



CRUD Utilizando Spring Boot e JavaFX

Neste tutorial, você aprenderá a desenvolver um aplicativo CRUD (Create, Read, Update, Delete) integrando uma API Spring Boot com uma interface gráfica JavaFX. Essa combinação permite criar aplicações robustas e interativas. Vamos começar!

1. FERRAMENTAS NECESSÁRIAS

1. [SceneBuilder](#)
2. [JavaFX](#)
3. Dependências necessárias para executar o JavaFX:

```
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-fxml</artifactId>
  <version>23-ea+3</version>
</dependency>

<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>23-ea+3</version>
</dependency>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.32</version>
</dependency>
```

O arquivo pom.xml com as dependências pode ser consultado [aqui](#)

2. MODIFICAÇÕES NECESSÁRIAS

Na classe “**ApiApplication.java**”, vamos integrar o JavaFX. Para isso, você deve estender a classe “**Application**” e importar a biblioteca JavaFX. Em seguida, implemente o método “**start**” necessário para a classe.

```
import javafx.application.Application;
import javafx.stage.Stage;

@SpringBootApplication
public class ApiApplication extends Application {

    public static ConfigurableApplicationContext context;

    @Run | Debug
    public static void main(String[] args) {
        launch();
        SpringApplication.run(primarySource:ApiApplication.class, args);
    }

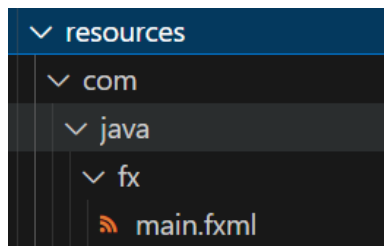
    @Override
    public void start(Stage stage) throws Exception {
        context = SpringApplication.run(primarySource:ApiApplication.class);
    }
}
```

3. CRIANDO O PROJETO NO SCENE BUILDER

Vamos começar criando uma tela chamada “Cadastro de Pessoa” no Scene Builder. Primeiro, selecione o componente “**Pane**”, que servirá como o contêiner principal onde nossa interface será desenhada. Em seguida, adicione um componente “**Label**”, que será responsável por exibir o título da nossa aplicação.



Após criar a tela, salve-a em uma pasta específica dentro do nosso projeto. O caminho para salvar o arquivo será “**resources > com > java > javafx > main.fxml**”.



4. IMPORTANDO A INTERFACE

Agora, vamos importar a interface que acabamos de criar no Scene Builder e exibi-la na nossa aplicação. Para fazer isso, utilize o código a seguir no método “start”:

```
@Override
public void start(Stage stage) throws Exception {
    context = SpringApplication.run(primarySource:ApiApplication.class);
    FXMLLoader fxml = new FXMLLoader(getClass().getResource(name: "/com/java/fx/main.fxml"));
    fxml.setControllerFactory(context::getBean);

    Scene scene = new Scene(fxml.load());
    stage.setTitle(value:"Cadastro de Pessoa");
    stage.setScene(scene);
    stage.show();
}
```

5. Testando a aplicação

Para testar a aplicação, é necessário configurar o arquivo `launch.json`, que é responsável pela execução da aplicação. Adicione o seguinte `vmArgs` conforme mostrado abaixo:

O caminho destacado em **negrito** deve ser substituído pelo local onde está salvo o seu sdk do javafx.

```
"vmArgs": "--module-path C:/javafx-sdk-20.0.1/lib --add-modules  
javafx.controls,javafx.fxml"
```

```
{  
  "type": "java",  
  "name": "ApiApplication",  
  "request": "launch",  
  "mainClass": "com.ideau.api.ApiApplication",  
  "projectName": "api",  
  "vmArgs": "--module-path C:/javafx-sdk-20.0.1/lib --add-modules javafx.controls,javafx.fxml"  
}
```

Em seguida, acesse o arquivo `ApiApplication.java` e clique em **"Run"** para executar a aplicação.

```
12 @SpringBootApplication  
13 public class ApiApplication extends Application{  
14  
15     public static ConfigurableApplicationContext context;  
16  
17     Run | Debug  
18     public static void main(String[] args) {  
19         launch(args);  
20         SpringApplication.run(primarySource:ApiApplication.class, args);  
21     }  
22 }
```

A tela exibida após a execução da aplicação deve corresponder à imagem abaixo:



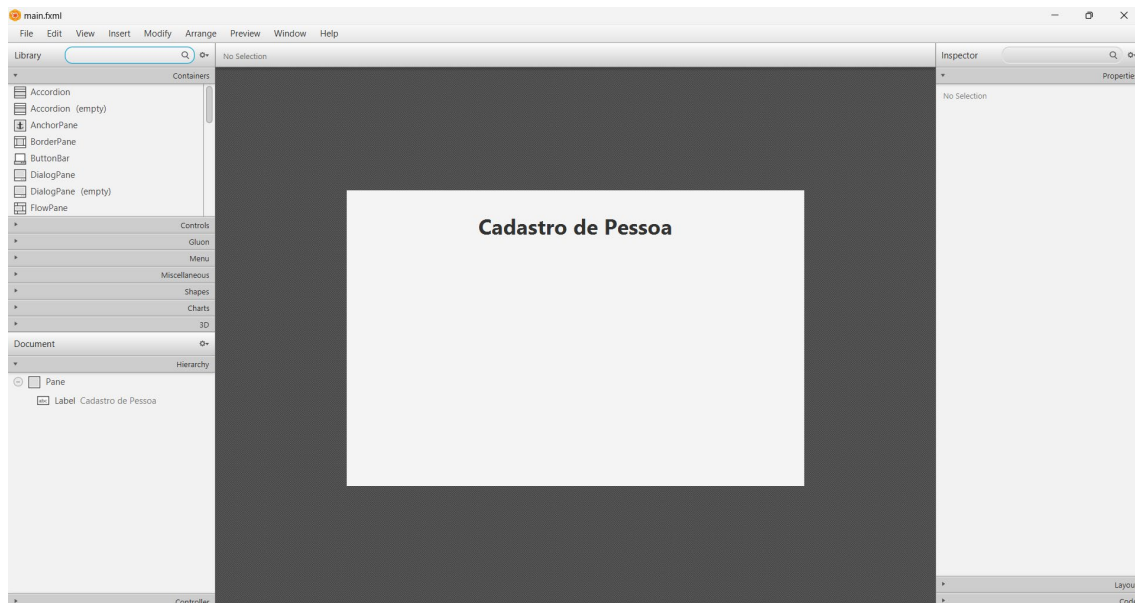
Se a tela exibida corresponder à imagem, significa que a configuração do JavaFX foi bem-sucedida.

6. FINALIZANDO A INTERFACE FRONT-END

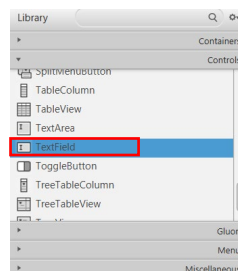
Nesta seção, vamos adicionar novos campos para inserção de textos, uma listagem de pessoas cadastradas e os botões para salvar, atualizar, deletar e excluir registros.

6.1 Adicionando os campos de textos

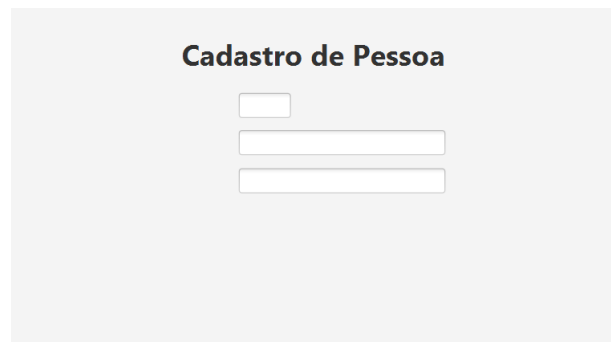
Abra o Scene Builder e carregue o nosso arquivo de interface “main.fxml”.



Com o projeto aberto no Scene Builder, vamos adicionar os campos de texto. Para fazer isso, no menu à esquerda, selecione a aba “**Controls**” e, em seguida, escolha “**TextField**”.



Para adicionar o campo de texto à nossa interface, mantenha pressionado o botão esquerdo do mouse sobre o componente “**TextField**” e arraste-o para o local desejado. Neste exemplo, vamos adicionar três “**TextFields**”, conforme mostrado na imagem abaixo:



Além disso, para identificar cada campo, vamos adicionar um componente “**Label**” acima de cada “**TextField**”. O primeiro campo será identificado como “**ID**”, o segundo como “**Nome**” e o terceiro como “**Idade**”.

Cadastro de Pessoa

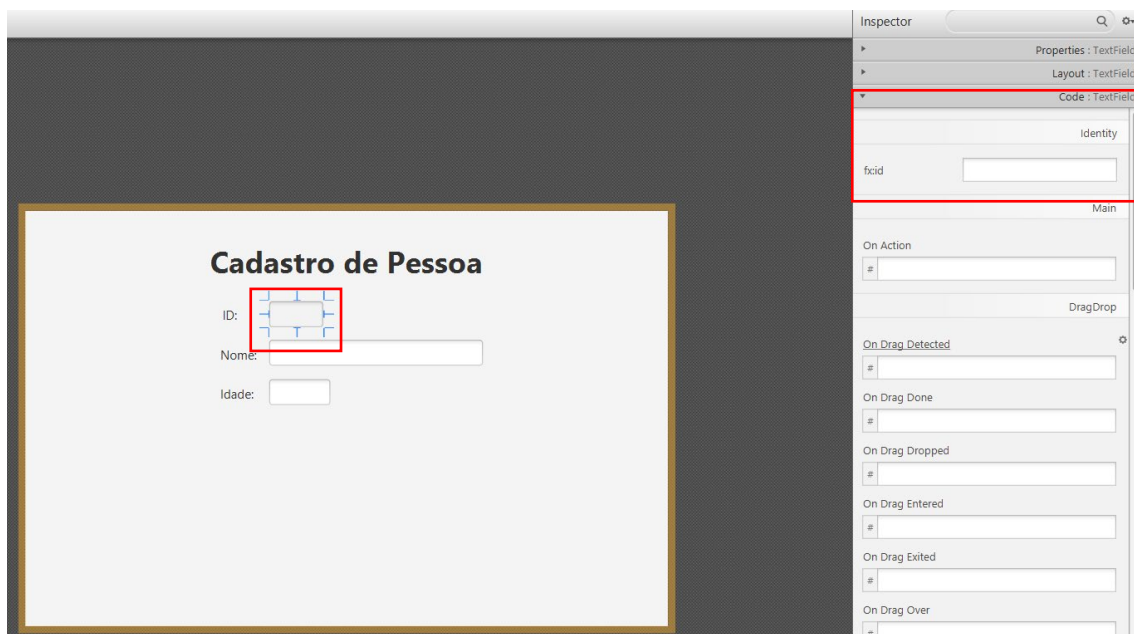
ID:

Nome:

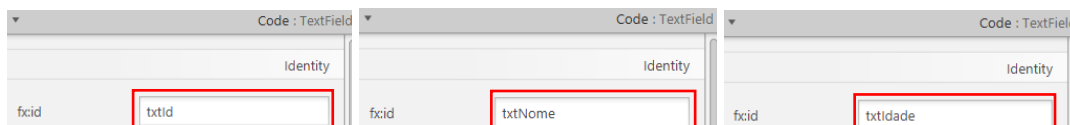
Idade:

Os campos que acabamos de adicionar precisam ser identificados para que possamos utilizá-los na API. Para fazer isso, siga os passos abaixo:

1. Clique sobre o campo que deseja identificar.
2. No menu à direita, vá até a aba **“Code”**.
3. Localize o campo chamado **“fx:id”** e atribua um nome para identificar cada campo.



Vamos começar identificando o campo “ID”, e em seguida, os campos “Nome” e “Idade”.



Após concluir essa etapa, salve o arquivo clicando em **“File > Save”**. Em seguida, execute a aplicação para verificar se está funcionando corretamente e com as alterações que realizamos. Se tudo ocorrer bem, a aplicação deverá ser exibida conforme a imagem abaixo:

Cadastro de Pessoa

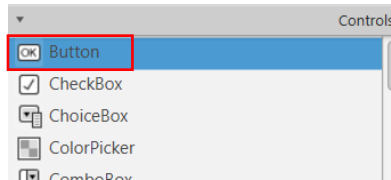
ID:

Nome:

Idade:

6.2 Adicionando os botões

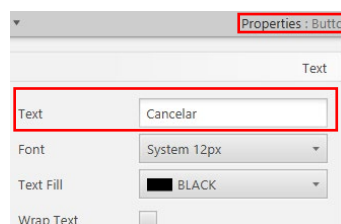
Agora, vamos adicionar os botões de salvar, atualizar, deletar e cancelar. Para fazer isso, siga os mesmos passos que fizemos anteriormente, mas desta vez selecione o componente “Button” no menu à esquerda.



Precisamos arrastar quatro botões para serem exibidos na nossa interface.



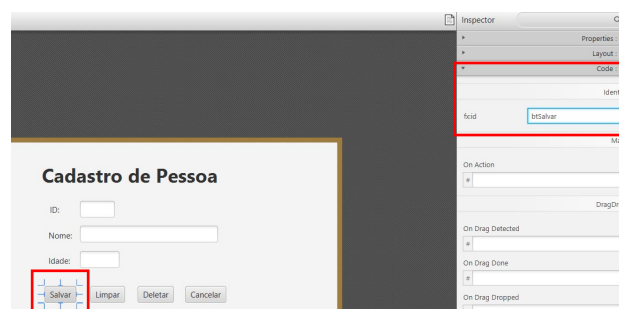
No entanto, é importante nomear os botões de acordo com a funcionalidade desejada. Para renomear os botões, vá ao menu à direita e selecione “Properties”:



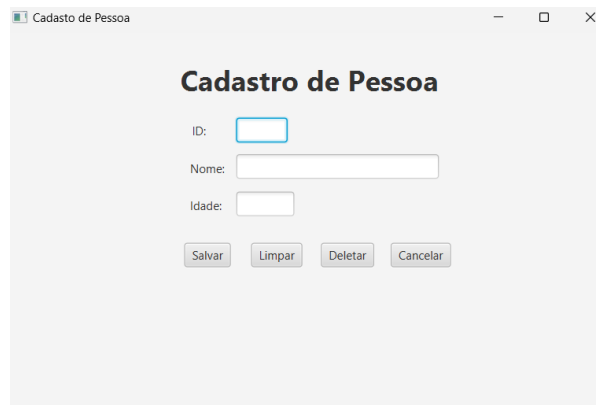
Alterado todos os nomes dos botões, deverá ficar dessa forma:



Assim como fizemos com os campos de inserção de texto, é necessário nomear os botões para que possam ser utilizados no código:



Para finalizar esta etapa dos botões, salve o arquivo e, em seguida, execute a aplicação para verificar se os botões foram adicionados corretamente e estão funcionando conforme o esperado.



A imagem mostra uma janela de aplicativo intitulada "Cadastro de Pessoa". No centro, há um formulário com o mesmo título. O formulário contém três campos de entrada: "ID:" com um campo de texto pequeno, "Nome:" com um campo de texto maior, e "Idade:" com um campo de texto pequeno. Abaixo dos campos, há uma barra contendo quatro botões: "Salvar", "Limpar", "Deletar" e "Cancelar".

7 Configuração da classe controle

Nessa seção vamos configurar o arquivo “Controle.java”, nele serão realizadas todas as operações necessárias para o CRUD.

7.1 Definindo as variáveis

Abra o arquivo “**Controle.java**” e defina algumas variáveis que serão necessárias para receber ações do nosso código. Lembre-se de adicionar as anotações “@FXML”. A imagem a seguir mostra quais são as variáveis:

```
26 @RestController
27 public class Controle {
28
29
30     @Autowired
31     private Repositorio acao;
32
33     @Autowired
34     private Servico servico;
35
36     @FXML
37     private AnchorPane anchorPane;
38
39     @FXML
40     private TextField txtId, txtNome, txtIdade;
41
42     @FXML
43     private Button btnSalvar, btnAtualizar, btnDeletar, btnLimpar;
44 }
```

7.2 Criando a função salvar

Agora, vamos criar uma função para salvar o nosso formulário por meio de uma ação no botão "Salvar".

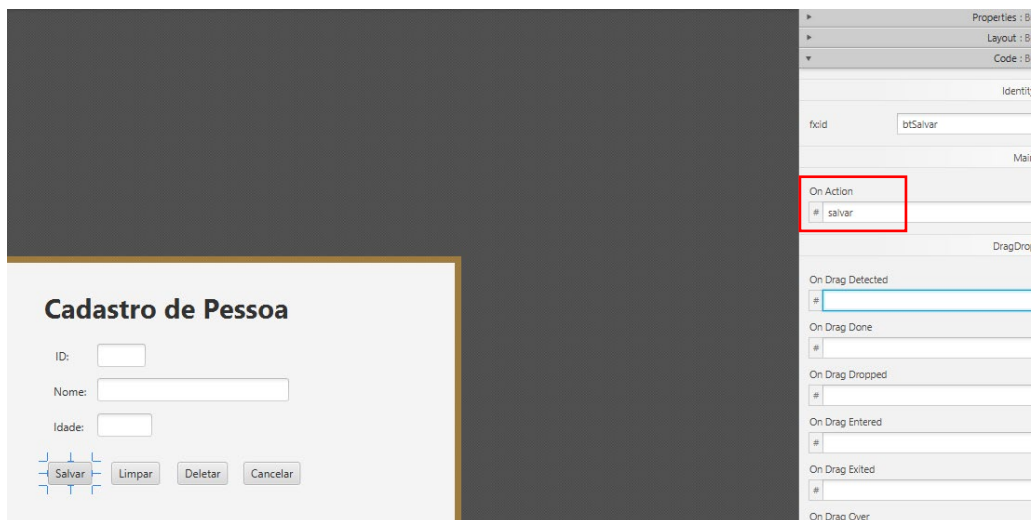
Logo abaixo da definição das variáveis, vamos implementar uma função para salvar uma nova pessoa no banco de dados. Esta função verificará inicialmente se o nome não está vazio. Se não estiver, ela criará uma nova instância de pessoa, atribuirá os valores dos campos preenchidos na aplicação e salvará no banco de dados. O Spring Boot já possui um método “**save()**” para essa finalidade.


```

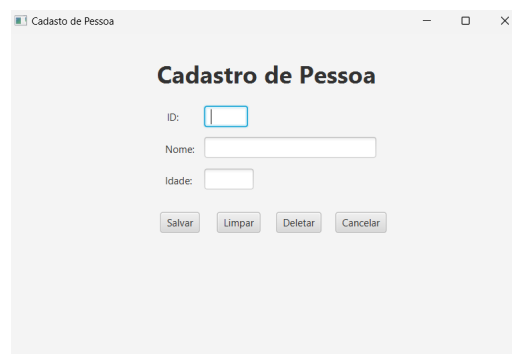
45
46
47 @FXML
48 public void salvar(){
49     if(!txtNome.getText().trim().isEmpty()){
50         Pessoa pessoa = new Pessoa();
51         pessoa.setNome(txtNome.getText());
52         pessoa.setIdade(Integer.valueOf(txtIdade.getText()));
53         acao.save(pessoa);
54     }
55 }

```

Para que o nosso botão funcione, precisamos associar essa função à interface gráfica utilizando o Scene Builder. Abra o Scene Builder e localize no menu lateral direito a aba “**Code**”. Lá, você encontrará uma opção **On Action** para associar a função que acabamos de criar ao botão “**Salvar**”.



Agora, salve novamente a interface no Scene Builder e execute a aplicação para testar a nova funcionalidade de salvar que acabamos de desenvolver.



Com a aplicação em execução, preencha os valores nos campos e, em seguida, clique no botão "Salvar". Neste momento, não é necessário inserir um valor no campo "ID", pois ele é gerado automaticamente.



Para confirmar que as informações foram salvas no banco de dados, acesse a tabela “**pessoa**” no MySQL e verifique se a tabela foi atualizada com essa nova informação.

DATABASE

127.0.0.1@3306 8.0.34

api_spring 32k

api_tutorial 16k

Query

Tables (1)

pessoas 5

columns

codigo int

idade int

nome varchar(255)

index

partitions

Views

Controle.java 4

ApiApplication.java

pessoas

Properties

DATA

Monitor

SELECT * FROM pessoas LIMIT 100

Search results

1

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

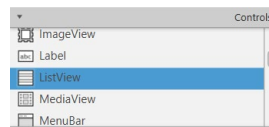
+</

Se o registro constar na tabela, significa que nossa codificação foi bem-sucedida.

Para evitar ter que abrir o banco de dados toda vez para verificar se as informações foram armazenadas, vamos adicionar uma funcionalidade de listagem na nossa aplicação. Assim, conseguiremos visualizar todas as informações cadastradas diretamente na interface.

7.3 Criando a função listar

Antes de começar a codificar a função de listagem, devemos adicionar uma lista na interface da nossa aplicação. Para fazer isso, abra a interface gráfica no Scene Builder. Com a interface carregada, adicione o componente “**ListView**”, que está localizado no menu lateral esquerdo (no mesmo local onde adicionamos os outros componentes).



Nossa aplicação deve ficar semelhante a imagem abaixo:



Cadastro de Pessoa

ID:

Nome:

Idade:

Na próxima etapa, vamos adicionar a função “**listar()**” na nossa classe Controle. Abra o arquivo “**Controle.java**” e adicione a variável **list**, que será responsável por receber a nossa lista de cadastros.

```

44
45     @FXML
46     private Button btnSalvar, btnAtualizar, btnDeletar, btnLimpar;
47
48     @FXML
49     private ListView<Pessoa> list;

```

Em seguida, abaixo da função salvar, vamos adicionar nossa função listar.

```

61
62     public void listar() {
63         list.setItems(FXCollections.observableArrayList(acao.findAll()));
64     }
65

```

No entanto, apenas criar a função “**listar()**” não será suficiente para exibir os cadastros. Precisamos chamá-la em algum momento. É importante que sempre que a aplicação for iniciada, a lista de cadastros seja carregada. Para isso, vamos adicionar um “**implements**” na nossa classe Controle.

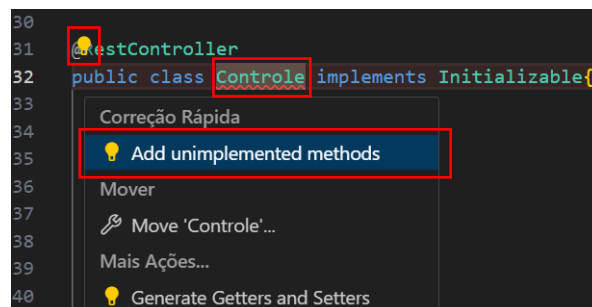
No início da classe Controle, adicione o “**implements Initializable**”.

```

30
31     @RestController
32     public class Controle implements Initializable{
33

```

Observe que o nome da classe Controle está sublinhado em vermelho. Isso ocorre porque o Initializable exige a implementação de um método específico. Clique sobre a classe Controle e, em seguida, na lâmpada que aparece. Selecione a opção “**Add unimplemented methods**” para adicionar o método necessário.



```

30
31     @RestController
32     public class Controle implements Initializable{
33

```

Será criado um método no final do arquivo:

```

163
164     @Override
165     public void initialize(URL location, ResourceBundle resources) {
166         // TODO Auto-generated method stub
167         throw new UnsupportedOperationException(message:"Unimplemented method 'initialize'");
168     }

```

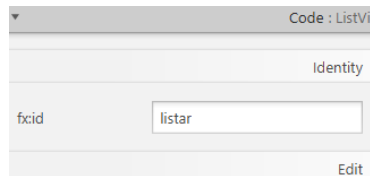
Vamos apagar o que tiver dentro dessa função e deixar somente nossa função listar:

```

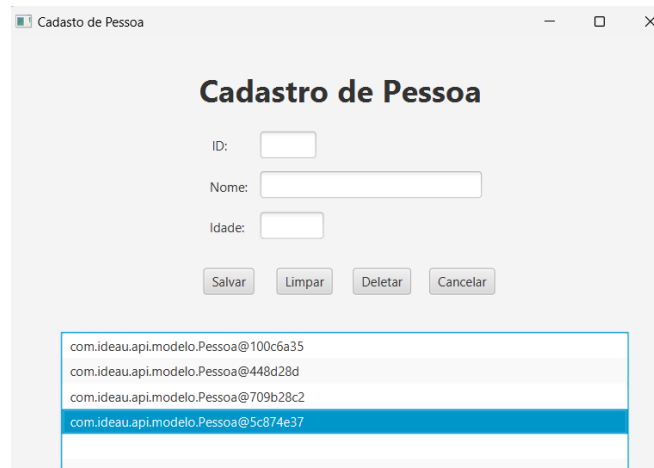
163
164     @Override
165     public void initialize(URL location, ResourceBundle resources) {
166         listar();
167     }

```

Ainda falta uma configuração, adicionar no Scene Builder essa função de listar, da mesma forma que foi adicionado na subseção anterior.



A nossa aplicação deverá parecer semelhante à imagem abaixo. No entanto, observe que o retorno está sendo exibido, mas não da forma como gostaríamos.



Para exibir as informações de cadastro em um formato desejado, precisamos fazer uma modificação na classe **“Pessoa.java”**, criando o método **“toString()”**. Vamos acessar a classe Pessoa e realizar as seguintes modificações no final do arquivo.

```
38
39 public String toString(){
40     return "ID: " + codigo + " - Nome: " + nome + " - Idade: " + idade;
41 }
```

Após essa modificação, salvar o arquivo e executar novamente a aplicação. O retorno deve ficar igual a imagem a seguir:



Com as partes de listagem e salvar prontas, vamos agora para a parte de deletar.

7.4 Desenvolvendo a função de deletar

Primeiramente, vamos ter que criar duas variáveis, no início do arquivo:

```

34     private Pessoa item;
35
36     private int id;

```

Em seguida, devemos criar uma função “item”, que será responsável por buscar os cadastros no banco de dados

```

167     public Pessoa item(){
168         return acao.findById(id).get();
169     }

```

Agora, vamos atualizar a função initialize para selecionar e substituir as informações nos campos quando um item da lista for clicado:

```

170
171     @Override
172     public void initialize(URL location, ResourceBundle resources) {
173         listar();
174         list.getSelectionModel().selectedItemProperty().addListener((obs, old, newValue) -> {
175             if (newValue != null){
176                 id = newValue.getCodigo();
177                 item = item();
178                 txtId.setText(String.valueOf(item.getCodigo()));
179                 txtNome.setText(item.getNome());
180                 txtIdade.setText(String.valueOf(item.getIdade()));
181             }
182         });
183     }

```

Executando novamente a aplicação, ao clicar em algum dos cadastros, vamos obter o seguinte retorno:

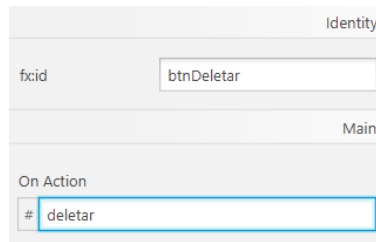
Agora vamos criar a função “deletar”. Podemos adicioná-la logo abaixo da função “listar”:

```

71     public void delete(){
72         acao.delete(item);
73         listar();
74     }

```

Para que o botão funcione, é necessário adicionar a ação no botão. Abra novamente o Scene Builder e adicione a ação da mesma forma que foi feito com o botão “Salvar”: acesse o menu lateral direito, vá até a aba “Code” e passe o nome da função em “On Action”. Não se esqueça de salvar a interface após as alterações.



Agora sim, podemos executar novamente a aplicação e testar se o botão **“Deletar”** vai funcionar e deletar algum dos cadastros.



Podemos observar na imagem acima que o cadastro de código 2 foi excluído, indicando que nossa aplicação está funcionando para inserir, listar e deletar. Agora, vamos configurar a funcionalidade de atualização dos dados cadastrados.

7.5 Criando a função de editar

Logo abaixo da função deletar, vamos criar a função editar. O código que deverá ser digitado está na imagem abaixo:

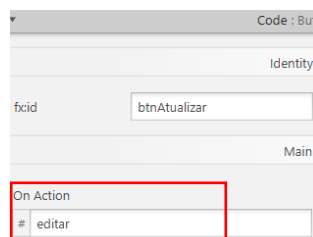
```

76  @FXML
77  public void editar(){
78      item.setNome(txtNome.getText());
79      item.setIdade(Integer.valueOf(txtIdade.getText()));
80      acao.save(item);
81      listar();
82  }

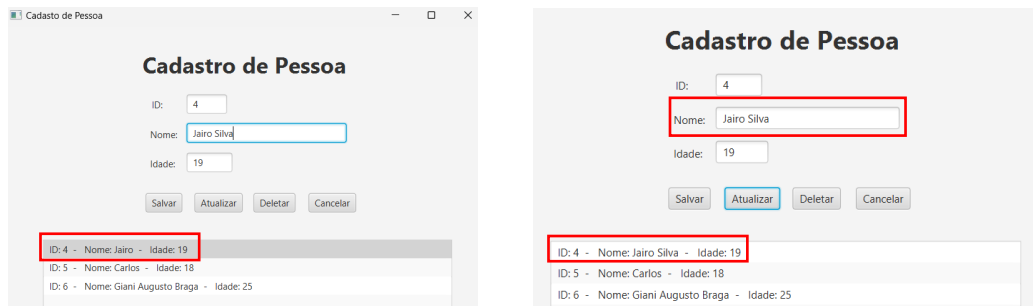
```

Notem que a função vai definir o novo valor de acordo com o **“item”** que foi clicado, e em seguida, ele é salvo pela função **“save”**, passando como parâmetro o **“item”** que foi modificado.

Agora, devemos associar a função editar ao botão **“Atualizar”**. Acesse o Scene Builder, clique no botão **“Atualizar”** e no menu lateral direito, adicione a função editar no campo **“On Action”**.



Agora, salve as alterações e execute novamente a aplicação para testar se o botão “Atualizar” já está funcionando corretamente.



Note que na primeira imagem o nome constava como “Jairo”. Foi adicionado o sobrenome “Silva” e clicado em “Atualizar”, demonstrando que nossa aplicação está funcional.

Finalizadas as funções de salvar, atualizar, deletar e listar. Na próxima seção, iremos fazer alguns ajustes finais na aplicação.

8 AJUSTES FINAIS

Nesta seção, vamos realizar alguns ajustes na aplicação, como implementar a função de limpar os campos e habilitar/desabilitar apenas os botões necessários.

8.1 Criar a função limpar

Vamos implementar uma função para limpar os campos “ID”, “Nome” e “Idade” sempre que uma operação de salvar, atualizar ou deletar for realizada.

Logo abaixo da função **editar**, adicionaremos a função **limpar**. Além de limpar os campos, vamos definir um padrão para o comportamento dos botões. Assim, toda vez que essa função for chamada, deixaremos apenas o botão “Salvar” habilitado.

```
84      @FXML
85      public void limpar() {
86          txtId.clear();
87          txtNome.clear();
88          txtIdade.clear();
89
90          btnSalvar.setVisible(value:true);
91          btnCancelar.setVisible(value:false);
92          btnAtualizar.setVisible(value:false);
93          btnDeletar.setVisible(value:false);
94      }
```

Podemos chamar essa função de **limpar**, dentro de outras funções. Vamos começar colocando-a dentro da função **deletar**:

```
71      public void deletar(){
72          acao.delete(item);
73          limpar();
74          listar();
75      }
```

Agora podemos adicionar na função **editar**:

```
77 @FXML
78 public void editar(){
79     item.setNome(txtNome.getText());
80     item.setIdade(Integer.valueOf(txtIdade.getText()));
81     acao.save(item);
82     limpar();
83     listar();
84 }
```

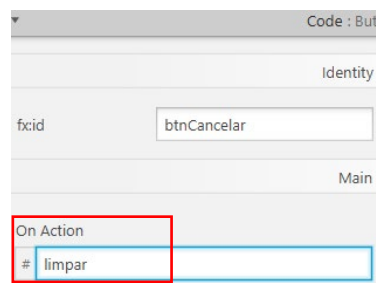
Por fim, devemos adicionar na função **salvar**:

```
56 @FXML
57 public void salvar() {
58     if (!txtNome.getText().trim().isEmpty()) {
59         Pessoa pessoa = new Pessoa();
60         pessoa.setNome(txtNome.getText());
61         pessoa.setIdade(Integer.valueOf(txtIdade.getText()));
62         acao.save(pessoa);
63         limpar();
64     }
65 }
```

8.2 Adicionar a função limpar no botão cancelar

Quando clicar em algum registro e decidir não prosseguir, podemos clicar em “Cancelar” e o sistema irá chamar a função limpar, voltando ao padrão.

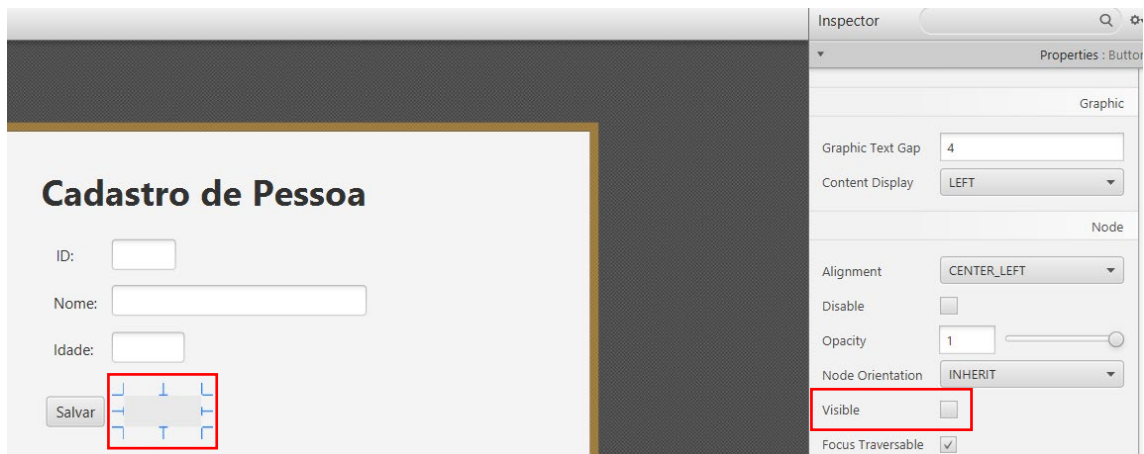
No Scene Builder, clique no botão “**Cancelar**”, acesse o menu lateral direito e adicione a função “**limpar**” em “**On Action**”:



Após salvar as alterações, você pode testar a aplicação executando-a, clicando em um cadastro e, em seguida, no botão “**Cancelar**”.

8.3 Configurando a visibilidade dos botões

Vamos ajustar a visibilidade dos botões quando a aplicação for iniciada. Para fazer isso, acesse o Scene Builder e configure a visibilidade dos botões, desmarcando a opção “**Visible**” no menu lateral direito, na aba “**Properties**”.



Assim, quando nossa aplicação iniciar, apenas o botão “**Salvar**” será exibido.

Agora, vamos editar o código para mostrar os botões que estão ocultos quando um dos cadastros da lista for selecionado.

```

205 @Override
206 public void initialize(URL location, ResourceBundle resources) {
207     listar();
208     list.getSelectionModel().selectedItemProperty().addListener((obs, old, newValue) -> {
209         if (newValue != null){
210             id = newValue.getCodigo();
211             item = item();
212             txtId.setText(String.valueOf(item.getCodigo()));
213             txtNome.setText(item.getNome());
214             txtIdade.setText(String.valueOf(item.getIdade()));
215
216             btnSalvar.setVisible(value:false);
217             btnCancelar.setVisible(value:true);
218             btnDeletar.setVisible(value:true);
219             btnAtualizar.setVisible(value:true);
220         }
221     });
222 }
223

```

Quando clicar em “**Cancelar**”, a aplicação deve retornar ao padrão, exibindo apenas o botão “**Salvar**” habilitado. Para isso, vamos atualizar o código da função limpar:

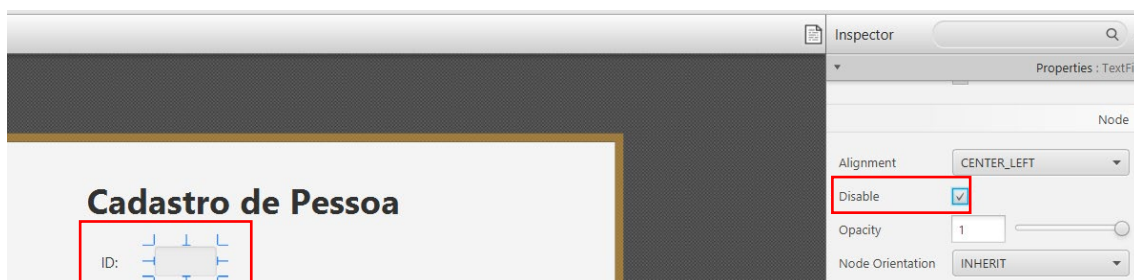
```

87 @FXML
88 public void limpar() {
89     txtId.clear();
90     txtNome.clear();
91     txtIdade.clear();
92
93     btnSalvar.setVisible(value:true);
94     btnCancelar.setVisible(value:false);
95     btnAtualizar.setVisible(value:false);
96     btnDeletar.setVisible(value:false);
97
98     btnSalvar.setVisible(value:true);
99     btnCancelar.setVisible(value:false);
100     btnAtualizar.setVisible(value:false);
101     btnDeletar.setVisible(value:false);
102 }

```

Para finalizar, vamos desabilitar o campo de ID, assim evitamos que seu valor seja alterado e ele ficará visível apenas quando clicarmos para atualizar.

Acesse o Scene Builder, clique sobre o campo “ID” e, em seguida, no menu lateral direito, na aba “Properties”, marque a opção “Disable”.



Com isso, finalizamos o desenvolvimento da nossa aplicação utilizando Spring Boot e JavaFX. O código completo está disponível no github, [clikando aqui](#).