

Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring



BitBank.01 - Spring MVC Introdução

por Frederico C. G. Pereira

© Copyright 2025

Objetivos

- Criar um projeto Maven no VSCode que use um container JEE
- Utilizar e configurar o Spring MVC
- Criar controladores Spring simples
- Importar classes de entidades do modelo de objetos do banco já prontas

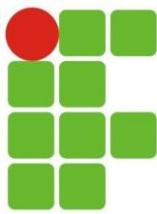
ATENÇÃO: Não copie e cole os trechos de código, DIGITE-OS VOCÊ MESMO! Isto faz uma gigantesca diferença para o aprendizado e é uma recomendação sempre encontrada em livros de programação. Sempre após digitar, certifique-se de que entendeu o que você fez, do contrário a prática é apenas um exercício de Ctrl C + Ctrl V. Siga corretamente os passos neste roteiro. Obedeça às versões indicadas do que deve ser baixado e não baixe alternativas mais recente porque pode quebrar seu código. Faça estas experiências depois de completar a prática.

Certifique-se de ter a versão Java 17 ou superior esteja instalada em sua máquina.

I. Criação e configuração do projeto no VSCode e Maven

1) Crie um projeto Java/Maven chamado **bitbank**.

- a) Antes de começar a criar o projeto, leia a letra (b) completamente e copie, com vários Ctrl+C, todos os dados que precisará digitar quando estiver criando o projeto. O VSCode aborta a criação se você sair da janela dele para copiar algum dado. Use a combinação + V para colar o que copiou.
 - i) Vou indicar na letra (b) o que você deve copiar com o ícone (**cuidado para não copiar o ícone junto!**).
- b) No VSCode, digite **F1** (para ir para a barra de comandos)
 - i) Digite **Create Java Project**
 - ii) Selecione Spring Boot
 - iii) Selecione Maven Project > 3.1.0 > Java
 - iv) Digite o GroupId **br.edu.ifpb.pweb2**
 - v) Digite o ArtifactId: **bitbank**
 - vi) Selecione **Jar** > **17** (obrigatório para o Spring 3+)
 - vii) Digite **Spring Web** e selecione a primeira opção de starter. Este *starter* contém o núcleo do Spring e suas extensões para desenvolvimento web e de *web services*.



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

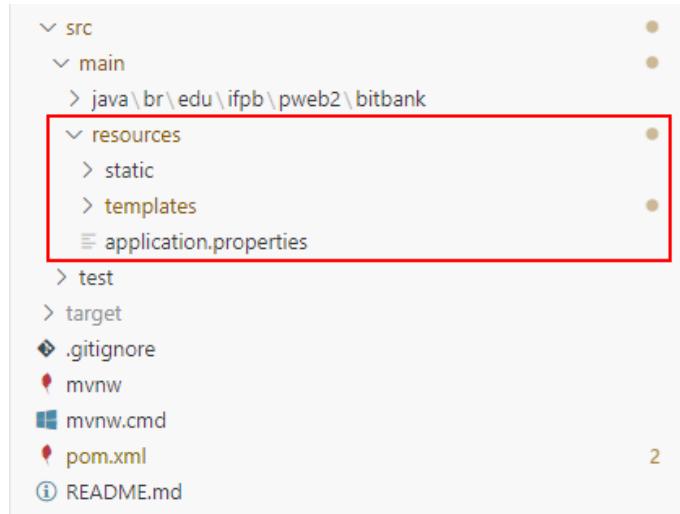
Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

- viii) Digite **DevTools** e selecione o Spring Boot DevTools. Este *starter* contém uma ferramenta que reinicia automaticamente o servidor de aplicação sempre que você alterar alguma classe ou outro arquivo do projeto.
 - ix) Digite **Thymeleaf** e selecione a opção apresentada. Este *starter* define que a *engine* de processamento de páginas dinâmicas será o Thymeleaf.
 - x) Digite **Lombok** e selecione a única opção exibida. Este *starter* permite eliminar muito de código repetitivo (*boiler plate*) das classes Java simples. Métodos como get's e set's, construtores, `toString()`, `hashCode()` e `equals()` que geralmente "poluem" as classes de entidades não precisam ser implementadas com o uso desta biblioteca extra. Veremos caso de uso dela adiante.
 - xi) Tecle Enter apenas para encerrar a definição de Starters do Spring Boot.
 - xii) Selecione a pasta onde o seu projeto será criado.
 - xiii) A partir da próxima prática não daremos mais detalhes do passo-a-passo para criação de um projeto Spring Boot no VSCode, apenas os parâmetros.
- 2) Abre o pom.xml e examine as dependências geradas a partir dos *starters* escolhidos.

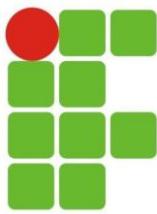
II. Estrutura do projeto e pastas protegidas e acessíveis por URL

Um projeto criado para o Spring Boot com Maven possui pastas e um arquivo que só existem para o Spring Boot.



A pasta **resources** (presente em todo projeto Maven) possui 2 subpastas e um arquivo que são próprios do Spring Boot:

- **static**: todos os arquivos de recursos (sem ser código-fonte) estáticos, isto é, arquivos que não terão nenhuma alteração específica para uma requisição. Exemplo: o arquivo `index.html` feito pela equipe de design ou imagens, folhas de estilo e códigos Javascript a serem usados pelo frontend.
- **templates**: arquivos que possuem elementos de processamento que, após executados por um engine qualquer receberão dados e serão finalizados. Exemplo: uma página que lista todos os alunos matriculados em uma turma.



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

- **application.properties**: arquivo para configuração de vários aspectos da aplicação Spring. O nome da aplicação a ser usado nas URLs, a porta onde o container executará etc.

Arquivos na pasta **static** são URL-endereçáveis, ou seja, uma vez a aplicação rodando, você pode digitar uma URL e acessá-los de um navegador. A URL é na forma

`http://<servidor>:<porta>/<webapp>/<recurso-estático>`. O nome do arquivo estatico vai logo depois do nome da aplicação web (caso tenha sido definido).

- 1) Crie um arquivo chamado index.html na pasta **static**. O conteúdo é livre.
- 2) Suba sua aplicação clicando no Run do Run | Debug sobre o método main da classe de aplicação.
- 3) Caso tenha dado erro na subida do Tomcat/Spring, informando que a porta 8080 já está sendo usada (algo bem comum nos nossos labs). O erro se apresenta assim:

```
*****
APPLICATION FAILED TO START
*****
```

Description:

```
Web server failed to start. Port 8080 was already in use.
```

Action:

Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

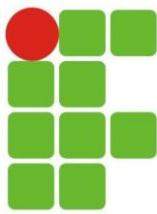
Para corrigir este erro, adicione a seguinte linha no application.properties:

server.port = 8090

- 4) Acesse o arquivo index.html com a URL `http://localhost:8080/index.html` (ou `http://localhost:8090/index.html`, caso tenha mudado a porta). Doravante nesta prática, sempre usaremos 8080 para a porta, mas saiba que você deve trocá-la caso tenha definido outra no passo anterior.
- 5) Crie um arquivo com nome **protegido.html** na pasta **templates** de conteúdo livre. Tente acessá-lo com URL similar à da questão anterior. O que houve?

III. Primeiro controlador Spring

- 1) Clique com o botão direito no último nome do pacote (bitbank) dentro da pasta src/main/java e escolha New Java File > Class....



Instituto Federal da Paraíba

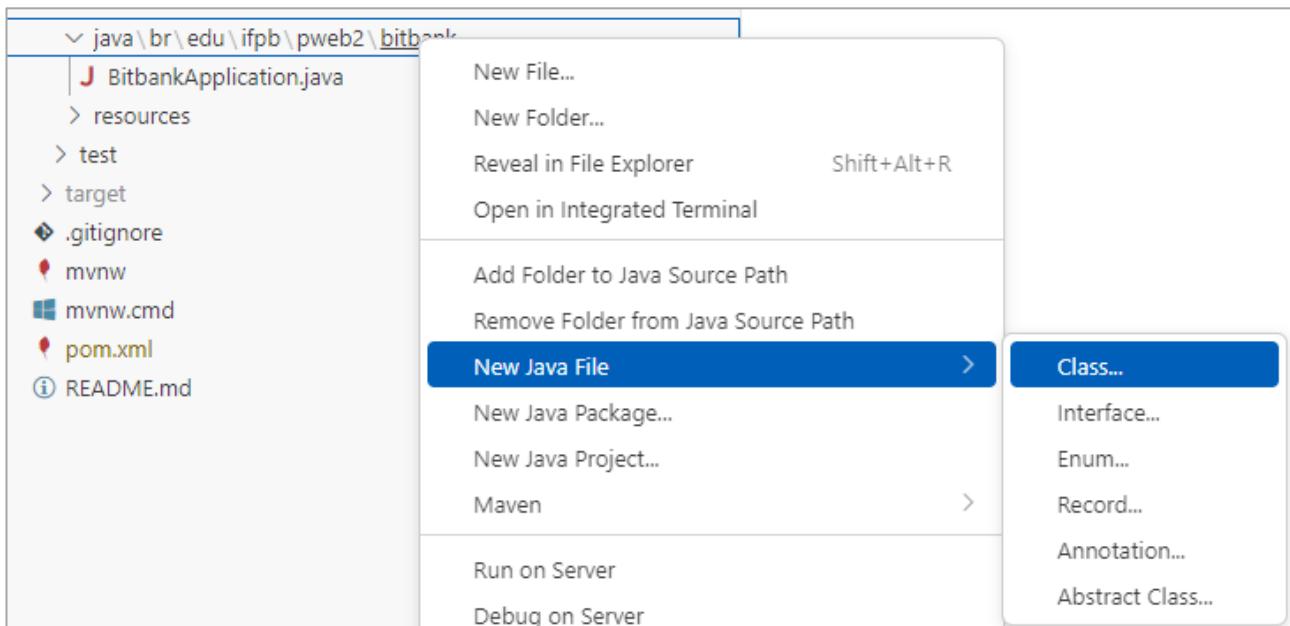
Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

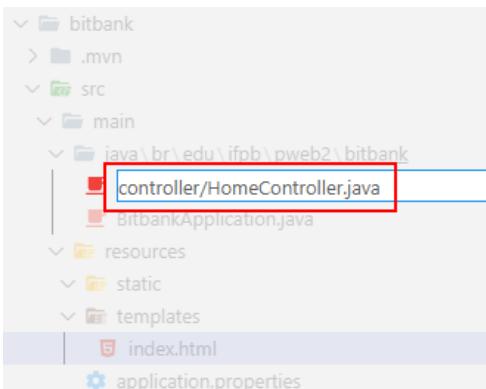
Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring



- 2) Digite **controller.HomeController** e tecle ENTER. Um novo subpacote **controller** e uma classe **HomeController .java** devem ter sido criados:



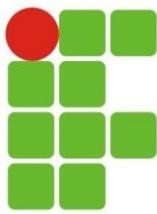
- 3) O código da classe deve ser digitado conforme abaixo (use Shift+Alt+O para inserir os imports necessários ou clique na palavra sublinhada em vermelho e depois na pequena lâmpada para acessar a opção de gerar imports):

```
package br.edu.ifpb.pweb2.bitbank.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

    @RequestMapping("/home")
    public String showHomepage() {
        return "index";
    }
}
```



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

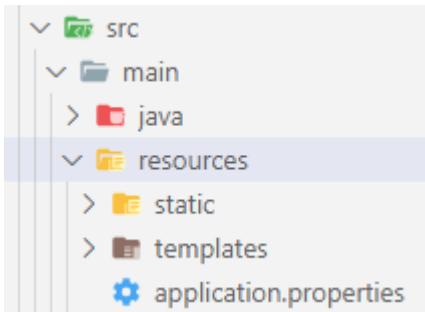
Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

Na pasta `src/main/java/resources` há duas pastas: **templates** e **static**. Em **templates** ficarão as páginas dinamicamente geradas (que chamaremos a partir de agora de *templates Thymeleaf*) enquanto em **static** ficarão arquivos de recursos definidos pelo programador (CSS, JS, imagens etc). A pasta **src** do seu projeto deve ficar como a figura a seguir:



- 4) Dentro de **templates** crie o primeiro documento Thymeleaf.

- Clique com o botão direito em **templates** e escolha New File.... Digite **index.html**.
- Dentro do arquivo **index.html**, digite `html:5` como atalho para o VSCode gerar um esqueleto de documento HTML para você. Isto poupa digitação. Edite o arquivo **index.html** para que fique como a seguir:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Página Inicial da Aplicação</title>
</head>
<body>

<h1>Bitbank no ar!</h1>

</body>
</html>
```

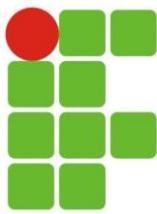
- 5) No arquivo **application.properties** localizado em `src/main/java/resources`, adicione a linha:

```
server.servlet.context-path=/Bitbank
```

Esta linha informa que se deve usar o nome **bitbank** como nome do contexto (ou aplicação web). Este nome é aquele que vem logo após `http://localhost:8080/` nas URLs. Ex: `http://localhost:8080/bitbank`

- Na classe `BitbankApplication.java` clique na opção **Run**. Você também pode, a partir de qualquer classe Java do projeto, teclar **Ctrl+F5** (ou **F5** para **Debug**).
- Teste a aplicação digitando a URL: `http://localhost:8080/bitbank/home`. A página HTML `index.html` em **templates** deverá ser exibida.

Vejamos o que aconteceu após o navegador usar a URL digitada: ele enviou uma requisição HTTP para o Tomcat® que a recebeu por meio do seu servlet que gerencia o Spring MVC (classes de configuração). Ao analisar a URL, o servlet *front controller* do Spring viu que havia uma “rota” chamada **home** logo após o nome da aplicação na URL. O Spring MVC tratou de procurar uma classe no pacote



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

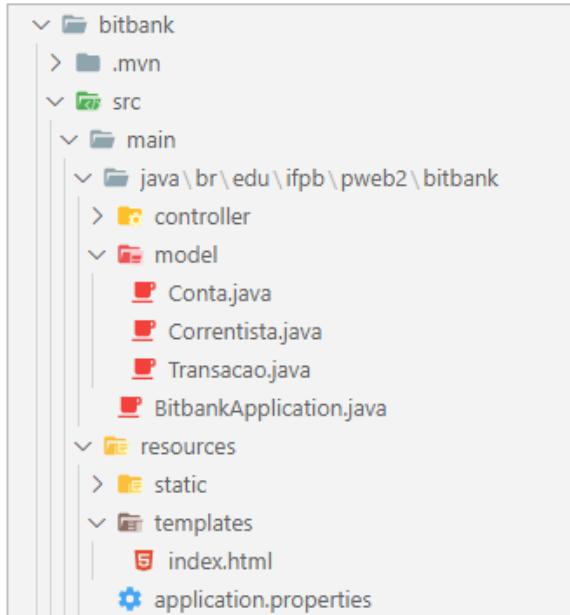
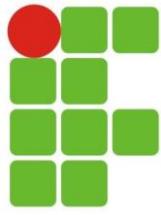
br.edu.ifpb.pweb2.bitbank.controller que possuísse um método com um `@RequestMapping` configurado com o valor “home” e encontrou a classe `HomeController` e mais especificamente o método `showHomepage()`. Este método simplesmente devolve a página `index.html` processada pelo Thymeleaf (que não tem muito o que processar porque é uma página HTML pura, sem expressões dinâmicas). E onde o Thymeleaf encontra a página `index.html`? Na pasta `src/main/resources/templates`, usada por convenção pelo Spring (você pode mudar esta convenção, mas para isto tem que escrever configuração; este princípio é o chamado **convenção sobre configuração**: siga a convenção do seu framework e tenha menos trabalho configurando coisas para ele). Isto é o básico que o framework Spring MVC faz, intercepta requisições, analisa URLs e redireciona para o método controlador da classe controladora específica que faz o serviço (ou chama uma classe de negócio de faz).

IV. Definindo classes de entidade do domínio da aplicação

Esta prática é uma de uma série que pretende implementar um sistema bancário bastante simples. Num sistema deste tipo, temos classes Java que representam as entidades do domínio da aplicação. O domínio da nossa aplicação é o sistema bancário, portanto teremos que implementar classes que modeleem entidades (reais e abstratas) como conta bancária, transação, cliente, gerente etc. Estas classes formarão nosso modelo de objetos do mundo que queremos informatizar, representarão objetos deste domínio e terão relacionamentos umas com as outras. Posteriormente, iremos persistir os objetos destas classes em tabelas de um banco de dados, momento este que em elas tornar-se-ão entidades da JPA. Nós já fornecemos estas classes prontas. Tudo que você precisa fazer é baixar o arquivo de recursos desta prática e copiar estas classes para o pacote **br.edu.ifpb.pweb2.bitbank.model**.

- 1) Acesse o site do classroom na seção Prática de Programação BitBank 01 e baixe o arquivo **prática01-model.zip** (é o primeiro link do OneDrive) e descompacte-o em alguma pasta fora da pasta do projeto. O resultado da descompactação deve ter 3 arquivos .java.
- 2) Clique no pacote **bitbank** com o botão direito e selecione **New Java Package....** O VSCode abre um campo no topo da janela para pedir o nome do subpacote. Digite **model** no final do nome do pacote
- 3) Copie as 3 classes do arquivo zip para dentro do pacote **model**. Você pode selecionar as 3 e copiá-las no Windows Explorer e colá-las dentro do pacote **model** no VSCode.
- 4) Abra cada uma das classes e observe suas propriedades e como estão relacionadas umas com as outras. O modelo é muito simples, pois o foco é aprendermos Spring e não orientação a objetos ou JPA.
- 5) Observe a anotação **@Data** do **Lombok** acima do nome da classe, que elimina a necessidade de implementarmos get's, is's e set's, além de construtores completos e default, métodos `hashCode()`, `equals()` e `toString()`. Se quiser conhecer mais sobre o Lombok depois acesse <https://projectlombok.org/features/all>.

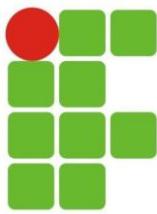
A esta altura, seu projeto deve ter a seguinte estrutura de pastas e arquivos:



V. Formulários e *Data Binding*

Como ainda não temos como saber acessar dados em um banco relacional, precisaremos de classes que simulem repositórios de dados usando listas em memória. Utilizaremos nesta prática um repositório para a classe Correntista.

- 1) Crie um pacote chamado br.edu.ifpb.pweb2.bitbank.repository.
- 2) Baixe do classroom o arquivo de recursos prática-01-repository.zip (segundo link para o OneDrive) e copie a classe **CorrentistaRepository** para o pacote repository criado anteriormente.
- 3) Crie a pasta src/main/java/resources/templates/correntistas e dentro dela o formulário **form.html**:



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

```
<html xmlns:th="www.thymeleaf.org">

    <head>
        <meta charset="UTF-8">
        <title>Cadastro de Correntistas</title>
    </head>

    <body>
        <h1>Cadastro de Correntista</h1>

        <div th:text="${mensagem}" style="color: #red"></div>

        <form th:action="@{/correntistas/save}" method="POST" th:object="${correntista}">
            <input type="hidden" th:field="*{id}" />
            Nome: <input type="text" th:field="*{nome}" /><br /><br />
            Senha: <input type="password" th:field="*{senha}" /><br /><br />
            Email: <input type="text" th:field="*{email}" /><br /><br />
            Admin? <input type="checkbox" th:field="*{admin}" /><br /><br />
            <input type="submit" value="Salvar" />
        </form>
    </body>

</html>
```

Observe o uso do Thymeleaf nesta página em vários pontos: no namespace da tag `<html>` para podermos usar os elementos deste engine de páginas dinâmicas.

A tag `<form>` usa um destes elementos, o `th:action` que irá produzir o atributo `action` do formulário HTML puro.

O “`@{...}`” é uma expressão do Thymeleaf que referencia URLs, neste caso, a rota no controlador que iremos implementar logo abaixo para receber os dados do formulário e salvá-los no repositório. Ainda na tag `<form>`, usamos o `th:object`, desta vez com uma expressão que referencia um objeto Java, pois começa por “`$`”.

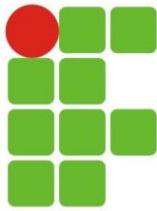
- 4) Dentro do pacote `br.edu.ifpb.pweb2.bitbank.controller` crie uma classe controladora chamada **CorrentistaController** com o seguinte código:

```
@Controller
@RequestMapping("/correntistas")
public class CorrentistaController {

    @RequestMapping("/form")
    public String getForm(Correntista correntista, Model model) {
        model.addAttribute("correntista", correntista);
        return "correntistas/form";
    }
}
```

[sobre o código]¹

¹ Você substituirá as anotações `@RequestMapping` por `@GetMapping` em prática futuras, por enquanto faça como se pede.



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

Note que tanto a classe quanto o método getForm() possuem a anotação @RequestMapping. O uso dela na classe faz com que todas as rotas para métodos desta classe tenham que ter o path “correntistas”. Isto melhora a organização do código e padroniza URLs correlacionadas ao mesmo tema, como veremos durante os testes.

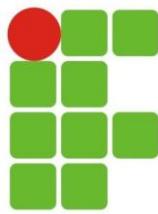
- 5) Teste a aplicação no navegador com <http://localhost:8080/bitbank/correntistas/form>. Preencha o formulário e salve os dados. Um erro deve surgir na tela do navegador, pois não implementamos o método controlador “/correntistas/save” usado no *action* do formulário.
- 6) Edite a classe CorrentistaController de modo a adicionar uma injeção de dependência para o repositório e um método:

```
@Controller  
@RequestMapping("/correntistas")  
public class CorrentistaController {  
  
    @Autowired  
    private CorrentistaRepository correntistaRepository;
```

```
@RequestMapping("/save")  
public String save(Correntista correntista, Model model) {  
    correntistaRepository.save(correntista);  
    model.addAttribute("correntistas", correntistaRepository.findAll());  
    return "correntistas/list";  
}
```

- 7) Adicione na pasta src/main/java/resources/templates/correntistas² a página **list.html** abaixo que lista os correntistas e é retornada pelo controlador acima.

² Doravante, referenciaremos a pasta src/main/java/resources/templates/ apenas como templates/.



```
<html xmlns:th="www.thymeleaf.org">

    <head>
        <meta charset="UTF-8">
        <title>Cadastro de Correntistas</title>
    </head>

    <body>
        <h1>Cadastro de Correntista</h1>

        <div th:text="${mensagem}" style="color: red"></div>

        <table border="1">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Nome</th>
                    <th>Email</th>
                    <th>Admin</th>
                    <th>Senha</th>
                </tr>
            </thead>
            <tbody>
                <tr th:each="correntista : ${correntistas}">
                    <td>[${correntista.id}]</td>
                    <td th:text="${correntista.nome}"></td>
                    <td th:text="${correntista.email}"></td>
                    <td th:text="${correntista.admin}"></td>
                    <td th:text="${#strings.substring(correntista.senha,0,3)} + '...'"></td>
                </tr>
            </tbody>
        </table>
    </body>

</html>
```

- 8) Digite <http://localhost:8080/bitbank/correntistas/form>, preencha o formulário e verifique se o correntista é listado na nova página que lista os correntistas existentes atualmente no repositório.
- 9) Sequência de telas para o cadastro:

Cadastro de Correntista

Nome:

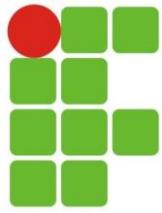
Senha:

Email:

Admin?

Cadastro de Correntista

ID	Nome	Email	Admin	Senha
2	Charles Darwin	darwin@geolsoc.org.uk	false	am...



Instituto Federal da Paraíba

Unidade Acadêmica de Informática

Curso Superior de Tecnologia em Sistemas para Internet

Disciplina: Programação para a Web II

Professor: Frederico C. G. Pereira

Projeto: Bitbank em Spring

10) Como desafio, sugerimos a você melhorar o controlador que cadastra um correntista para realizar algumas verificações:

- a) Nome do correntista é obrigatório
- b) Nome do correntista deve ter tamanho máximo de 50 caracteres, se for maior não deve aceitar e retornar para o formulário.
- c) Senha e email também são obrigatórios
- d) Caso alguma destas regras não seja respeitada, o sistema deve devolver o formulário para o usuário com uma mensagem indicando o erro (notem que a página do formulário já possui um campo \${mensagem} preparado para isso).

11) Faça um cadastro e listagem para contas seguindo os passos feitos para o cadastro/listagem de correntistas.

Esta prática pode ser baixada de um repositório git com o seguinte comando (digite num terminal):

```
git clone https://gitlab.com/fredguedespereira/bitbank-pratica01-mvc1.git
```