



# Curso de Spring Boot

Instrutor: Bergson Barros

# Apresentação do instrutor



- 41 anos, casado, pai do Davi e da Laura
- Bacharel em Ciência da Computação (UFAL)
- Pós-Graduado em Segurança de Redes e Criptografia (UFF)
- Analista de Sistemas do Serpro
- Trabalha profissionalmente com Java há 19 anos
- Certificações em Python (PCEP), Azure (DP-900), LGPD (LGPDP) e Scrum Foundations (SFPC)
- Trabalha com Spring Boot desde 2019

# Motivação

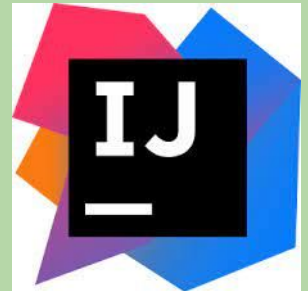
Relaxe, pause para afiar o seu machado!



**“Se eu tivesse apenas uma hora para cortar uma árvore, eu usaria os primeiros quarenta e cinco minutos afiando meu machado.”**

**“Tempo de treinamento não é tempo perdido.”**

# IDEs que utilizaremos



# Atalhos do Eclipse

<Ctrl> + <Shift> + **F** - Formata o código

<Ctrl> + <Shift> + **O** - Organiza os imports

<Ctrl> + <Shift> + **D** - Apaga a linha

<Ctrl> + <Shift> + **G** - Procura por referência da classe/método

<Ctrl> + <Shift> + **T** (Type) - Procura por classes

<Ctrl> + <Shift> + **R** (Resource) - Procura por arquivos

<Ctrl> + **L** (Line) - Vai para a linha

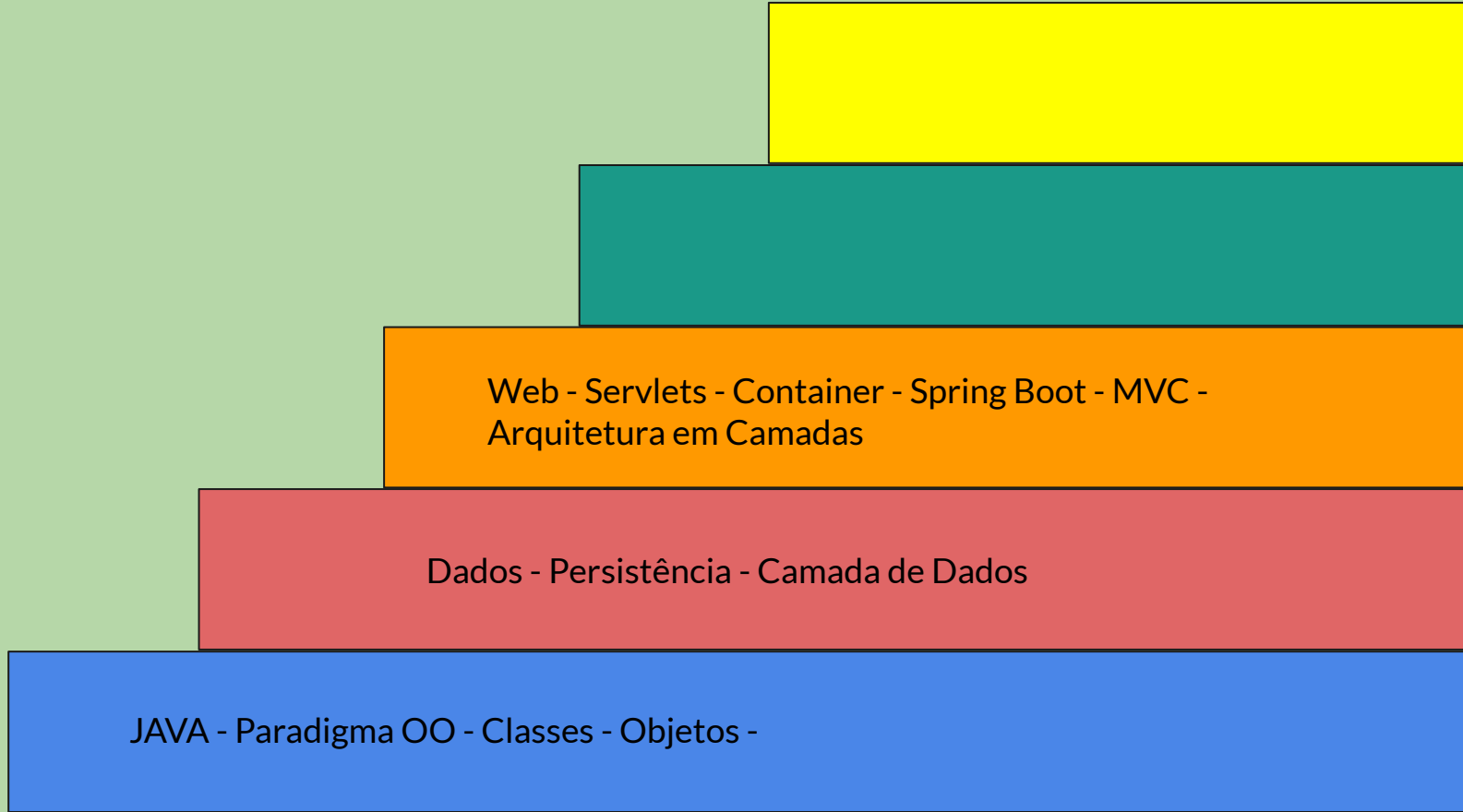
<Ctrl> + **S** (Save) - Salva o trabalho atual

<Ctrl> + **Z** (Undo) - Desfazer a última operação

<Ctrl> + **R** (Redo) - Refaz a última operação



# Avançando no Aprendizado



# Por que estudar Spring?

“Spring torna a programação em Java mais **rápida**, mais **simples** e mais **segura** para todos. Spring foca na **velocidade**, **simplicidade** e a **alta produtividade** tornou o Spring o framework Java mais popular do mundo.” (fonte: site Spring)

- Relatório dos frameworks mais utilizados no mundo:  
<https://snyk.io/jvm-ecosystem-report-2021/>



# Benefícios de usar Spring?

- Spring está em todos os lugares (big techs)
- Flexível (Spring Core e bibliotecas de terceiros)
- Produtivo (web serve embarcado)
- Rápido (iniciar, parar, execução otimizada)
- Seguro (cuidado dos desenvolvedores em gerenciar as vulnerabilidades das bibliotecas)
- Solidário (grande comunidade mundial para todas as diversidades, idades...)

Fonte: <https://spring.io/why-spring>



# Ecosystem Spring

- Spring Boot
- Spring Framework
- Spring Data
- Spring Cloud
- Spring Security
- Spring Session
- Spring Batch

Todos os projetos do ecossistema Spring estão em <https://spring.io/projects>

# Conhecendo o Spring Boot

- Documentação <https://spring.io/projects/spring-boot>
- Spring Boot facilita a criação de aplicações independentes (stand-alone), baseado em Spring, que você pode simplesmente executar



# Funcionalidades do Spring Boot

- Criação de aplicações stand-alone (independentes)
- Tomcat embarcado, Jetty ou Undertown
- Não há necessidade de arquivos WAR
- Simplificação na configuração da build através do 'starter'
- Configuração das bibliotecas Spring e de terceiros (3rd party)
- Provê ferramentas de apoio e monitoração da produção (metrics, health checks, etc)

# Criando projetos com Spring Initializr

<https://start.spring.io/>

# Apache Maven



- Apache Maven é uma ferramenta responsável pelo gerenciamento das builds do projeto, suas configurações e suas dependências
- Baseado em um arquivo **POM**
- Gerenciamento de dependências do projeto

# Prática 1

- Criar um projeto inicial no Spring Initializr (<https://start.spring.io/>)
- Verificar no pom todas as informações passadas no Spring Initializr
- Criar uma classe controladora e anotar com @Controller
- Na classe criada, criar método hello que retorna uma mensagem de boas vindas. Exemplo: **Olá aluno, seja bem-vindo!!!**

# Prática 2

- Criar uma classe controladora com seu nome e anotar com `@Controller`
- Na classe criada, criar método `hello` que retorna uma mensagem de boas vindas
- Após isto, decore a mensagem de boas vindas com tags HTML (`h1`, `h2`, `p`, `strong`, ect)

# Projeto do Curso - Scholl Control API

- Criação de uma API REST para praticar os conceitos aprendidos nas aulas voltando para o negócio de uma escola.

Projetos:

- Spring MVC
- Spring Data JPA
- Spring Validation



# Aula 02 - Alvos para hoje

- Spring Core
- O que é **API**?
- Arquivos **JSON**
- Arquitetura **REST**
- Criar nosso primeiro EndPoint
- Iniciar JPA (se possível....)



# Aula 02 - Material de Apoio

1) Notepad oficial do curso:

<https://notepad.link/fectura-spring-boot-20220806>

2) Repósitório do curso no Git Hub:

<https://github.com/Bergolito/curso-springboot-fectura>

3) Classes do Modelo - Api Escola:

<https://github.com/Bergolito/curso-springboot-fectura/blob/main/Aula02%20-%202022-08-06/escola-model.zip>

# Spring Core

- Atualmente na versão **5.3.22** (Dependency Hierarchy no POM)
- **Inversão de Controle** (IoC) é um padrão de projeto (Abstrato) no qual, os objetos apenas declaram suas dependências, sem criá-las, delegando essa tarefa da criação de dependências a um Container IoC (Core Container).
- **Injeção de Dependência** é a implementação (Concreta) utilizada pelo Spring Framework para a aplicação da Inversão de Controle (quando necessário).

# Beans e Estereótipos

- **Bean** é um objeto que é instanciado, montado e gerenciado por um container do Spring através da **Inversão de Controle** (IoC) e da **Injeção de Dependência**.
- **Estereótipos**: Categorias de Beans específicos do Spring Framework: @Component, @Service, @Controller, @Repository, @Autowired

# API

- **API**: Acrônimo para **Application Programming Interface**, ou Interface de Programação de/para Aplicações
- Define uma forma de como dois (ou mais) componentes de software possam se comunicar, através de um conjunto de definições e protocolos.

# API

- Exemplos:

1) API da Via CEP <https://viacep.com.br/>

2) Consulta CEP dos Correios:

<https://buscacepinter.correios.com.br/app/endereco/index.php>

3) API de Serviços de Dados do IBGE:

<https://servicodados.ibge.gov.br/api/docs>

<https://www.ibge.gov.br/censo2010/apps/nomes/#/search>

# Arquivos JSON

- **JSON** - Acrônimo para **J**ava **S**cript **O**bject **N**otation
- Surgiu como uma alternativa para **XML**
- Compacto, bem mais leve e mais prático que XML
- Usado na troca simples e rápida de dados entre sistemas

# REST - REpresentative State Transfer

- É um modelo de arquitetura que fornece diretrizes para que os **sistemas distribuídos** se comuniquem diretamente usando os princípios e protocolos existentes da Web sem a necessidade de **SOAP** ou outro protocolo sofisticado.
- Não é uma linguagem nem tecnologia nem framework



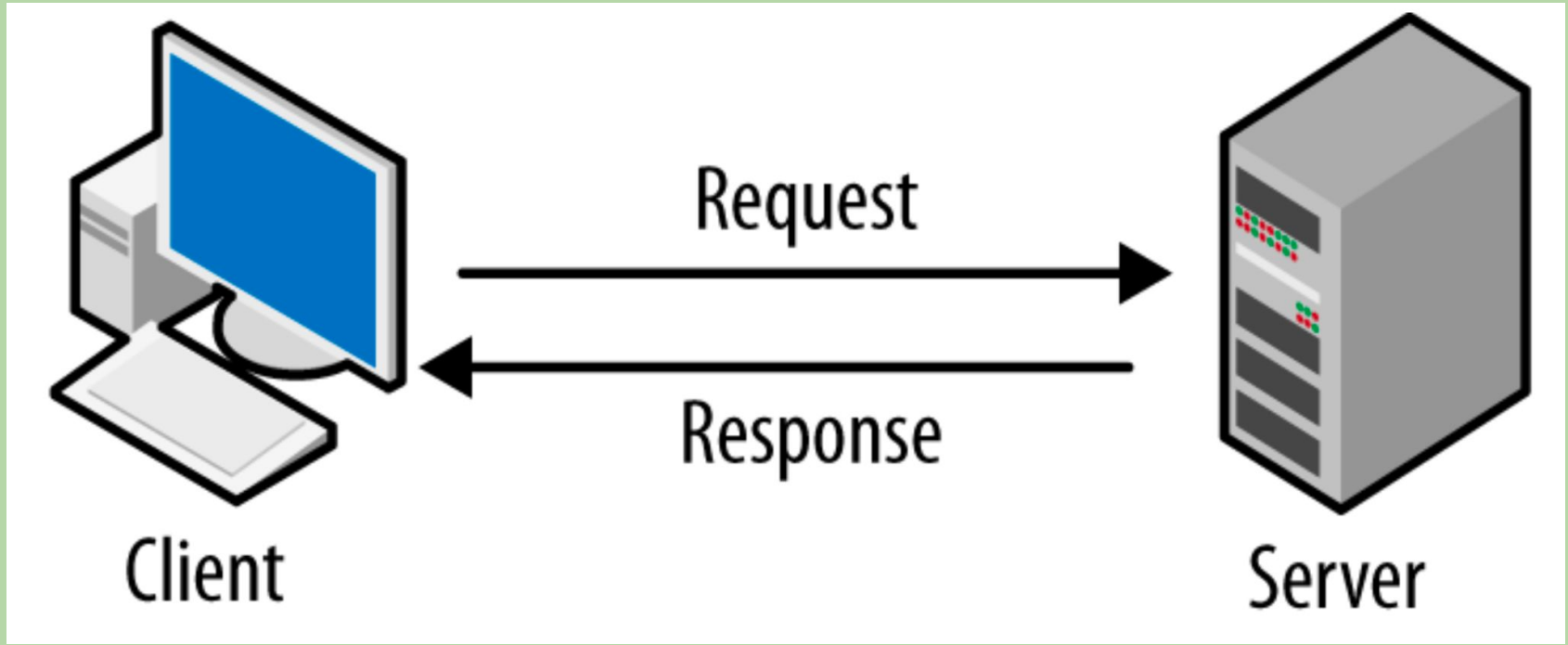
# Responsabilidades no REST

- Cliente
- Servidor
- **STATELESSNESS** - Sem estado (não guarda estado)

“Servidor não precisa saber o estado do cliente e vice-versa”

- Request / Response

# Responsabilidades no REST



Fonte: [https://darvishdarab.github.io/cs421\\_f20/docs/readings/restful/api/](https://darvishdarab.github.io/cs421_f20/docs/readings/restful/api/)

# REST - Requisições e comunicações

- O REST precisa que um cliente faça uma requisição (request) para o servidor para enviar ou modificar dados (response).
- Uma requisição consiste em:
  - ❑ Um método HTTP
  - ❑ Um cabeçalho (header)
  - ❑ Um caminho ou rota (path)
  - ❑ Uma informação no corpo da requisição (opcional)

# REST - Métodos HTTP

- Em aplicações REST, os métodos mais utilizados são:
  - ❑ método GET
  - ❑ método POST
  - ❑ método PUT
  - ❑ método DELETE

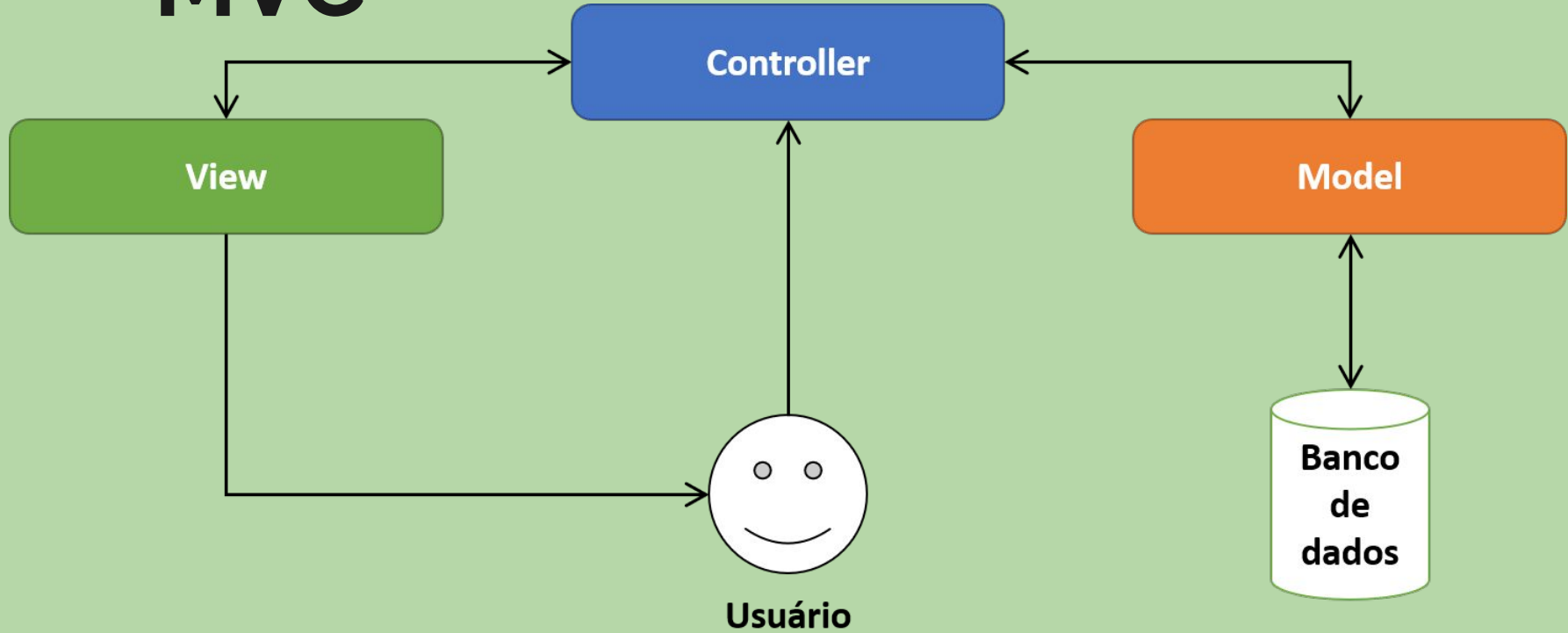
# REST - Códigos de Resposta

- Para cada resposta de requisição, existe um código de status associado:
  - ❑ 200 (**OK**), requisição atendida com sucesso;
  - ❑ 201 (**CREATED**), objeto ou recurso criado com sucesso;
  - ❑ 204 (**NO CONTENT**), objeto ou recurso deletado com sucesso;
  - ❑ 400 (**BAD REQUEST**), ocorreu algum erro na requisição (podem existir inúmeras causas);
  - ❑ 404 (**NOT FOUND**), rota ou coleção não encontrada;
  - ❑ 500 (**INTERNAL SERVER ERROR**), ocorreu algum erro no servidor

# MVC

- Acrônimo para **Model-View-Controller**
- É um padrão de projeto de software focado no **reúso de código** e na separação de conceitos em **três camadas interconectadas**, onde a apresentação dos dados e interação dos usuários são separados dos métodos que interagem com o banco de dados.

# MVC



Fonte: <https://www.treinaweb.com.br/blog/o-que-e-mvc>

# Criando nosso primeiro Controlador

- No pacote ***br.com.fuctura.escola.model***, criar a classe Aluno, com seus atributos, métodos getters e setters, equals() e hashCode()

## **Atributos do Aluno:**

```
private Long id;  
private String cpf;  
private String nome;  
private String email;  
private String fone;  
// Tipo pode ser CONVENCIONAL ou MONITOR  
private String tipo = TipoAluno.CONVENCIONAL.toString();
```



# Criando nosso primeiro Controlador

- No pacote ***br.com.fuctura.escola.controller***, criar a classe PrimeiroController
- Anotar com @RestController
- Anotar com @RequestMapping("/primeiro")
- criar um método que irá retornar uma lista de alunos

```
public List<Aluno> listarAlunos()  
  
}
```

- No método, anotar com @GetMapping("/listar")

# Aula 03 - Alvos para hoje



- Padrão **DTO**
- Melhorando nosso primeiro EndPoint
- **JPA**
- Criar o Controlador de Aluno
- Criar o Repositório de Aluno
- Instalação do cliente Postman
- Criar os endpoints para os métodos GET/POST/PUT e DELETE

# Padrão DTO

- **DTO** - Acrônimo para **D**ata **T**ransfer **O**bjeto
- Evitar usar a classe de Entidade de banco (Aluno) como retorno de serviço
- Classe DTO deve ser leve e com atributos simples
- Por padrão deve conter apenas dados simples: String, Long, Integer, Float, etc

# Melhorando nosso primeiro Controlador

- Criar o pacote ***br.com.fuctura.escola.dto***
- Criar a classe **AlunoDTO**
- Atributos do dto: cpf, nome, email
- Criar um método que irá retornar uma lista de AlunosDTO
- No método, anotar com @GetMapping("/listar2")
- Adicionar a dependência no pom referente ao Bean Validation

# Spring Data JPA

- No arquivo **application.properties** do projeto, adicionar as configurações específicas do banco H2
- Na classe Aluno, adicionar as anotações específicas do JPA para mapear a classe aluno do modelo como uma entidade/tabela do banco de dados

# Arquivo Application.properties

```
# =====  
# CONFIGURACOES EM GERAL  
# =====  
  
# datasource  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.url=jdbc:h2:mem:escola-controle-api  
spring.datasource.username=sa  
spring.datasource.password=  
  
# jpa  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.jpa.hibernate.ddl-auto=update  
  
# ATENCAO  
spring.jpa.defer-datasource-initialization=true  
  
# Console do banco H2  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console
```

# Classe Aluno

- Adicionar as anotações na classe: @Entity e @Table
- Adicionar as anotações em cada atributo da classe:

**@Id @GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**@Column(nullable = false, name = "CPF")**

**private String cpf;**

# Classe Aluno

```
@Entity
@Table
public class Aluno implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, name = "CPF")
    private String cpf;

    @Column(nullable = false, name = "NOME")
    private String nome;

    @Column(nullable = true, name = "EMAIL")
    private String email;

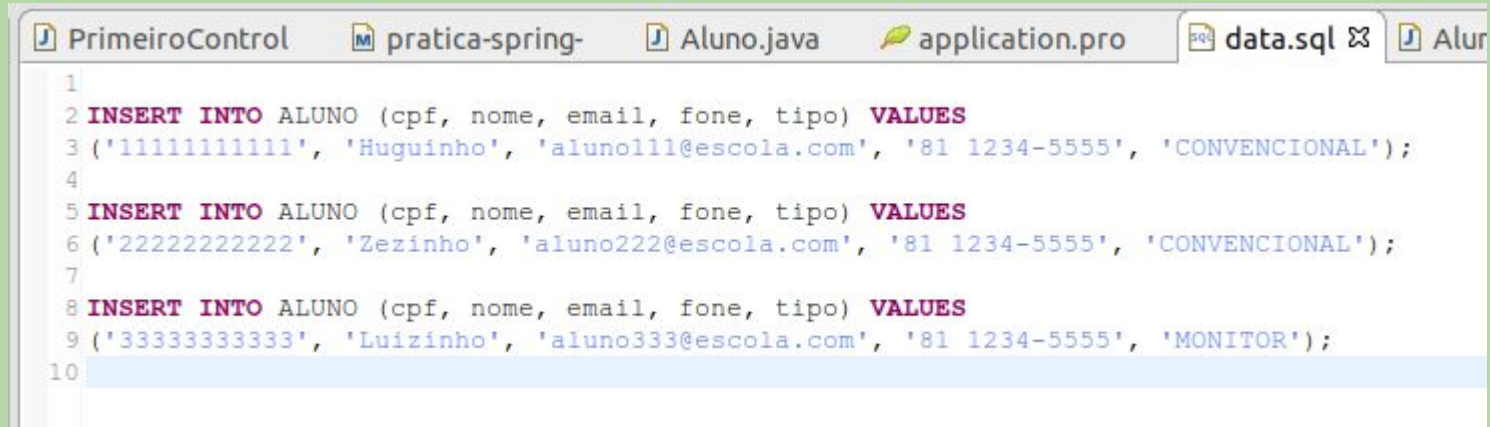
    @Column(nullable = false, name = "FONE")
    private String fone;

    @Column(nullable = false, name = "TIPO")
    private String tipo = TipoAluno.CONVENCIONAL.toString();

    public Aluno() {
```



# Arquivo data.sql



```
1
2 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
3 ('11111111111', 'Huguinho', 'aluno111@escola.com', '81 1234-5555', 'CONVENCIONAL');
4
5 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
6 ('22222222222', 'Zezinho', 'aluno222@escola.com', '81 1234-5555', 'CONVENCIONAL');
7
8 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
9 ('33333333333', 'Luizinho', 'aluno333@escola.com', '81 1234-5555', 'MONITOR');
10
```

# Console H2

English ▼ Preferences Tools Help

---

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

---

Driver Class: org.h2.Driver

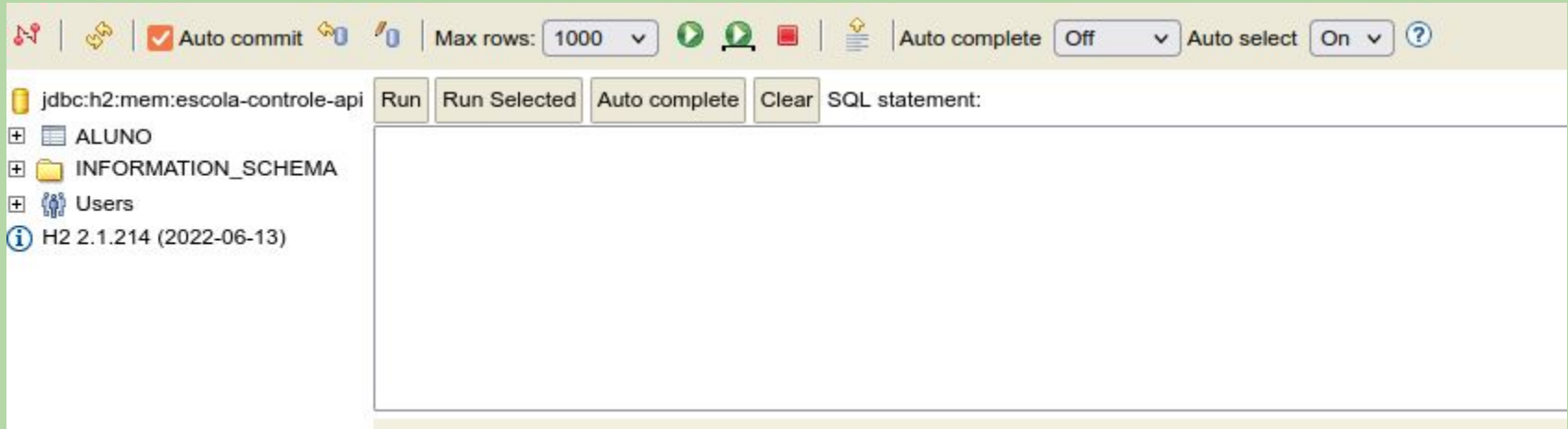
JDBC URL: jdbc:h2:mem:escola-controle-api

User Name: sa

Password:

Connect Test Connection

# Console H2



# Repository



- Repositório - lugar para armazenar objetos
- Repository é uma interface pré-definida responsável pela camada de dados (persistence)
- Na prática: interface que estenderá JpaRepository<Model, Id>
- Recebe por herança todos os métodos da classe abstrata
- Ver os métodos na documentação:

<https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

# Repository

```
package br.com.fectura.escola.repository;

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

import br.com.fectura.escola.model.Aluno;

public interface AlunoRepository extends JpaRepository<Aluno, Long> {

    //
}
```

# Aula 04 - Alvos para hoje

- Criar os endpoints para os métodos GET/POST/PUT e DELETE
- Melhorar o método de listagem de alunos
- Paginação e Ordenação
- Cache de consultas



# Melhorando o método listarAlunos()

- Adicionar o parâmetro (não obrigatório) de pesquisa para o nome do Aluno
- `@RequestParam(required = false) String nomeAluno`

# Paginação

- Adicionar nova anotação na classe principal do sistema **@EnableSpringDataWebSupport**
- Adição de mais um parâmetro para gerenciar a paginação
- @PageableDefault(sort = "id", direction = Direction.ASC, page = 0, size = 10) Pageable paginacao)
- Método agora retorna uma coleção de Page<AlunoDto>



# Ordenação

- @PageableDefault(sort = "id", direction = Direction.ASC, **page** = 0, **size** = 10) Pageable paginação)
- Com a paginação já configurada, é só gerenciar o número da página (page) e a quantidade de registros (size)

# Paginação e Ordenação

- Para testar a paginação, melhorar o arquivo data.sql

```
1
2 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
3 ('11111111111', 'Alberto', 'aluno111@escola.com', '81 1234-5555', 'CONVENCIONAL');
4
5 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
6 ('22222222222', 'Bruno', 'aluno222@escola.com', '81 1234-5555', 'CONVENCIONAL');
7
8 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
9 ('33333333333', 'Carlos', 'aluno333@escola.com', '81 1234-5555', 'MONITOR');
10
11 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
12 ('44444444444', 'Daniel', 'aluno111@escola.com', '81 1234-5555', 'CONVENCIONAL');
13
14 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
15 ('55555555555', 'Emerson', 'aluno222@escola.com', '81 1234-5555', 'CONVENCIONAL');
16
17 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
18 ('66666666666', 'Fernando', 'aluno333@escola.com', '81 1234-5555', 'MONITOR');
19
20 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
21 ('77777777777', 'Guilherme', 'aluno111@escola.com', '81 1234-5555', 'CONVENCIONAL');
22
23 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
24 ('88888888888', 'Heitor', 'aluno222@escola.com', '81 1234-5555', 'CONVENCIONAL');
25
26 INSERT INTO ALUNO (cpf, nome, email, fone, tipo) VALUES
27 ('99999999999', 'José', 'aluno333@escola.com', '81 1234-5555', 'MONITOR');
```

# Cache de Consultas

- Adicionar nova dependência que dá suporte ao cache de consultas
- Na classe principal do sistema, adicionar a anotação **@EnableCaching**
- No método da consulta, adicionar a anotação `@Cacheable(value = "xpto")`

# Cache de Consultas

- Adicionar 2 novas linhas no application.properties:

**spring.jpa.properties.hibernate.show\_sql=true**

**spring.jpa.properties.hibernate.format\_sql=true**

# Cache de Consultas

- Nos métodos que não podem ter cache, adicionar nova anotação para evitar o cache

**@CacheEvict(value = "xpto", allEntries = true)**

# Aula 05 - O quanto já avançamos

- IDE
- Atalhos do Eclipse
- Ecossistema Spring
- Fundamentos Básicos do Spring Boot
- Spring Initializr
- Anotação @SpringBootApplication
- Apache Maven



# Aula 05 - O quanto já avançamos

- Spring Core
- Inversão de Controle
- Injeção de Dependência
- Beans e Estereótipos
- API
- Arquivos Json
- Arquitetura REST



# Aula 05 - O quanto já avançamos

- Padrão MVC
- RestController
- Padrão DTO
- JPA
- Arquivo application.properties
- Arquivo data.sql
- Console H2





# Aula 05 - O quanto já avançamos

- Interface JpaRepository
- Paginação
- Ordenação
- Cache de Consultas



# Aula 05 - Alvos para hoje

- Monitorando nossa aplicação com Spring Actuator
- Documentando sua API com Spring Doc Open Api
- Melhorando seu código com Lombok
- Projeto da API da Escola





# Nosso Projeto

- Criando nosso primeiro Controller
- Criação dos pacotes de código: model, repository e controller
- Dentro do pacote model, importar as classes de entidade do nosso projeto

ALUNO

PROFESSOR

CURSO

TURMA

MATRICULA

# Links Úteis

- Site do Spring <https://spring.io/>
- Documentação do Spring Boot <https://spring.io/projects/spring-boot>
- Linguagens, frameworks e tecnologias mais usadas no mundo  
<https://snyk.io/jvm-ecosystem-report-2021/>
- Maven <https://maven.apache.org/>
- Eclipse IDE <https://www.eclipse.org/>
- Ecossistema Spring <https://spring.io/projects>
- REST - conceitos e fundamentos  
<https://www.alura.com.br/artigos/rest-conceito-e-fundamentos>
- 
- API REST  
[https://darvishdarab.github.io/cs421\\_f20/docs/readings/restful/api/](https://darvishdarab.github.io/cs421_f20/docs/readings/restful/api/)