Universidade Federal do ABC (UFABC)
Campus Santo André
Av. dos Estados, 5001 – Bairro Bangu - CEP 09210-170
Santo André /SP - Brasil

Programação para Web
Q2/2018

Atividade 04: Spring

1. Instalar e configurar o Gradle

1. **Realizar** o download do arquivo de instalação do Gradle no site: https://gradle.org/install/#manually
2. **Descompactar** o conteúdo em uma pasta de conhecimento, por ex., c:/Gradle/
3. **Configurar** o Gradle como variável de ambiente:
    a. Win: Meu Computador/Propriedades/Variáveis de Ambiente
    b. Lin: export PATH=$PATH:/opt/gradle/gradle-4.8.1/bin
4. Após a configuração, **realizar** um teste na linha de comando com a seguinte instrução:
    a. gradle –v

2. Criar projetos com o Gradle

1. Depois de configurar o Gradle, na linha de comando **utilizar** as seguintes instruções para interagir com a ferramenta:
    a. **Gradle init --type java-application:** cria um projeto java application
    b. **Gradle build:** "constroi" o projeto gradle
    c. **Gradle tasks:** apresenta a lista de tarefas disponíveis **para o** gradle
    d. **Gradle eclipse:** configura o projeto para o ambiente do eclipse
    e. **Gradle run:** executa o arquivo gradle
2. Todas as instruções são definidas em um arquivo de configuração, chamado **build.gradle**, gerado após a execução da instrução **gradle init**.
3. Abaixo, segue um exemplo do arquivo de configuração:

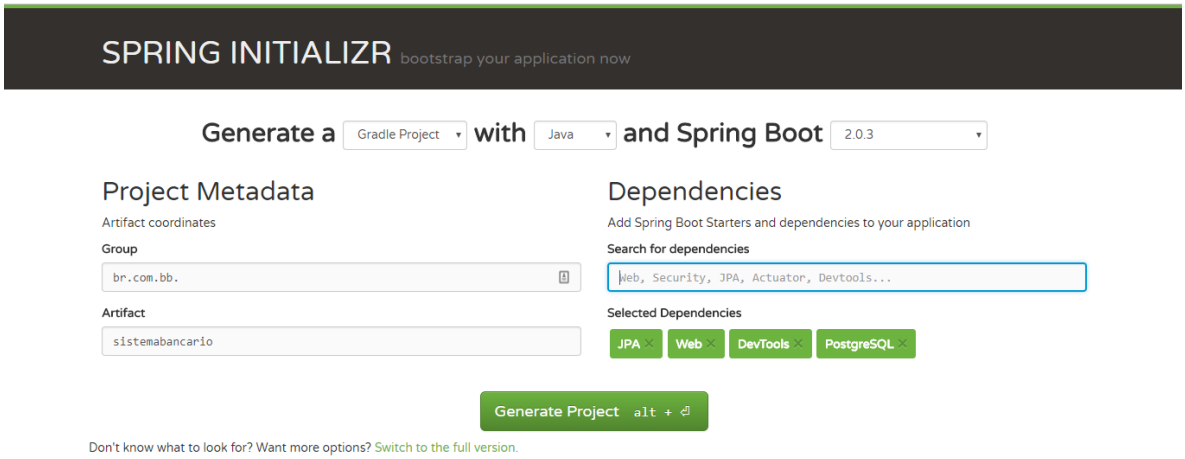
```
plugins {
    // Apply the java plugin to add support for Java

    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'

    // Apply the eclipse plugin to add support for using Eclipse IDE
    id 'eclipse'
}

// Define the main class for the application
mainClassName = 'App'

dependencies {
```

```
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'

    // https://mvnrepository.com/artifact/org.hibernate/hibernate-core
    compile group: 'org.hibernate', name: 'hibernate-core', version: '5.0.0.Final'

    // https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager
    compile group: 'org.hibernate', name: 'hibernate-entitymanager', version: '5.0.0.Final'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

1. As duas partes principais do arquivo são: 1) dependencies onde você consegue acrescentar mais pacotes para seu projeto (por ex., jdbc, hibernate, etc) e 2) plugins onde você acrescenta plug-ins necessários para o seu ambiente de desenvolvimento (por ex., eclipse)
    a. Mais dependências são encontradas no site: https://mvnrepository.com/
2. **Criar** um projeto SistemaBancarioGradle utilizando o gradle.
    a. **Criar** uma pasta na worskpace
    b. **Acessar** a pasta e **executar** a instrução para criar projeto do Gradle
3. **Incluir as dependências** do hibernate e jdbc postgres no build.gradle
4. **Importar** no eclipse (gradle eclipse).
    a. Clique com botão direito no Project Explorer, Import, Import, Existing Projects into Workspace, Browse. Procure a pasta onde o projeto foi criado.
5. **Implementar** apenas a operação de inserir uma conta corrente no banco de dados.
    a. OBS: A execução do projeto será por meio do comando gradle run ou gradle bootRun na linha de comando. Não mais pelo eclipse.
    b. O arquivo persitence.xml deve ser incluído em uma pasta de projeto chamada /src/main/resources/WEB-INF/ (criar a pasta no projeto criado).
    c. Se preferir, utilize os arquivos das aulas anteriores.

3. Spring Boot e a Geração de Projetos

1. A automatização na criação de projetos empregando SpringBoot e SpringMVC pode ser realizada por meio do seguinte site: https://start.spring.io/



1. No site:
2. Project Metadata
   a. Group: refere-se ao pacote organizador do nosso projeto; sem o nome do projeto em si (por ex., br.edu.ufabc ou br.com.bb)
   b. Artifact: nome do nosso projeto.
3. Dependencies
   a. JPA: inclui no arquivo build.gradle as dependências do hibernate
   b. Web: inclui no arquivo build.gradle as dependências do SpringMVC
   c. Postgres: Inclui no arquivo build.gradle os drivers do Postgres

4. Criar um projeto Spring MVC

1. **Criar** um novo projeto SistemaBancarioMVC utilizando o sistema de automatização para Spring Boot, conforme descrito acima.
2. **Executar** as instruções no Gradle para importar as dependências e, em seguida, **importar** o projeto no eclipse.
   a. Caso você tenha incluído o Postgres como dependência do projeto, é necessário configurar o acesso ao banco de dados. Para isso, abra o arquivo application.properties dentro da pasta /src/main/resources e insira as configurações:

```
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/<nome_banco>
spring.datasource.username=<usuario>
spring.datasource.password=<senha>
```

Atente-se: para alterar o nome do banco de dados, usuário e senha para acesso ao banco.

   b. Para inicializar o projeto, execute a instrução gradle bootRun na linha de comando. Se houver sucesso no processo a seguinte mensagem deve ser exibida na linha de comando: “Started <nome do projeto> in XXX seconds”.
   c. Em seguida, digitar no navegador http://localhost:8080/ e uma página em branco deve aparecer, ou uma página com a mensagem “*This application has no*

explicit mapping for /error, so you are seeing this as a fallback.". Verifique na linha de comando se não houve nenhuma mensagem de erro.

d. Para parar o projeto, digite Ctrl + C na linha de comando e, em seguida, Y e Enter.

3. **Criar** os pacotes do modelo MVC, ou seja, br.com.bb.sistemabancario.**model.entity**, br.com.bb.sistemabancario.**model.dao**, br.com.bb.sistemabancario.**controller**, br.com.bb.sistemabancario.**service**.
4. **Implementar** a operação de listar conta corrente.
   a. Você deve usar, principalmente, as seguintes anotações do Spring MVC para incluir as classes no escopo da aplicação.
      a.i. @Controller
      a.ii. @Repository
      a.iii. @Service
      a.iv. @RequestMapping(value = "<ref_url>")
   b. Além disso, para evitar o acoplamento por instanciação, utilize a Inversão de Controle e Injeção de Dependências com a anotação @Autowired.
   c. Por fim, o DAO tem que ser implementado como uma interface e herdar as características da classe JpaRepository<_classe_, _tipo atributo id_>, onde classe é a classe em si e o tipo pode ser Long, Integer, etc. Exemplo:

```
package br.com.bb.SistemaBancarioMVC.model.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import br.com.bb.SistemaBancarioMVC.model.entity.ContaCorrente;

@Repository
public interface ContaCorrenteDao extends JpaRepository<ContaCorrente,
Long> {

}
```

5. Desafios

- **Implementar** a interface para a operação de listar conta corrente:
  - Dica 1: utilizar a dependência: compile('org.springframework.boot:spring-boot-starter-thymeleaf')
  - Dica 2: as interfaces/páginas HTML devem ficar dentro da pasta /resources/templates/
  - Dica 3: O retorno do controle deve ser uma instância da classe ModelAndView
- Implementar a interface para a operação de cadastrar nova conta corrente:
  - Dica 1: o formulário não precisa ser em outra página. Pode utilizar a instrução de código abaixo.

```
<form action="save" method="post">
<div class="form-group">
<label for="nome">Numero</label> <input type="text"
class="form-control" id="numero" name="numero" placeholder="numero" />
</div>
<div class="form-group">
<label for="email">Descricao</label> <input type="text"
```

```
class="form-control" id="descricao" name="descricao"
placeholder="descricao" />
</div>
<div class="form-group">
<label for="email">Ativa</label>
<input type="radio" name="ativa" value="true"> Ativa<br>
  <input type="radio" name="ativa" value="false"> Desativa<br>
</div>
<div class="form-group">
<label for="telefone">Variacao</label> <input type="text"
class="form-control" id="variacao" name="variacao"
placeholder="variacao" />
</div>
<button type="submit" class="btn btn-success">Salvar</button>
</form>
```

- Dica 2: o valor da “action” no formulário deve ter um valor de referência no value do controller.
- Dica 3: para receber os valores do formulário pelo método POST, utilize a anotação @RequestParam(“<nome ref. do campo no html>”), antes do tipo do parâmetro no método do controller.