Desenvolvimento Web I

Residência de TI Aplicada
à Área Jurídica - JF e TCE

Professor: Uirá Kulesza

Outubro 2018

[Aula 5: Spring Web MVC com Spring Boot]

Framework Spring

- Framework que promove a integração entre frameworks de infra-estrutura existente
- Oferece funcionalidade básica de injeção de dependências
- Oferecer suporte para a fácil integração de frameworks
  - MVC Web
  - Transação/Persistência
  - Segurança

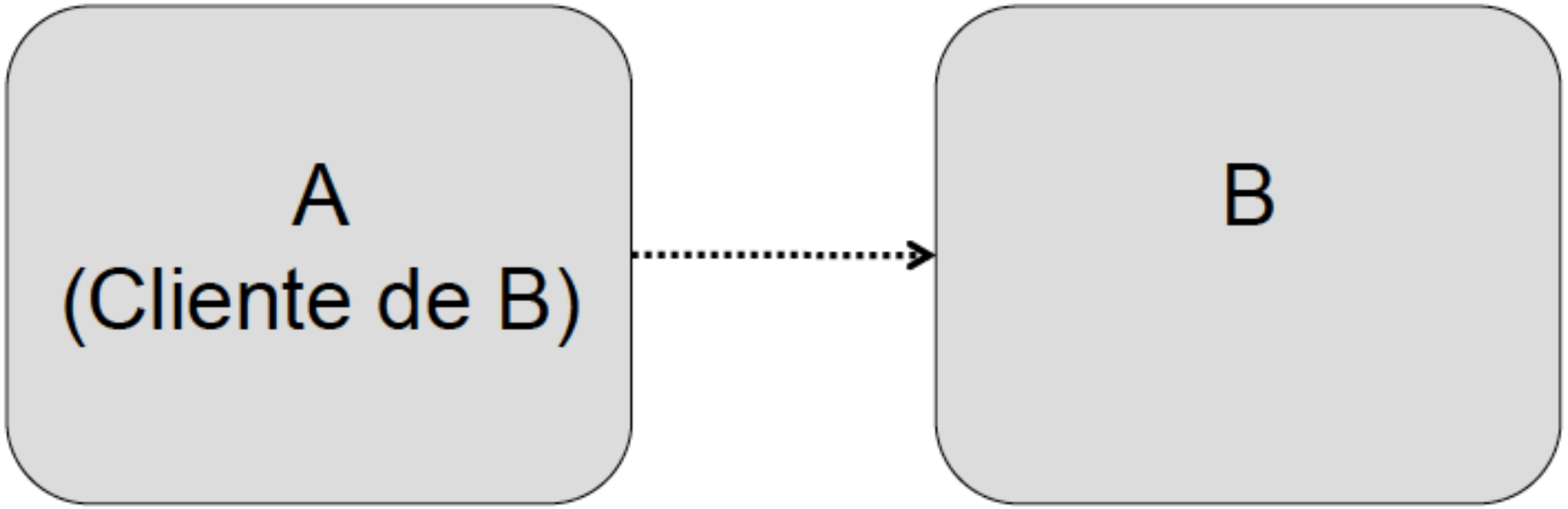
Framework Spring

- Oferece facilidades para o uso de frameworks e bibliotecas reusáveis que possam ser configuradas facilmente para ser usadas na aplicação final do usuário
- Pode ser visto como um container leve (*lightweight container*)
- Disponibiliza e permite configuração tais frameworks e bibliotecas por meio da especificação de *Spring Beans*

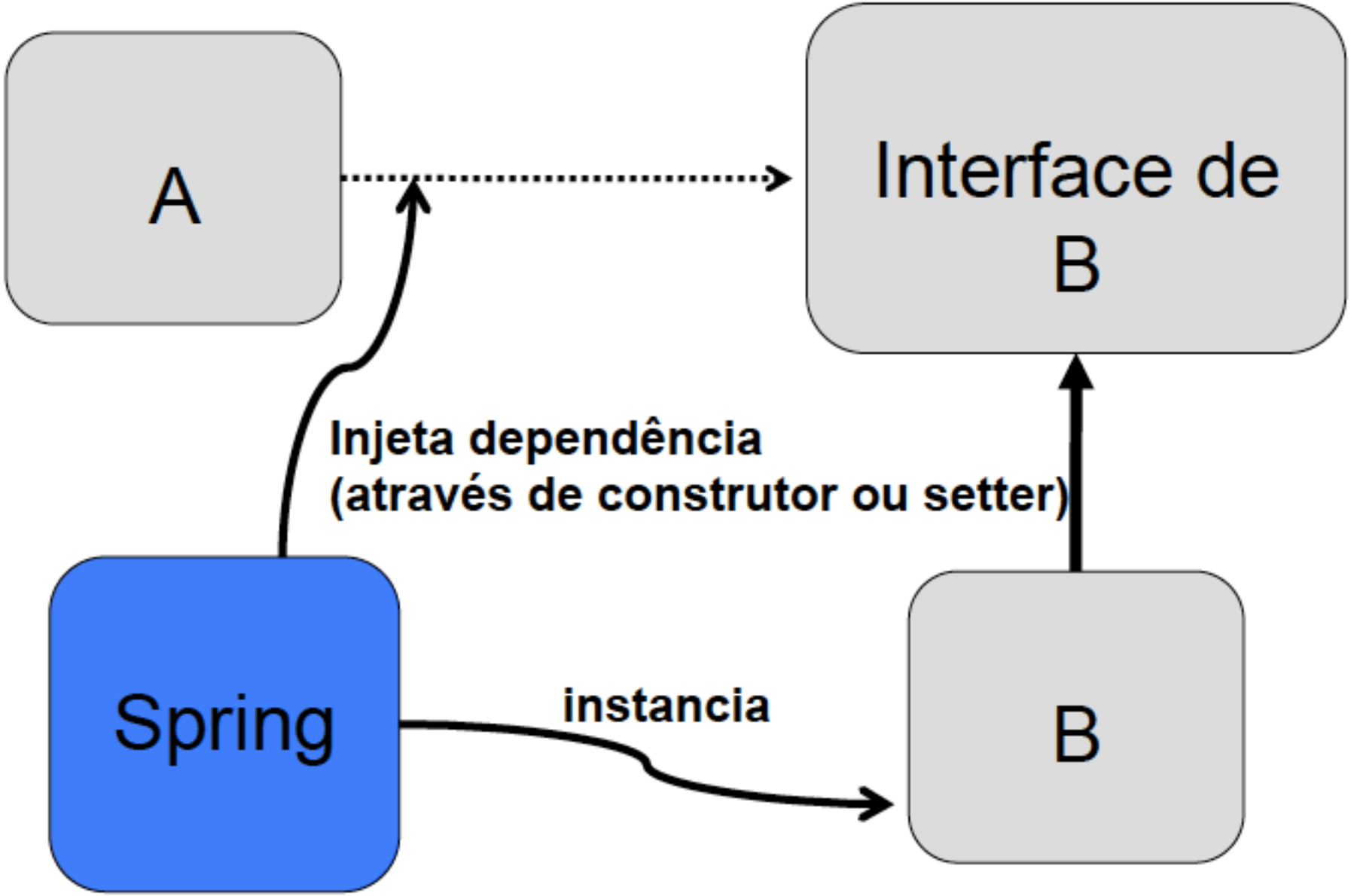
Injeção de Dependências

- Classes/Componentes de um sistema deveriam ser o mais independente possível de uma dada implementação
- Uma classe A têm uma dependência para uma classe B, se ela declara um atributo do tipo B dentro dela
- Com o uso do mecanismo de injeção de dependência, o objeto referenciado do tipo B dentro de A, deve:
  - Ser configurado pelo construtor da classe
  - Através de um método *setter()* dentro da classe A

Gerência de Dependências



Gerência de Dependências



Benefícios

- A injeção de dependência permite fazer projetos mais fáceis de reusar e manter, além de também facilitar o teste

- Exemplo:
  - Se um dado objeto X referencia um DAO, você pode injetar um objeto “mock” que simula tal DAO, de forma a fazer o teste de X

Spring MVC e Spring Boot

- Spring MVC é um framework disponível dentro do Spring para desenvolver aplicações web em Java

- Spring Boot permite definir a configuração de uma aplicação web através da definição de um simples *main*

Aplicação Spring MVC

- Aplicações organizadas em:
  - Controladores
    - Classe Java que faz o tratamento do request
  - *Templates*
    - Página HTML customizável
  - Serviços
    - Classe Java que implementa as regras de negócio
  - Repositórios
    - Classe Java que faz a persistência em um meio de armazenamento
  - Entidades
    - Classe Java que contém os dados do sistema

Classes Entidade

- Representam classes Java que terão informações persistidas no banco de dados

- Anotações JPA (*Java Persistence API*)
  - Indicam que classes serão persistidas como tabelas no banco de dados
  - E para atributo da classe como será feito tal mapeamento

Entidade *Student*

```java
@Entity
@Table(name = "students")
public class Student implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(name = "name")
    private String name;

    @Column(name = "email")
    private String email;

    @Column(name = "registration")
    private String registration;

    @ManyToOne
    @JoinColumn(name="module_id")
    private Module module;

    public void setName(String name) {
        this.name = name;
    }
```

Classes Repositórios

- Representam classes Java que funcionam como DAOs para acessar o banco de dados

- O Spring traz uma série de interfaces e classes que permite definir código padrão de CRUDs de tais repositórios escrevendo poucas linhas de código
  - Apenas consultas mais sofisticadas precisam ser codificadas

Interface *JPARepository*

- Herda indiretamente de CrudRepository

- Oferece vários métodos para persistir os dados, tais como:
  - save(): salva os dados de uma entidade
  - findById(): retorna pelo id alguma entidade
  - findAll(): retorna as diversas instâncias de uma entidade
  - deleteById(): remove uma dada entidade
  - existsById(): verifica se uma dada entidade existe

Classe *StudentRepository*

```java
package com.example.repository;
import org.springframework.stereotype.Repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.model.Student;

@Repository
public interface StudentRepository extends JpaRepository<Student, Integer> {

}
```

Definindo métodos específicos *JPARepository*

- É possível definir métodos específicos de buscas usando anotações com consultas
  - Também método findAllByX(): onde X pode ser trocado por diferentes colunas nas tabelas

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.jpa.repository.JpaRepository;

import com.vogella.example.entity.IssueReport;

public interface IssueRepository extends JpaRepository<IssueReport, Long> {
    @Query(value = "SELECT i FROM IssueReport i WHERE markedAsPrivate = false")
    List<IssueReport> findAllButPrivate();

    List<IssueReport> findAllByEmail(String email);
}
```

Classes Serviços

- Representam classes Java que delimitam as regras de negócio, sendo usadas para:
  - Implementar as regras de negócio do sistema
  - Delimitar transações com o banco de dados

- Tais classes mantém uma referência para classes repositórios da camada de dados e usam seus serviços para recuperar ou armazenar informações de classes Entidade com o banco de dados

Classe *StudentService*

```java
@Service
@Transactional(readOnly = true)
public class StudentService {

    @Autowired
    private StudentRepository studentRepository;

    public List<Student> findAll() {
        return studentRepository.findAll();
    }

    public Optional<Student> findOne(Integer id) {
        return studentRepository.findById(id);
    }

    @Transactional(readOnly = false)
    public Student save(Student entity) {
        return studentRepository.save(entity);
    }

    @Transactional(readOnly = false)
    public void delete(Student entity) {
        studentRepository.delete(entity);
    }

}
```

Classes *Controller*

- Representam classes Java que processam requisições web solicitadas por formulários implementados pelos *template* ThymeLeaf

- Comportamento é tipicamente:
  - Método específico é chamado para tratar requisições
  - Faz chamadas a classes de serviço para armazenar novos objetos Entidade ou recuperar algum existente
  - Repassa o controle para um template junto com objetos Entidade eventualmente recuperados

Classe *StudentController*

```java
@Controller
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @Autowired
    private ModuleService moduleService; //module service

    // Primeira tela da pagina de Students
    @GetMapping
    public String index(Model model) {
        List<Student> all = studentService.findAll();
        model.addAttribute("listStudent", all);
        model.addAttribute("");
        return "student/index";
    }

    // Tela de Show Student
    @GetMapping("/{id}")
    public String show(Model model, @PathVariable("id") Integer id) {
        if (id != null) {
            Student student = studentService.findOne(id).get();
            model.addAttribute("student", student);
        }
        return "student/show";
    }
```

Classe *StudentController*

```
// Tela de Show Student
@GetMapping("/{id}")
public String show(Model model, @PathVariable("id") Integer id) {
    if (id != null) {
        Student student = studentService.findOne(id).get();
        model.addAttribute("student", student);
    }
    return "student/show";
}

// Tela com Formulario de New Student
@GetMapping(value = "/new")
public String create(Model model, @ModelAttribute Student entityStudent,
                     @ModelAttribute Module entityModule) {
    // model.addAttribute("student", entityStudent);
    List<Module> all = moduleService.findAll();
    model.addAttribute("modules", all);

    return "student/form";
}
```

```java
// Processamento do formulario New Student (ou Alter Student)
@PostMapping
public String create(@Valid @ModelAttribute Student entityStudent,
                     @Valid @ModelAttribute Module entityModule,
                     BindingResult result, RedirectAttributes redirectAttributes) {
    Student student = null;

    try {
        student = studentService.save(entityStudent);
        redirectAttributes.addFlashAttribute("success", MSG_SUCESS_INSERT);
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", MSG_ERROR);
        e.printStackTrace();
    }
    return "redirect:/students/" + student.getId();
}

@GetMapping("/{id}/edit")
public String update(Model model, @PathVariable("id") Integer id) {

    try {
        if (id != null) {
            List<Module> all = moduleService.findAll();
            model.addAttribute("modules", all);

            Student entity = studentService.findOne(id).get();
            model.addAttribute("student", entity);
        }
    } catch (Exception e) {
        throw new ServiceException(e.getMessage());
    }
    return "student/form";
}
```

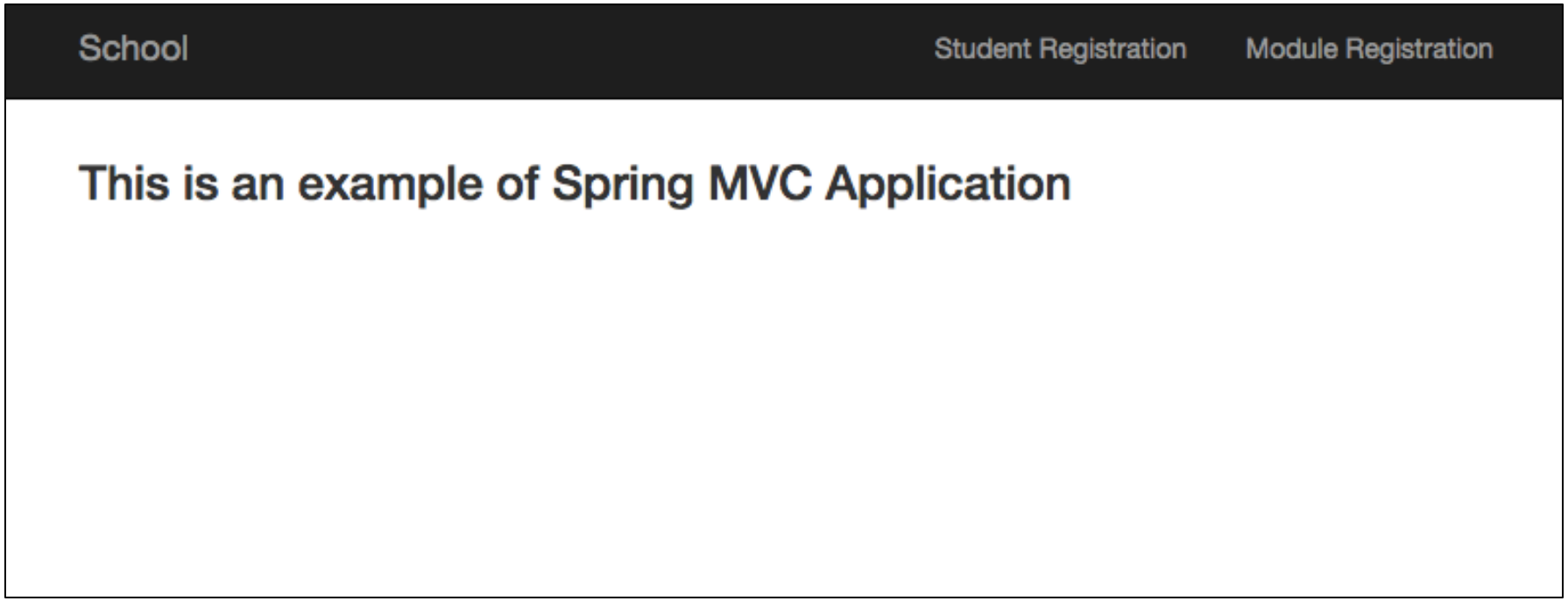
```java
@PutMapping
public String update(@Valid @ModelAttribute Student entity, BindingResult result,
                     RedirectAttributes redirectAttributes) {
    Student student = null;
    try {
        student = studentService.save(entity);
        redirectAttributes.addFlashAttribute("success", MSG_SUCESS_UPDATE);
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", MSG_ERROR);
        e.printStackTrace();
    }
    return "redirect:/students/" + student.getId();
}

@RequestMapping("/{id}/delete")
public String delete(@PathVariable("id") Integer id, RedirectAttributes redirectAttributes) {
    try {
        if (id != null) {
            Student entity = studentService.findOne(id).get();
            studentService.delete(entity);
            redirectAttributes.addFlashAttribute("success", MSG_SUCESS_DELETE);
        }
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", MSG_ERROR);
        throw new ServiceException(e.getMessage());
    }
    return "redirect:/students/";
}
```

Templates *ThymeLeaf*

- Tecnologia para processamento de *templates* e criação de páginas web em aplicações Spring MVC

- São páginas HTML que importam o seguinte namespace
  - <html lang="en" xmlns:th="http://www.thymeleaf.org">

Página de entrada da Aplicação



Página de Entrada de *Students*

Página de entrada da Aplicação

```
<body>
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">School</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav navbar-right">
                <li><a href="/students">Student Registration</a></li>
                <li><a href="/modules">Module Registration </a></li>
            </ul>
        </div>
    </div>
</div>
```

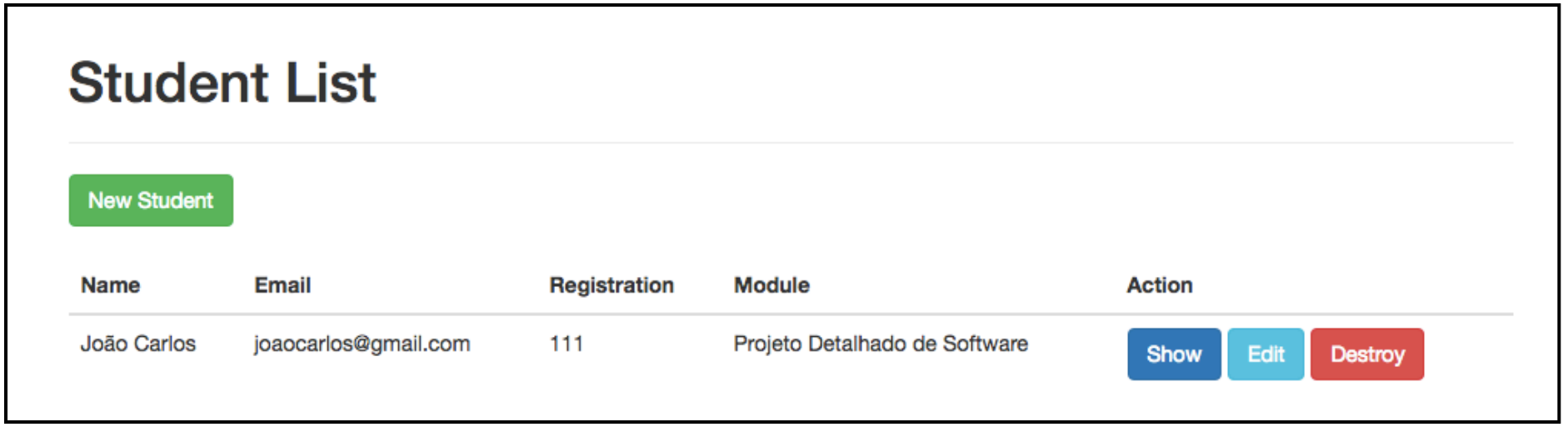
Método *index()* trata página de entrada

```java
@Controller
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @Autowired
    private ModuleService moduleService; //module service

    // Primeira tela da pagina de Students
    @GetMapping
    public String index(Model model) {
        List<Student> all = studentService.findAll();
        model.addAttribute("listStudent", all);
        model.addAttribute("");
        return "student/index";
    }

    // Tela de Show Student
    @GetMapping("/{id}")
    public String show(Model model, @PathVariable("id") Integer id) {
        if (id != null) {
            Student student = studentService.findOne(id).get();
            model.addAttribute("student", student);
        }
        return "student/show";
    }
```

Página de entrada *student/index*

```
<div layout:fragment="content">
    <h1>Student List</h1>
    <hr />
    <a th:href="@{/students/new}" class="btn btn-success">New Student</a><br/><br/>

    <table class="table">
        <thead>
            <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Registration</th>
                <th>Module</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="student : ${listStudent}">
                <td th:text="${student.name}"></td>
                <td th:text="${student.email}"></td>
                <td th:text="${student.registration}"></td>
                <td th:text="${student.module}"></td>
                <td>
                        <a th:href="@{/students/{id}(id = ${student.id})}" class="btn btn-primary" >Show</a>
                        <a th:href="@{/students/{id}/edit(id = ${student.id})}" class="btn btn-info">Edit</a>
                        <a th:href="${'students/'+ student.id +'/delete'}" class="btn btn-danger">Destroy</a>
                </td>
            </tr>
        </tbody>
    </table>
  </div>
```

Página de entrada *student/index*



Página *Show Students*

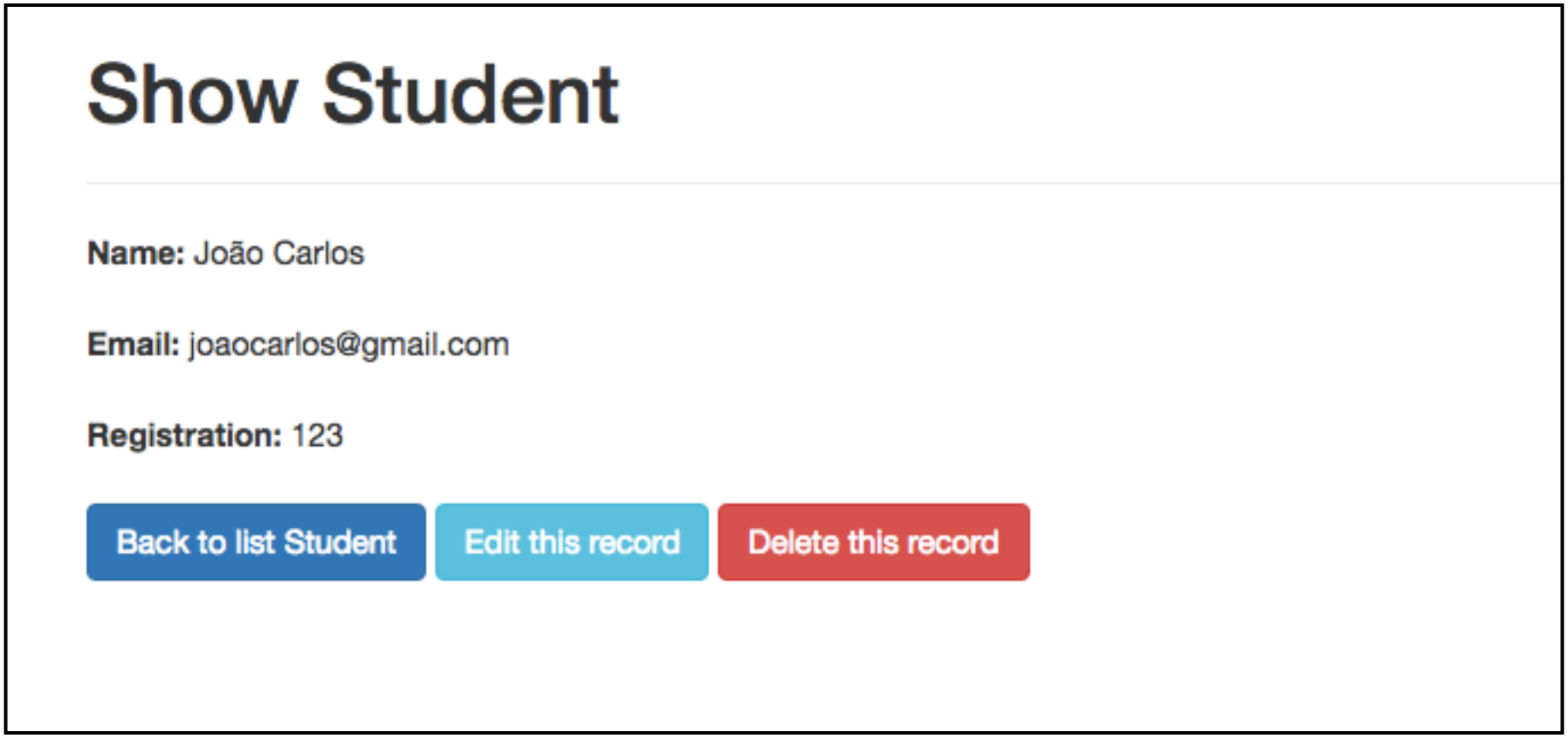
Método *show()* trata *ShowStudent*

```
// Tela de Show Student
@GetMapping("/{id}")
public String show(Model model, @PathVariable("id") Integer id) {
    if (id != null) {
        Student student = studentService.findOne(id).get();
        model.addAttribute("student", student);
    }
    return "student/show";
}

// Tela com Formulario de New Student
@GetMapping(value = "/new")
public String create(Model model, @ModelAttribute Student entityStudent,
                     @ModelAttribute Module entityModule) {
    // model.addAttribute("student", entityStudent);
    List<Module> all = moduleService.findAll();
    model.addAttribute("modules", all);

    return "student/form";
}
```

Template *ShowStudent*

```
<div layout:fragment="content">
    <h1 th:if="${student.id == null}">New Student</h1>
    <h1 th:if="${student.id != null}">Edit Student</h1>
    <hr />
    <form th:action="@{/students}"
        th:method="@{${student.id == null} ? 'post' : 'put'}"
        th:object="${student}">
        <input type="hidden" th:field="*{id}" />

        <div class="form-group">
            <label for="name">Name</label> <input id="name" name="name"
                type="text" class="form-control" th:field="*{name}" />
        </div>
        <div class="form-group">
            <label for="email">Email</label> <input id="email" name="email"
                type="text" class="form-control" th:field="*{email}" />
        </div>
        <div class="form-group">
            <label for="registration">Registration</label> <input
                id="registration" name="registration" type="text"
                class="form-control" th:field="*{registration}" />
        </div>

         <div class="form-group" th:object="${module}">
             <select name="module.id" class="form-control">
                 <option value="">Selecione o Modulo</option>
                 <option th:each="module : ${modules}" th:value="${module.id}" th:text="${module.name}"></option>
            </select>
         </div>

        <button type="submit" class="btn btn-success">Save</button>
        <a th:href="@{/students}" class="btn btn-default">Back</a>
    </form>
</div>
```

Página de show *Student*



Página *New Students*

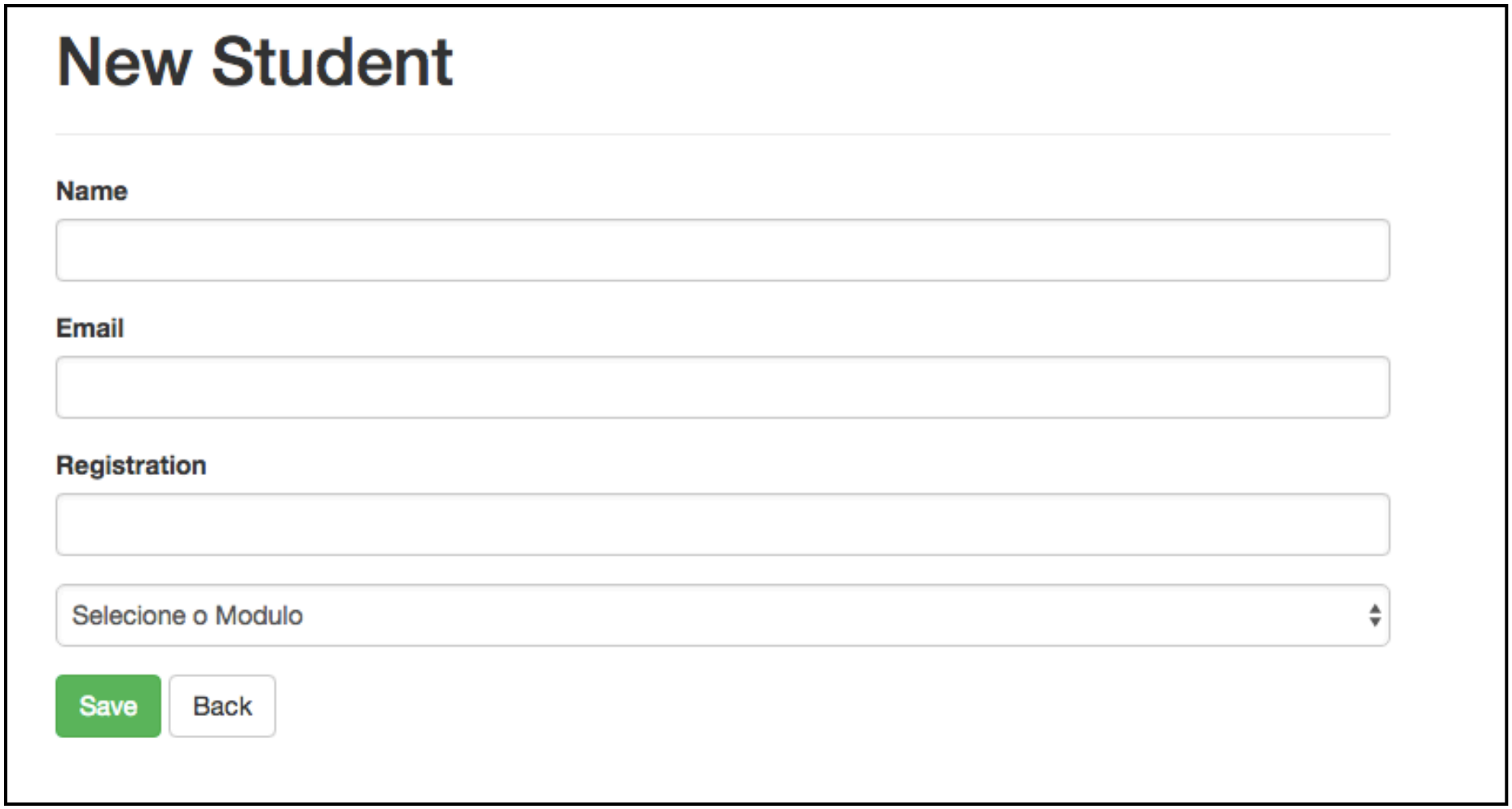
Página de entrada *student/index*

```
<div layout:fragment="content">
    <h1>Student List</h1>
    <hr />
    <a th:href="@{/students/new}" class="btn btn-success">New Student</a><br/><br/>

    <table class="table">
        <thead>
            <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Registration</th>
                <th>Module</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="student : ${listStudent}">
                <td th:text="${student.name}"></td>
                <td th:text="${student.email}"></td>
                <td th:text="${student.registration}"></td>
                <td th:text="${student.module}"></td>
                <td>
                        <a th:href="@{/students/{id}(id = ${student.id})}" class="btn btn-primary" >Show</a>
                        <a th:href="@{/students/{id}/edit(id = ${student.id})}" class="btn btn-info">Edit</a>
                        <a th:href="${'students/'+ student.id +'/delete'}" class="btn btn-danger">Destroy</a>
                </td>
            </tr>
        </tbody>
    </table>
  </div>
```

Abertura do *Form New Student*

```
// Tela de Show Student
@GetMapping("/{id}")
public String show(Model model, @PathVariable("id") Integer id) {
    if (id != null) {
        Student student = studentService.findOne(id).get();
        model.addAttribute("student", student);
    }
    return "student/show";
}

// Tela com Formulario de New Student
@GetMapping(value = "/new")
public String create(Model model, @ModelAttribute Student entityStudent,
                     @ModelAttribute Module entityModule) {
    // model.addAttribute("student", entityStudent);
    List<Module> all = moduleService.findAll();
    model.addAttribute("modules", all);

    return "student/form";
}
```

Página de new *Student*



Página de new *Student*

```
<body>
    <div layout:fragment="content">
        <h1 th:if="${student.id == null}">New Student</h1>
        <h1 th:if="${student.id != null}">Edit Student</h1>
        <hr />
        <form th:action="@{/students}"
            th:method="@{${student.id == null} ? 'post' : 'put'}"
            th:object="${student}">
            <input type="hidden" th:field="*{id}" />

            <div class="form-group">
                <label for="name">Name</label> <input id="name" name="name"
                    type="text" class="form-control" th:field="*{name}" />
            </div>
            <div class="form-group">
                <label for="email">Email</label> <input id="email" name="email"
                    type="text" class="form-control" th:field="*{email}" />
            </div>
            <div class="form-group">
                <label for="registration">Registration</label> <input
                    id="registration" name="registration" type="text"
                    class="form-control" th:field="*{registration}" />
            </div>

             <div class="form-group" th:object="${module}">
                 <select name="module.id" class="form-control">
                     <option value="">Selecione o Modulo</option>
                     <option th:each="module : ${modules}" th:value="${module.id}" th:text="${module.name}"></option>
                </select>
             </div>

            <button type="submit" class="btn btn-success">Save</button>
            <a th:href="@{/students}" class="btn btn-default">Back</a>
        </form>
    </div>
</body>
```

```java
@PutMapping
public String update(@Valid @ModelAttribute Student entity, BindingResult result,
                     RedirectAttributes redirectAttributes) {
    Student student = null;
    try {
        student = studentService.save(entity);
        redirectAttributes.addFlashAttribute("success", MSG_SUCESS_UPDATE);
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", MSG_ERROR);
        e.printStackTrace();
    }
    return "redirect:/students/" + student.getId();
}

@RequestMapping("/{id}/delete")
public String delete(@PathVariable("id") Integer id, RedirectAttributes redirectAttributes) {
    try {
        if (id != null) {
            Student entity = studentService.findOne(id).get();
            studentService.delete(entity);
            redirectAttributes.addFlashAttribute("success", MSG_SUCESS_DELETE);
        }
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", MSG_ERROR);
        throw new ServiceException(e.getMessage());
    }
    return "redirect:/students/";
}
```

Configuração da aplicação com o *SpringBoot*

DemoAplication.java

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication(scanBasePackages="com.example")
@EntityScan("com.example.model")
@EnableJpaRepositories("com.example.repository")
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Referências

- Tutorial Spring MVC
  - http://www.vogella.com/tutorials/SpringBoot/article.html#example-spring-boot-application

- Mastering Spring framework 5, Part 1: Spring MVC
  - https://www.javaworld.com/article/2078034/spring-framework/spring-framework-mastering-spring-mvc.html