



# Spring Boot API

Desenvolvendo nossa API de backend para a SPA de Checklist

Carlos A J Lazarin





## O que irei aprender no Back-end?

1. O que é Spring Boot e porque utilizá-lo;
2. Como criar minha primeira aplicação com Spring Boot;
  - a. Maven ou Gradle?
  - b. Dependências do Spring (H2, Web, Data, Actuator etc.)
3. Configurar ambiente de desenvolvimento:
  - a. Instalar JDK;
  - b. Instalar e utilizar IntelliJ IDEA;



## O que irei aprender no Back-end?

3. Desenvolver uma API REST e seus principais conceitos;
  - a. Métodos HTTP;
  - b. Como definir endpoints;
  - c. Códigos de resposta e como utilizá-los;
  - d. Paginação e filtros;
4. Especificação da API REST: Spring Docs e Open API 3.0;



## O que irei aprender no Back-end?

5. Configurar uma instância de banco de dados local;
6. Testes unitários;
7. Testar a camada web com `@WebMvcTest`
8. Monitorar a API com Spring Actuator;

# Atividade Prática - API Checklist

---

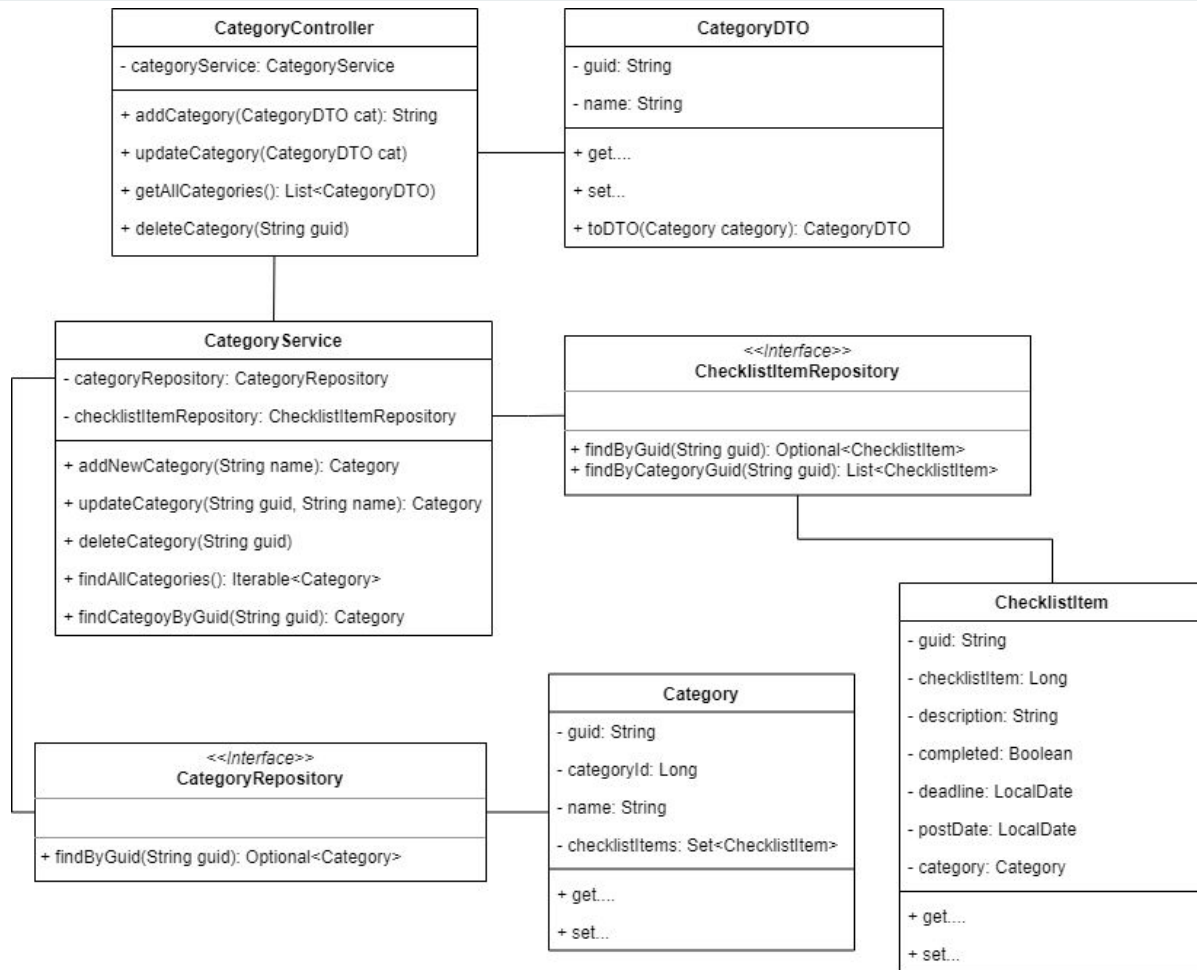
## Visão Geral

- Criação de uma API com Spring Boot para prover os **resources** para a SPA de Checklist



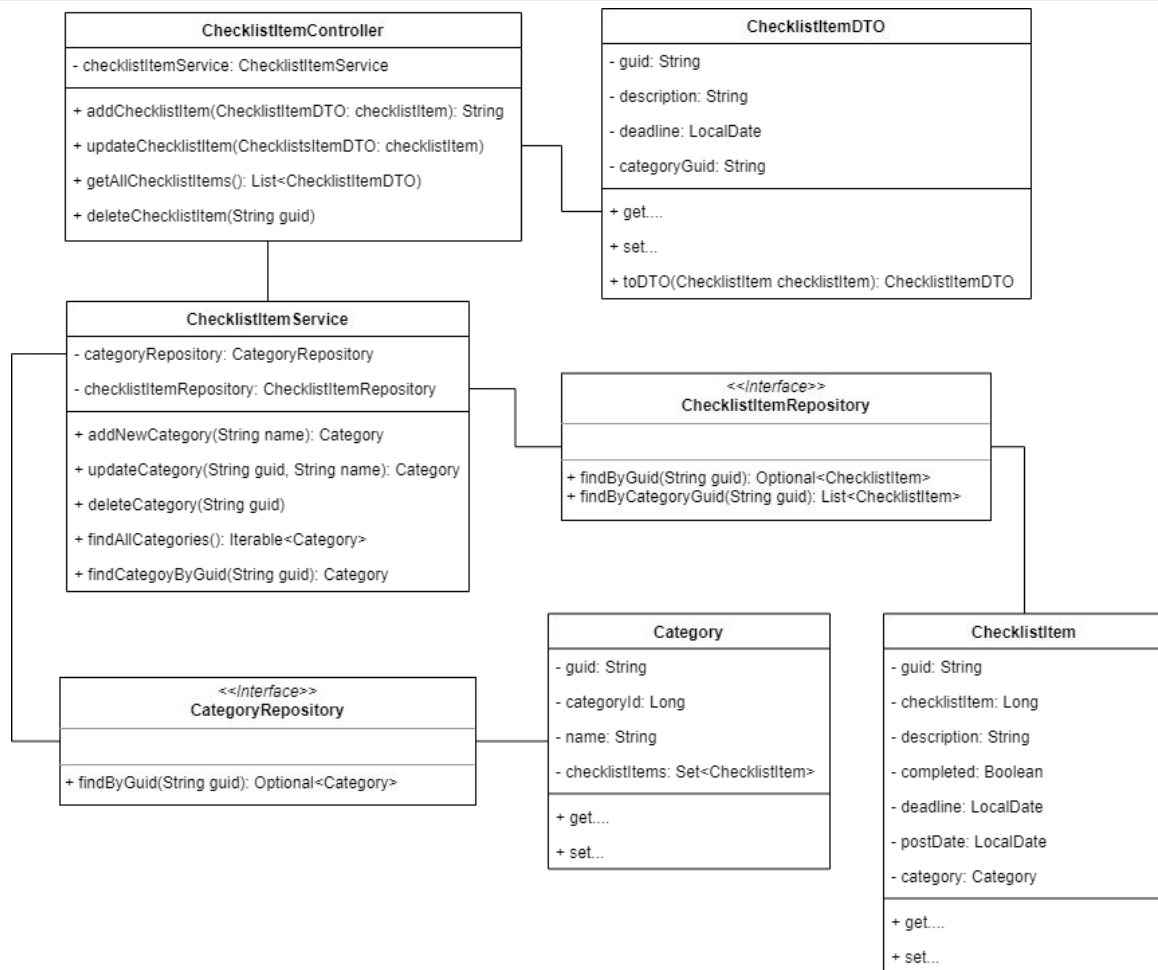
# Diagrama de Classe

## Parte 1



# Diagrama de Classe

## Parte 2





# Maven ou Gradle?

---



## Gradle ou Maven?

- Você pode utilizar Gradle ou Maven (instalado ou wrappers);
- Instalação Gradle: <https://gradle.org/install/>
- Instalação do Maven: <https://maven.apache.org/download.cgi>



# Configurando o Ambiente de Desenvolvimento

---



# Configurando o ambiente Java (Windows)

## Download JDK

1. Acesse o site de download da JDK: <https://www.oracle.com/java/technologies/javase-downloads.html>.
2. Em "Oracle JDK", clique em "JDK Download";
3. Faça o download do executável com o nome de "Windows x64 Installer";
4. Execute o instalador e siga os passos para concluir a instalação;
5. Configure o Java path (a partir do Java 15 não é mais necessário);
  - a. Copy our Java installation folder path, add ("/bin") and put this value in your environment variables under "PATH" key;



# Editar a variável de ambiente PATH no Windows 10:

Para editar a variável de ambiente PATH no Windows 10:

1. Abra o "Painel de Controle" ⇒ "Sistema" ⇒ Clique em "Configurações avançadas de sistema" no painel esquerdo;
2. Mude para a aba "Avançado" ⇒ clique no botão "Variáveis de ambiente";
3. Em "Variáveis de Sistema", desça até a variável "Path" ⇒ Clique em "Editar...".



## Verificação a instalação da JDK (Windows 10)

Abra uma janela do Prompt Command:

1. Digite ``javac -version`` e ``java -version``
2. Se tudo estiver 100%, você deverá ver como resposta a versão do Java recém instalada;

# Configurando o Java no MacOS ou distribuição Linux?


Eu recomendo a utilização do HomeBrew:

- [https://brew.sh/index\\_pt-br](https://brew.sh/index_pt-br)




# Instalando o IntelliJ IDEA

- Para codificação da API eu recomendo o IntelliJ IDEA
  - <https://www.jetbrains.com/idea/download/#section=windows>
- Sinta-se livre para utilizar a IDE que mais lhe agrada

 **IntelliJ IDEA**

Coming in 2021.1 | [What's New](#) | [Features](#) | [Resources](#)



Version: 2020.3.3  
Build: 203.7717.56  
15 March 2021  
[Release notes](#)

[System requirements](#)  
[Installation Instructions](#)  
[Other versions](#)

## Download IntelliJ IDEA

[Windows](#) | [macOS](#) | [Linux](#)

### Ultimate

For web and enterprise development

[Download](#) | [.exe](#)

Free 30-day trial

### Community

For JVM and Android development

[Download](#) | [.exe](#)

Free, open-source

	IntelliJ IDEA Ultimate	IntelliJ ID
Java, Kotlin, Groovy, Scala	✓	✓



# Instalando o Maven

---



## Processo de instalação Maven (Windows)

- Acesse: <https://maven.apache.org/download.cgi>
- Faça o download da versão binária (\*.zip)
- Adicione o caminho da pasta \*\bin ao PATH
  - <https://maven.apache.org/install.html>
- Teste a instalação:
  - Em um janela do Prompt Command: mvn -version



# Instalando o Gradle

---

## Processo de Instalação do Gradle (Windows)

- Faça o download da versão binária:
  - <https://gradle.org/releases/>
- Descompacte o conteúdo em uma pasta de sua preferência;
- Adicione o caminho da pasta \*\bin ao PATH
- Verifique a instalação:
  - Em uma janela do Prompt Command: `gradle -v`



# Spring Framework

---

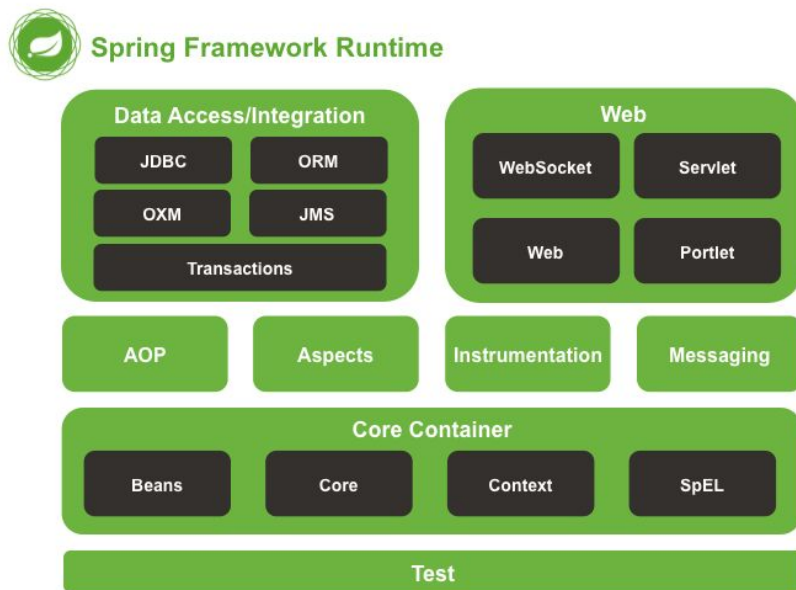


# Spring Framework

- Framework mais popular para a criação de aplicações Java
  - Micro-serviços
  - Cloud
  - Reactive programming
  - Servless
  - Batch
  - Web apps
- Open source, versátil e focado na produtividade;
- Imensa comunidade de desenvolvedores e seguro;
- Em seu core: Inversão de controle (Ioc) e Injeção de dependências (DI)



# Spring Framework





# Spring Boot

- Stand-alone Spring applications;
- *Embedded* Tomcat ou Jetty (não é necessária a criação de um war e fazer deploy em um servidor de aplicação);
- Construção simplificada por meio de dependências (também auto-configuráveis);
- Configurável com annotations e sem XML (mínima quantidade de código possível).







# Spring Boot

@SpringBootApplication:


- @Configuration: indica que é uma classe de configuração do Spring
- @EnableAutoConfiguration: carrega os beans mapeados no classpath;
- @EnableWebMvc: demarca a aplicação como uma aplicação Web e configura o *Dispatch Servlet*;
- @ComponentScan: lê os pacotes da aplicação procurando por beans, services e components;

**É hora de codificar!**

---

# Spring Initializer

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



☒ Maven Project

☐ Gradle Project

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 2.5.0 (SNAPSHOT) ☐ 2.5.0 (M3) ☐ 2.4.5 (SNAPSHOT) ☒ 2.4.4 ☐ 2.3.10 (SNAPSHOT) ☐ 2.3.9

Project Metadata

Group

com.lazarin.learning.sample.app

Artifact

checklist-resource-server

Name

checklist-resource-server

Description

Checklist Application For Udemy

Package name

com.lazarin.learning.sample.app.checklist-resource-server

Packaging

☒ Jar ☐ War

Java

☐ 16 ☒ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

H2 Database


SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.



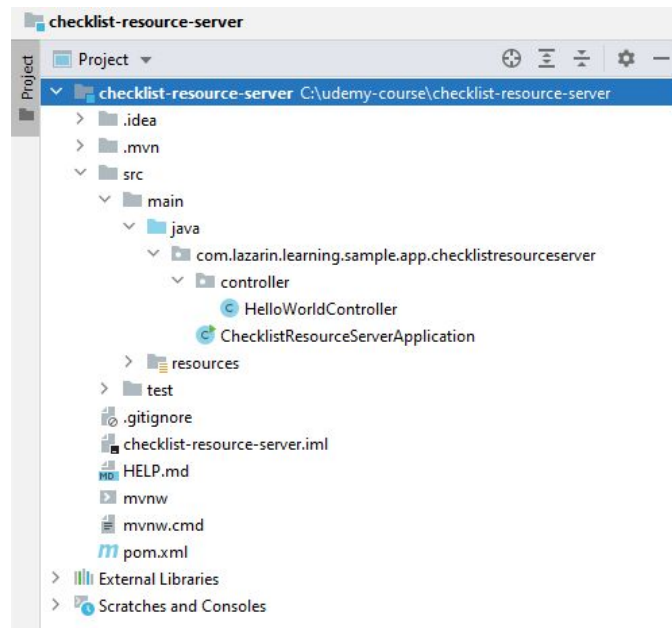
GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

# Importar o projeto no IntelliJ e rodar o build

- Importe o projeto no IntelliJ;
- Crie um novo controller chamado HelloWorldController;
- Clique em build para que o Maven/Gradle construa a aplicação;
- Você pode utilizar o Maven wrapper se você preferir:
  - Abra uma janela do Prompt Command;
  - Digite `mvnw spring-boot:run``
  - Abra o browser e digite:
    - `http://localhost:8080/hello?name=YourName`





## Inicializando o repositório Git para o BE

- Abra uma nova janela do Prompt Command;
- Vá até a pasta do projeto e digite:
  - `git init`
  - Você deverá ver uma mensagem indicação a criação do repositório com sucesso!
- Faça o commit de todas as alterações até então
  - `git add *`
  - `git commit -m "sua_mensagem_aqui"`

# Criando as Entidades

---



# Spring Data

- Fornece meios de acesso à dados seguindo os padrões de desenvolvimento do Spring Framework;
  - Bancos de dados relacionais
  - Bancos de dados não relacionais
  - Serviços baseados em cloud
- Projeto base para sub-projetos e implementações de diversos BDs
- Benefícios:
  - Forte abstração à nível de repositório e mapeamento de dados
  - Criação de dinâmica de queries
  - Fácil integração por meio de JavaConfig
- Principais módulos: <https://spring.io/projects/spring-data>



# Spring Data e Mapeamento de Entidades

- Verifique se a aplicação já possui como dependência o artefato Spring Data JPA
- Caso não tenha, adicionei a seguinte dependência ao pom.xml:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

- Agora é hora de mapear as entidades e os relacionamentos:
  - com.learning.api.checklist.entity
    - Base.java
    - Category.java
    - Checklist.java





## O que é um banco de dados H2?

- Banco de dados de código de livre escrito em Java;
- Rápido e pequeno;
- Principal uso: banco de dados em memória;
- Pode ser utilizado também para persistência de dados;
- Não recomendado para produção;
- Ideal para POCs.



## Adicionando as configurações do BD H2

- application-local.yml

```
spring:
  jpa:
    hibernate.ddl-auto: update
    database-platform: org.hibernate.dialect.H2Dialect
  datasource:
    url: jdbc:h2:mem:checklist-app-db
    driverClassName: org.h2.Driver
    username: sa
    password: password
  h2:
    console:
      path: /h2-console
```



# Adicionando as configurações do MySQL

- application-aws.yml

```
spring:
  datasource:
    url: jdbc:mysql://[db_name].cvhzq4aq6qx6.sa-east-1.rds.amazonaws.com:3306/ebdb
    username: [username]
    password: [password]
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    hibernate:
      ddl-auto: update
    generate-ddl: true
    show-sql: false
```



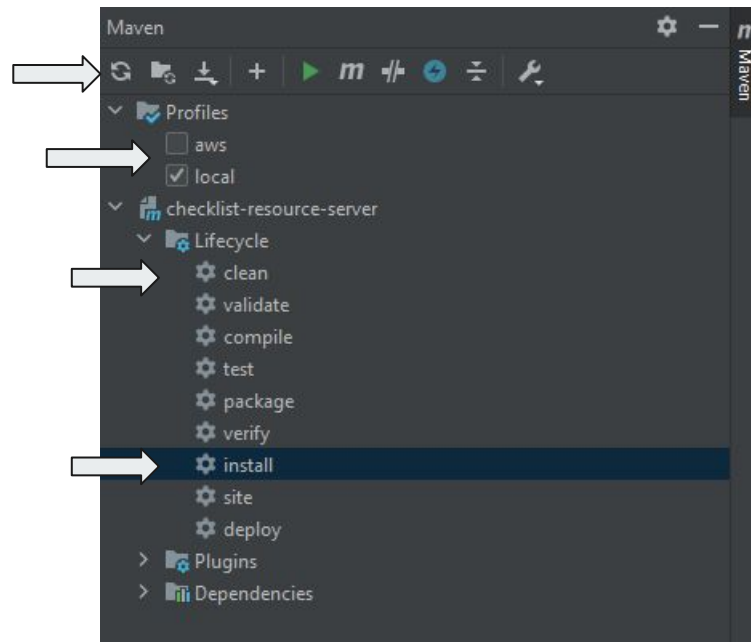
## Configurando o pom.xml

- Iremos criar 2 profiles para adicionar a dependência específica para cada ambiente

```
<profiles>
  <profile>
    <id>local</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <dependencies>
      <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
      </dependency>
    </dependencies>
  </profile>
```

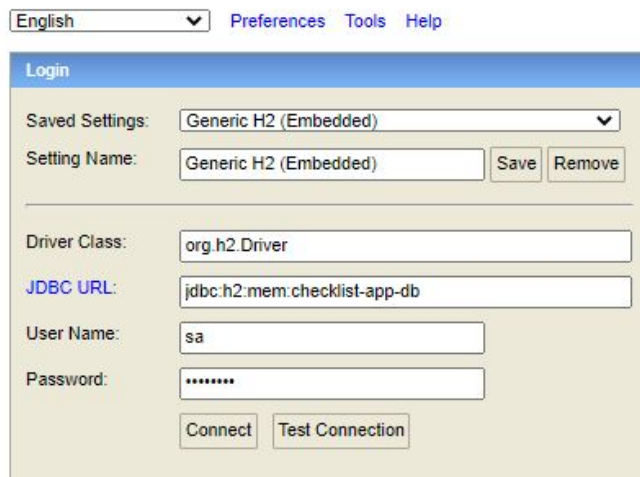
# Configurando o pom.xml

```
<profile>
  <id>aws</id>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.23</version>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
</profile>
</profiles>
```



# Acessando o console do BD H2

- URL: `http://localhost:[porta]/h2-console`



The screenshot shows the H2 Database Console Login dialog box. At the top, there is a language dropdown menu set to 'English' and three links: 'Preferences', 'Tools', and 'Help'. The dialog has a blue header bar with the title 'Login'. Below the header, there are several input fields and buttons. The 'Saved Settings' section includes a dropdown menu currently showing 'Generic H2 (Embedded)'. Below this, the 'Setting Name' field also contains 'Generic H2 (Embedded)', with 'Save' and 'Remove' buttons to its right. The 'Driver Class' field is filled with 'org.h2.Driver'. The 'JDBC URL' field contains 'jdbc:h2:mem:checklist-app-db'. The 'User Name' field is filled with 'sa', and the 'Password' field is filled with seven asterisks. At the bottom, there are two buttons: 'Connect' and 'Test Connection'.

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:checklist-app-db

User Name: sa

Password: .....

Connect Test Connection

# Criando a camada de acesso a dados (Repository)

---

# Criando os Serviços

---





## Serviços - Categoria e ChecklistItem

- As classes de serviço de *Category* e *ChecklistItem* irão conter toda a lógica de negócios, se comunicar com o BD por meio das classes de *repository* e retornar os dados para o controller.

Category Service
- categoryRepository: CategoryRepository - checklistItemRepository: ChecklistItemRepository
+ addNewCategory(String name): Category + updateCategory(String guid, String name): Category + deleteCategory(String guid) + findAllCategories(): Iterable<Category> + findCategoryByGuid(String guid): Category

ChecklistItem Service
- categoryRepository: CategoryRepository - checklistItemRepository: ChecklistItemRepository
+ addNewCategory(String name): Category + updateCategory(String guid, String name): Category + deleteCategory(String guid) + findAllCategories(): Iterable<Category> + findCategoryByGuid(String guid): Category

# O que é uma API RESTFul?

---



# RESTFul APIs

- **API** - *Application Program Interface* (Interface de Programa de Aplicações):
  - Conjunto de instruções/padrões que fornecem informações relevantes à uma determinada aplicação;
  - Grande parte delas são utilizadas para a integração entre sistemas;
  - Utilizam uma rede para se comunicar;
- **REST** - *Representational State Transfer* (Transferência de Estado Representacional);
  - Estilo de arquitetura de software para interoperabilidade de sistemas;
  - Introduzido no ano 2000 por Roy Fielding em sua tese de doutorado;
- **RESTFul** - Sistemas que utilizam os princípios da arquitetura REST;
- Baseada no protocolo HTTP (criado a mais de 20 anos);
- **W3C** - **Recursos**: qualquer entidade que possa ser identificada, manipulada ou endereçada pela web.

# Métodos HTTP e quando utilizá-los

---



## RestFul API - HTTP Methods

Método HTTP	Descrição	Request body?	Spring Annotation
GET	Retorna a representação de um recurso da API (apenas dados)	Não	@GetMapping
POST	Utilizado para enviar dados ao servidor (causam uma mudança de estado; com efeitos colaterais) <u>Não idempotente!</u>	Sim	@PostMapping
PUT	Substitui o estado atual de um recurso. Diferentemente do POST, é idempotente!	Sim	@PutMapping
DELETE	Apaga um recurso específico.	Não	@DeletingMapping



## RestFul API - HTTP Methods

Método HTTP	Descrição	Request body?	Spring Annotation
PATCH	Atualizações <b>parciais</b> à um recurso	Sim	@PatchMapping
HEAD	Idêntico ao GET mas sem <i>response body</i>	Não	@RequestMapping (method=RequestMethod.HEAD)
OPTIONS	Utilizado para o cliente saber quais opções de requisição são permitidas pelo servidor	Não	@RequestMapping (method=RequestMethod.OPTIONS)

# Criando os Controllers

---



## ChecklistItem Controller

- Pacote: com.learning.api.checklist.controller
- Classe de controller: ChecklistItemController.java
- **GET:** getChecklistItems
  - Resposta: ChecklistDTO collection
- **POST:** addChecklistItem
  - Resposta: checklistItem guid
- **PUT:** updateChecklistItem
  - Void
- **DELETE:** deleteChecklistItem
  - Void

ChecklistItemController
- checklistItemService: ChecklistItemService
+ addChecklistItem(ChecklistItemDTO: checklistItem): String
+ updateChecklistItem(ChecklistItemDTO: checklistItem)
+ getAllChecklistItems(): List<ChecklistItemDTO>
+ deleteChecklistItem(String guid)





# Category Controller

- Pacote: com.learning.api.checklist.controller
- Classe de controller: CategoryController.java
- **GET:** getAllCategories
  - Resposta: CategoryDTO collection
- **POST:** addCategory
  - Resposta: category guid
- **PUT:** updateCategory
  - Void
- **DELETE:** deleteCategory
  - Void

CategoryController
- categoryService: CategoryService
+ addCategory(CategoryDTO cat): String
+ updateCategory(CategoryDTO cat)
+ getAllCategories(): List<CategoryDTO>
+ deleteCategory(String guid)

# Bean validations

---



## Adicionando bean validations (JSR-380)

- Objetivo: aplicar regras de validação nas estradas do usuário
- Benefícios: menor verbosidade e melhor leitura do código
- Pacote: `com.learning.api.checklist.dto`
- Classes:
  - `CategoryDTO.java`
  - `ChecklistitemDTO.java`
- **@NotBlank** - utilizado somente para strings, validar se o atributo é nulo ou contém espaços em branco;
- **@NotNull** - verifica se o atributo é nulo

# Spring Docs e Open API 3.0

---



# Documentando a API com Open API 3.0

- SpringDoc: simplifica a criação/manutenção da documentação de uma API utilizando Open API 3.0
  - Especificação Open API: <https://swagger.io/specification/>
- Adicionando dependência ao pom.xml:
  - A documentação da API estará disponível em: [http://localhost:\[porta\]/v3/api-docs/](http://localhost:[porta]/v3/api-docs/)
  - Caso seja necessário alterar o path:
    - Adicionar ao application.yml: *springdoc.api-docs.path=/api-doc*
  - A documentação também está disponível no formato JSON:
    - [path]/api-docs.yaml

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.5.2</version>
</dependency>
```



## Adicionando Swagger UI na API

- Adicione a seguinte dependencia:

```
<dependency>  
  <groupId>org.springdoc</groupId>  
  <artifactId>springdoc-openapi-ui</artifactId>  
  <version>1.5.2</version>  
</dependency>
```

- Swagger UI pode ser acessado na URL [http://localhost:\[porta\]/swagger-ui.html](http://localhost:[porta]/swagger-ui.html)



## Adicionando Swagger UI na API - Paginação

- Para expor informações sobre paginação (tamanho, ordenação etc.), é necessário adicionar a seguinte dependência:

```
<dependency>  
  <groupId>org.springdoc</groupId>  
  <artifactId>springdoc-openapi-data-rest</artifactId>  
  <version>1.5.2</version>  
</dependency>
```

# CORS: Cross Origin Resource Sharing

(compartilhamento de recursos de origens diferentes)

---





## O que é CORS?

- CORS: Cross-Origin Resource Sharing; é uma especificação da W3C
- Utiliza cabeçalhos HTTP para informar o browser e aceitar requisições que se originam de um domínio diferente;
- Por segurança, navegadores bloqueiam requisições cross origin iniciadas por scripts (*Same Origin Policy*);
- Ex.: código JS disponível em um domínio A que faça uma requisição XMLHttpRequest para um domínio B



# CORS

- Exceções ao CORS podem ser configuradas do lado da API
- Inclusão do de *Access-Control-Allow-Origin* no header HTTP de resposta;
- É possível também controlar qual método HTTP será aceito: *Access-Control-Allow-Methods*
- Solicitações **simples** e **não simples**
  - **Simples:** por padrão HEAD, POST e GET sem cabeçalhos personalizados
  - PUT, PATCH, DELETE: *pre-flight request*
- **Pre-flight request:** utiliza HTTP OPTIONS; “pedido de autorização” ao servidor com o header HTTP da requisição “real”
  - O servidor responde com um resposta vazia apenas com cabeçalhos CORS

# Adicionando a configuração de CORS na API de Checklist

---



# Configuração de CORS na API

```
@Bean
public WebMvcConfigurer corsConfig() {
    return new WebMvcConfigurer() {
        @Override
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**")
                .allowedOrigins("http://localhost:4200", "http://127.0.0.1:4200")
                .allowedMethods("POST", "GET", "OPTIONS", "DELETE", "PUT")
                .maxAge(3600)
                .allowedHeaders("Origin", "X-Requested-With", "Content-Type", "Accept", "Authorization");
        }
    };
}
```

# Criando os Testes Unitários

---



# Testando a aplicação

- Testes unitários (JUnit 5)
  - O mais popular framework de teste aprimorado com suporte para Java 8 +
- Testes da camada Web (Controllers) - “testes de integração”
- Utilizando um Rest Client: <https://www.postman.com/>
- Adicionando dependência de teste (JUnit 4 foi removido do SpringBoot 2.4):

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <version>2.2.6.RELEASE</version>
</dependency>
```



# Testando a aplicação

- Caso ainda queira utilizar o JUnit 4 (vintage engine):

```
<dependency>
  <groupId>org.junit.vintage</groupId>
  <artifactId>junit-vintage-engine</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```



## Testes da camada Web

- Objetivo: testar **somente** a API a nível de controller (requisições HTTP)
- **Benefício:** o Spring irá iniciar somente a camada web, apenas o controller a ser testado!
- Annotations:
  - @MockMvc
  - @WebMvcTest([nome\_classe\_controller].class)
  - @MockBean - útil para injetar mocks ao invés da dependência real



# Testes unitários: Testando a camada de serviço

---

# Atualizando serviços do FE com chamadas para a API

---

# Finalizando o Front-End

Implementação do CRUD com o Back-End

