixo ilustraremos o modelo Cliente. Os outros modelos utilizam o mesmo conceito também. Os atributos privados e as propriedades publicas de acesso aos atributos (chamados de getter e setters). Dessa forma usamos o conceito de encapsulamento da orientação a objetos

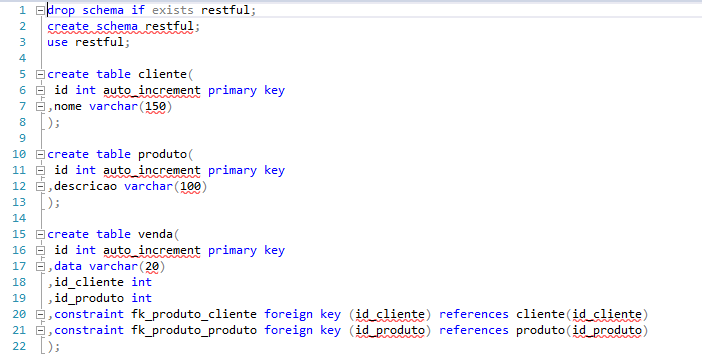


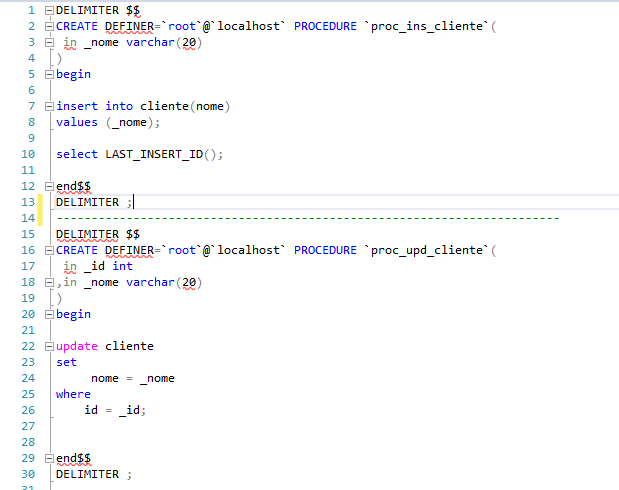
# Pasta SQL Scripts

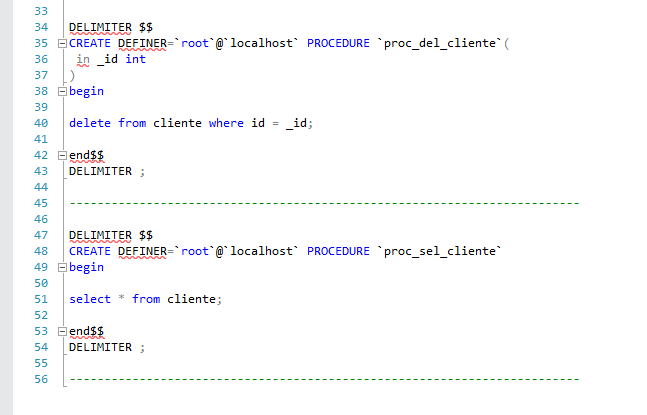
Nesta pasta está os scripts que foram usados para modelar o banco de dados do sistema.



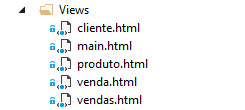
Abaixo vamos mostrar o script de criação das tabelas e os scripts das operações de criação, atualização, deleção e seleção da entidade cliente. O mesmo conceito de código foi utilizado para as outras entidades.





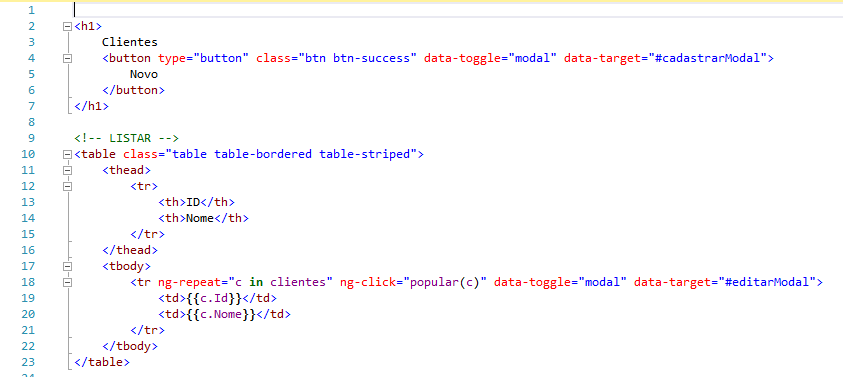
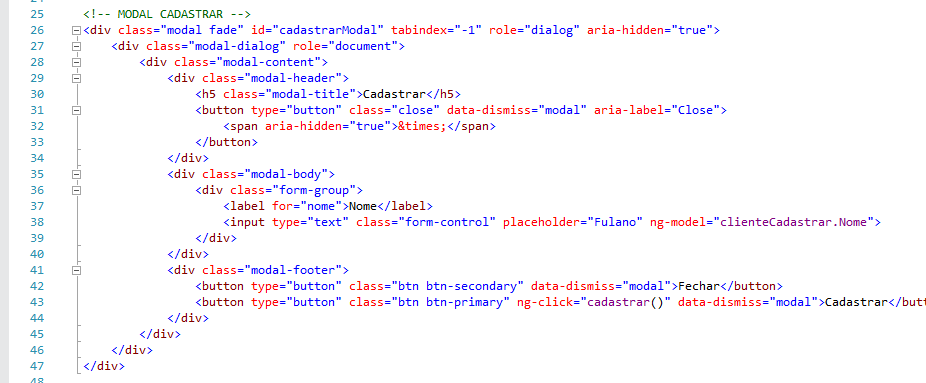


# Pasta Views

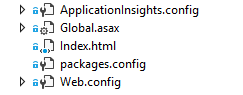


Pasta que contém os arquivos da camada de visualização. São as páginas do sistema.

Abaixo vamos ilustrar como foi desenvolvida a pagina de cliente. Todas as outras utilizaram o mesmo conceito. Neste projeto fizemos uma tabela para listar o clientes existentes, um pop up para quando o usuário for cadastrar um novo cliente e outro para quando o usuário for editar ou excluir algum clientes existente.

# Arquivos Soltos



Alguns arquivos soltos são gerados automaticamente na criação da solução que servem para configurar a aplicação web.

Vamos apenas demonstrar um deles que é a pagina Index.html

Esta página fica fora da pasta View, pois desta forma a aplicação já redireciona para essa página sempre que o endereço da apicação for acessado. É a pagina padrão da aplicação.

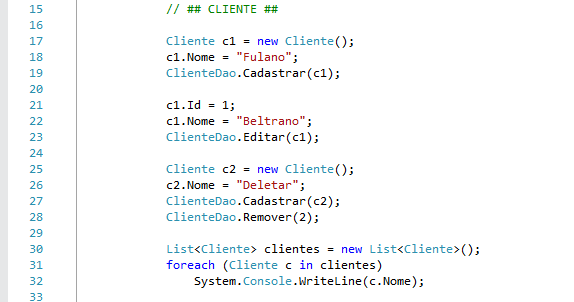
E nesta página, fica a master page da aplicação. A master page é a pagina que coném o menu, a importação de todos os scripts (javascript, angular, bootstrap) e que faz a integração entre as visualizações sem que tenha que atualizar a pagina no navegador, usando o conceito de Single Page Aplication.

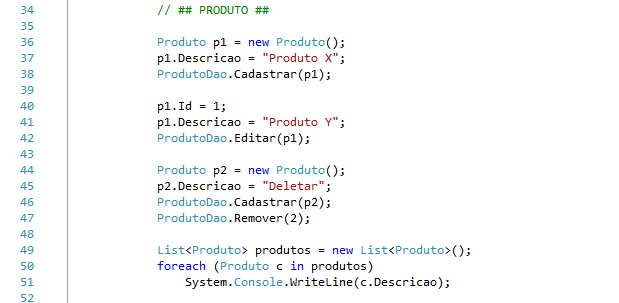


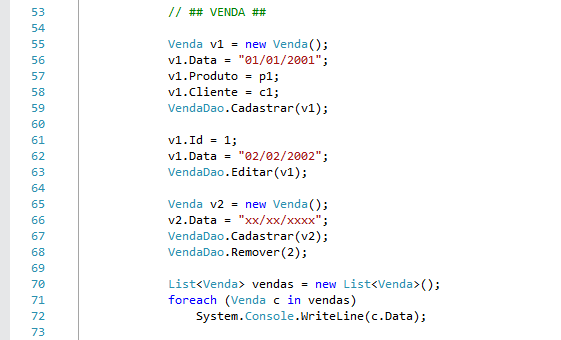


# Console APP

Abaixo vamos mostrar o código utilizado para testar o sistema através de uma segunda forma, que não é a aplicação web feita em AngularJS, pois era uma das exigencias do projeto. Portanto decidimos usar uma aplicação console para chamar os métodos do sistema para testá-los, pois uma aplicação console é bem simples de desenvolver.







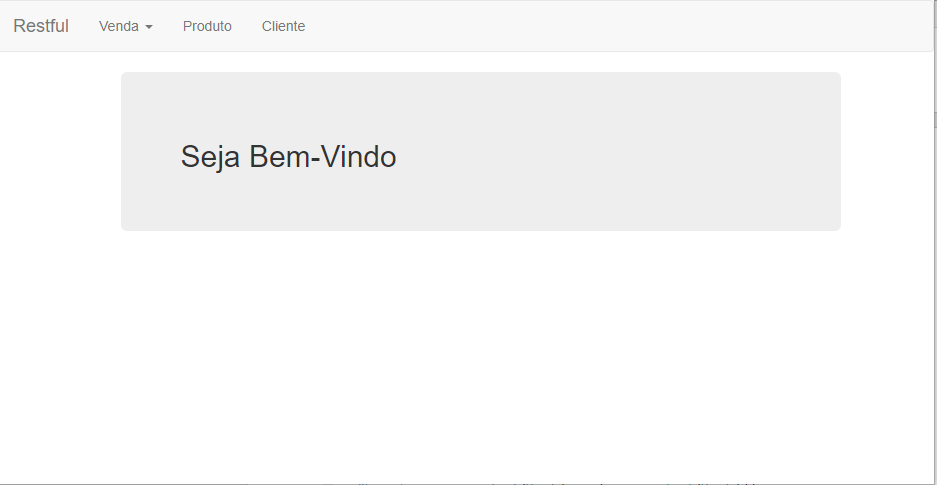
# Demonstração do Sistema

A seguir vamos ilustrar as páginas do sistema em funcionamento.

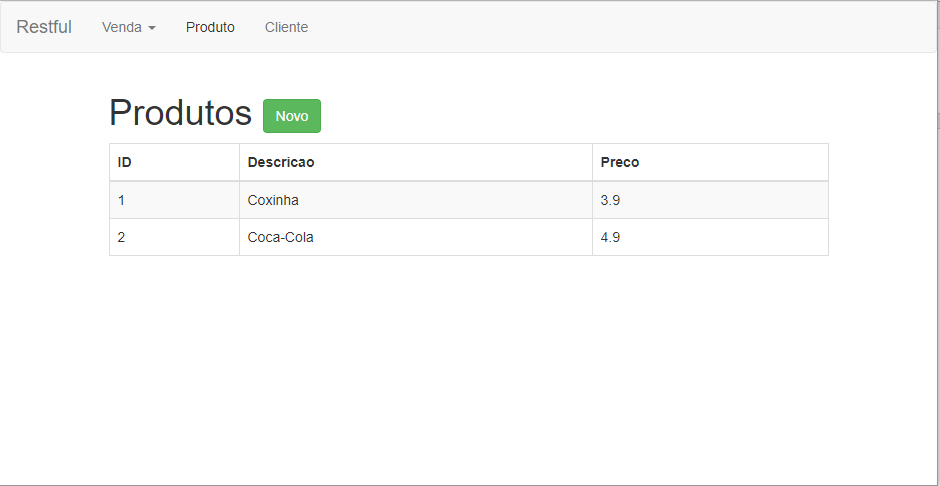
Para cliente e produto, as paginas contém uma tabela que faz a listagem das entidades já cadastradas no sistema, um botão que permite adicionar um nova entidade no sistema. Ao clicar neste botão, abrirá um popup que conterá os campos para o cadastro e botão de confirmação. Ao clicar em um item existente na tabela, também irá abrir um popup que permitirá alterar os campos de alguma entidade existente no sistema ou exclui-la permanentemente do sistema.

Para vendas, há duas paginas. Uma delas que consiste no cadastro de uma venda, que possibilita a escolha do produto e do cliente, cujas opções são as entidades cadastradas no sistema. E a segunda página é uma simples listagem de todas as vendas cadastradas no sistema.

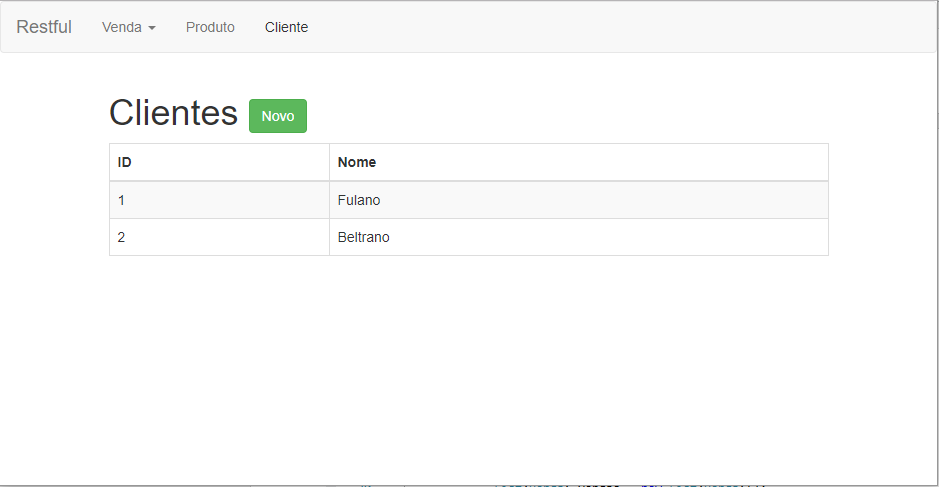
PAGINA INICIAL



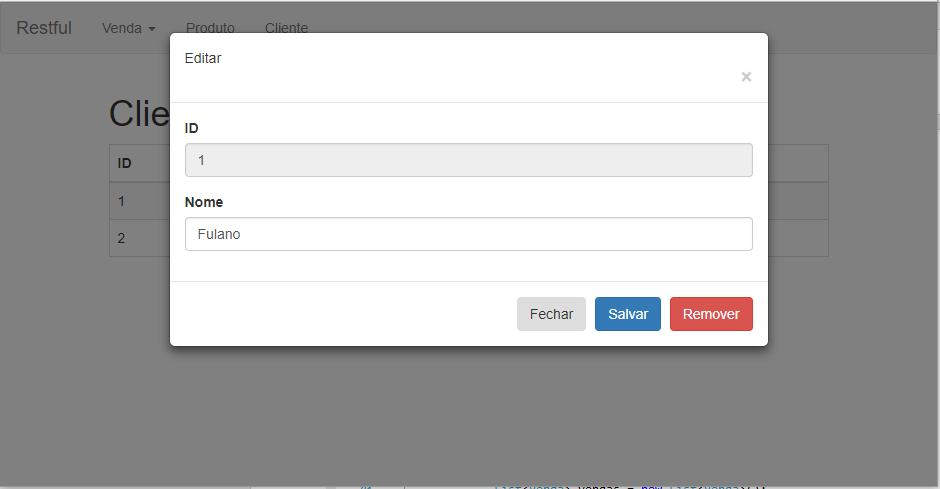
PAGINA DE PRODUTO



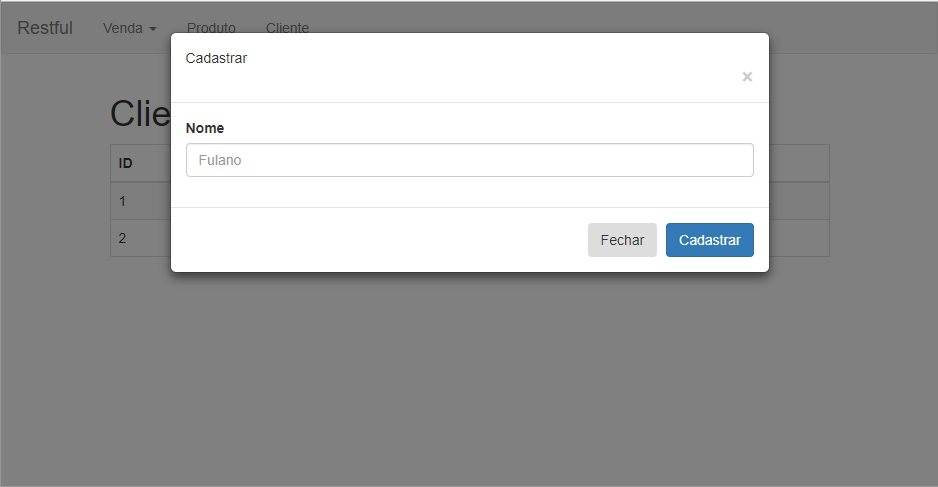
PAGINA DE CLIENTE



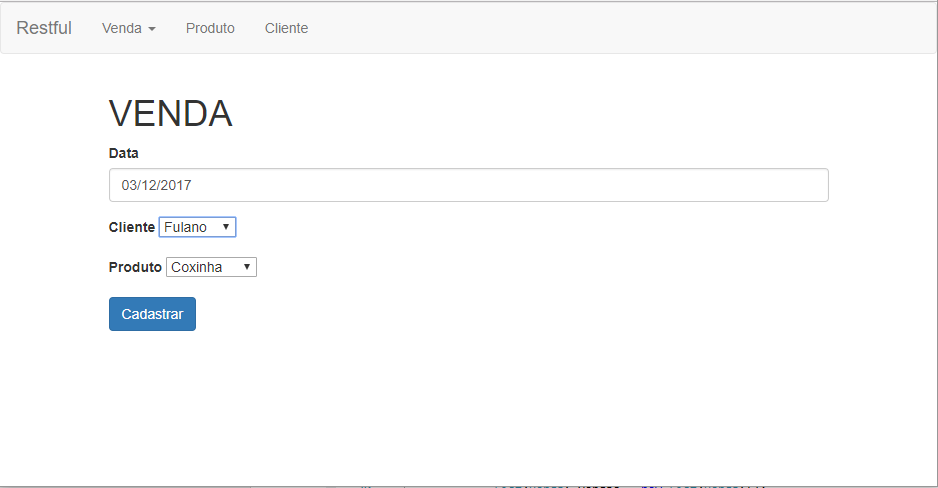
EDITAR UM CLIENTE



ADICIONAR UM CLIENTE



CADASTRO DE VENDA



LISTAGEM DE VENDAS

