

# **WebServices e MBaaS**

Aula 02 - Padrão REST

Mark Joselli

Mark.Joselli@pucpr.br

<https://www.linkedin.com/in/mjoselli/>

# Agenda

- Aula 1 (12/08/2023) – Introdução de Webservices e consumo;
- Aula 2 (19/08/2023) – Padrão Rest/Rotas e Tipos de serviços
- Aula 3 (02/09/2023) – Autenticação (EAD)
- Aula 4 (16/09/2023) – Banco de dados e MBaaS
- Aula 5 (16/09/2023) – Firebase
- Aula 6 (14/10/2023) – Projeto (EAD)

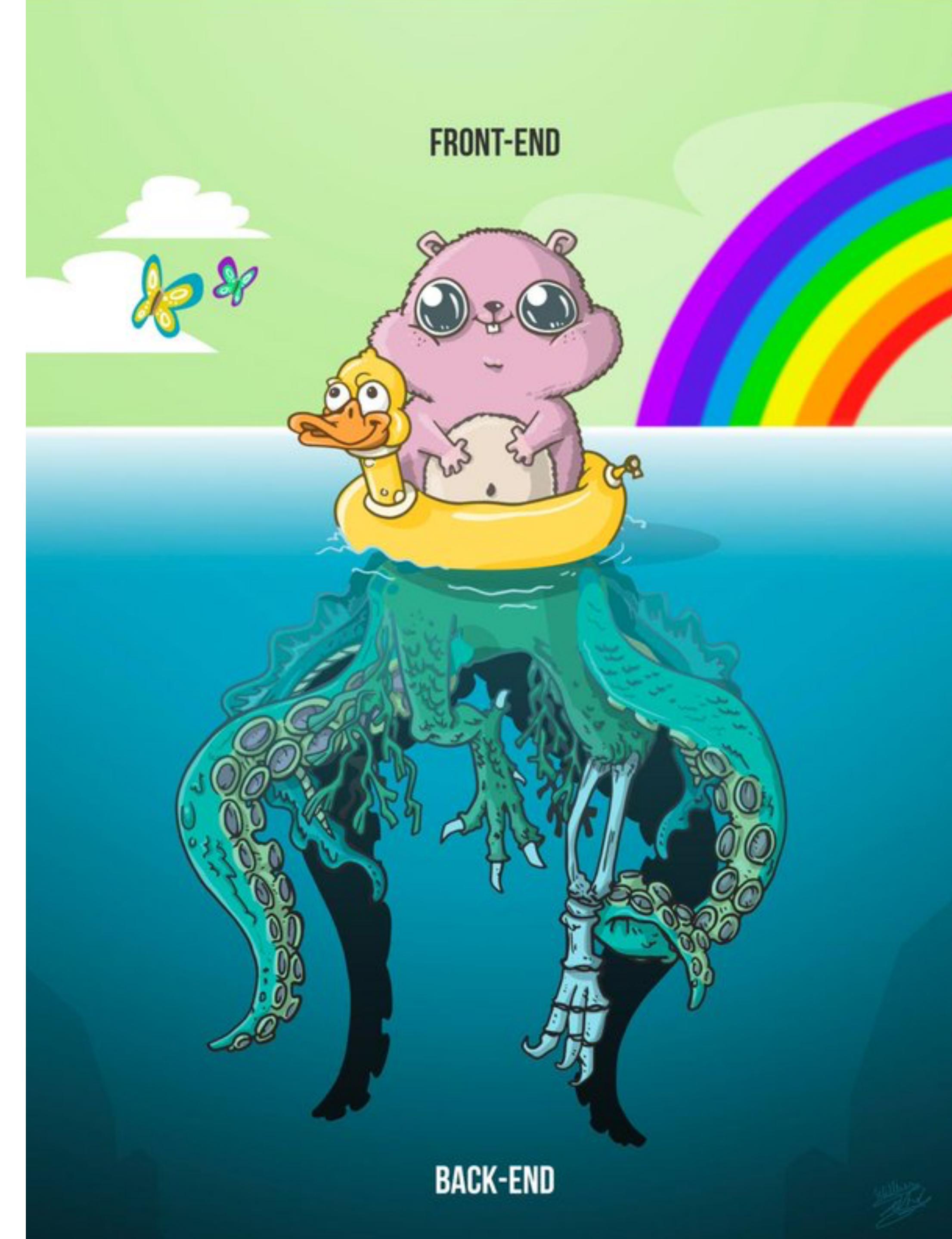
# Agenda

- Revisão REST
- O que precisamos de um backend?
- Introdução ao SpringBoot
- Exercícios

# Objetivos

- Apresentar os termos e conceitos principais desta tecnologia
- Mostrar o uso básico do Spring

Vamos Começar?



# REST

- "Padrão REST" se refere ao estilo arquitetural REST (Representational State Transfer), que é um conjunto de princípios e restrições para a criação e design de serviços web.
- É amplamente utilizado para construir sistemas web escaláveis e interoperáveis.
- A arquitetura REST foi popularizada por Roy Fielding em sua tese de doutorado em 2000.
- Ela é frequentemente aplicada no desenvolvimento de APIs (Interfaces de Programação de Aplicativos) para sistemas distribuídos, onde as ações são realizadas através dos métodos HTTP (como GET, POST, PUT, DELETE) e os recursos são representados por URLs.

# REST - Princípios Fundamentais

- Cliente-Servidor: Separação entre a interface do usuário (cliente) e a lógica de negócios (servidor), permitindo que eles evoluam independentemente.
- Stateless (Sem Estado): Cada solicitação do cliente para o servidor contém todas as informações necessárias para compreender e processar a solicitação, o que facilita a escalabilidade.
- Cache: As respostas devem ser marcadas como cacheáveis ou não-cacheáveis, permitindo que os clientes reutilizem as respostas quando apropriado.
- Interface Uniforme: Os recursos são identificados por URLs, e as operações sobre esses recursos são realizadas usando métodos HTTP padronizados. Isso inclui o uso de CRUD (Create, Read, Update, Delete) para ações sobre recursos.
- Sistema em Camadas: Permite a inclusão de intermediários, como proxies, caches e gateways, que podem melhorar a escalabilidade e a segurança.
- Código sob Demanda (Opcional): Os servidores podem transmitir código executável para os clientes, por exemplo, em linguagens como JavaScript. Isso é opcional e nem sempre é utilizado.

# REST

- Cada url deve representar um recurso;
- Ideal para CRUD (Criar, Ler, Atualizar e Excluir);
- Geralmente, via método GET/POST, cada recurso deve ser diferenciável;
- Uso de padrão de chamadas: GET, PUT, POST e DELETE
  - I **get** a page, I **post** an update, I **delete** a photo, I **put** up my information.
- Uso de cache
- NÃO TEM WSDL!
- Retorno livre: XML; JSON; Etc.
- <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

# Get



- O usuário digita no navegador: `http://www.amazon.com`
  - O navegador cria uma solicitação HTTP (sem corpo)
  - A solicitação HTTP identifica:
    - A ação desejada: GET ("obtenha-me recurso")
    - A máquina de destino (`www.amazon.com`)

# POST

- O usuário preenche um formulário em uma página da web.
- O navegador cria uma solicitação HTTP com um corpo composto pelos dados do formulário
- A solicitação HTTP identifica:
  - A ação: POST ("aqui estão algumas informações atualizadas")
  - A máquina de destino (amazon.com)
- O corpo contém:
  - Os dados que estão sendo POSTados (os dados do formulário)



# CRUD

- CRUD é um acrônimo que representa as quatro operações básicas realizadas em sistemas de gerenciamento de banco de dados ou em sistemas que envolvem manipulação de dados. Cada letra do acrônimo corresponde a uma operação específica:
- Create (C): Esta operação envolve a criação de novos dados ou registros no sistema. Por exemplo, criar um novo usuário, adicionar um novo produto a um catálogo ou inserir uma nova entrada em um banco de dados.
- Read (R): A operação de leitura envolve a recuperação de dados existentes do sistema. Isso pode incluir a busca por informações específicas ou a listagem de todos os registros. Por exemplo, recuperar detalhes de um usuário, visualizar uma lista de produtos ou exibir informações de um artigo.
- Update (U): Essa operação consiste em atualizar ou modificar dados existentes no sistema. Isso pode envolver a alteração de informações de um registro, como atualizar o endereço de um cliente ou modificar as propriedades de um item.
- Delete (D): A operação de exclusão envolve a remoção de dados ou registros do sistema. Isso pode ser usado para eliminar informações que não são mais necessárias. Por exemplo, excluir um registro de usuário, remover um produto que não é mais vendido ou deletar um comentário.
- O CRUD é uma estrutura básica que é amplamente utilizada em sistemas de software, especialmente aqueles que envolvem interação com bancos de dados ou armazenamento de dados. Muitas aplicações web e móveis implementam essas operações para permitir aos usuários criar, visualizar, atualizar e excluir dados de maneira organizada e eficiente. Além disso, as operações CRUD também são frequentemente usadas em APIs REST para representar ações sobre recursos.

# API HTTP

- HTTP fornece um conjunto simples de operações.
- Surpreendentemente, todas as trocas na Web são feitas usando esta simples API HTTP.
- Baseado na ideia de CRUD (Criar, Recuperar, Atualizar e Excluir)
  - POST: “Aqui estão algumas novas informações” (Criar)
  - GET: “Dê-me algumas informações” (Recuperar)
  - PUT: “Aqui estão algumas informações atualizadas” (Atualização)
  - DELETE: “Livre-se desta informação” (Delete)

# REpresentational State Transfer

- Um cliente faz referência a um recurso usando um URI.
- Uma representação do recurso é retornada.
  - Receber a representação coloca o cliente em um novo estado.
- Quando o cliente seleciona um hiperlink em Boeing747.html, ele acessa outro recurso.
  - A nova representação coloca o cliente em outro estado.
    - Assim, o aplicativo cliente transfere o estado com cada representação de recurso.



# Ideia Base

- REST é modelado a partir do fluxo de trabalho da Internet.
- Motivação: crie um padrão de design de como a web funciona, de modo que possa servir de guia para projetar serviços web.
- Um aplicativo web bem projetado se comporta como um rede de páginas da web (uma máquina de estado virtual).
- O usuário progride em um aplicativo selecionando links (transições de estado).
- Resultando na próxima página (o próximo estado do aplicativo) sendo transferido para o usuário e renderizado para seu uso.

# REST não é um protocolo

- REST não é um protocolo.
  - O W3C não publicará uma especificação REST.
  - A Microsoft não vende um kit de ferramentas do desenvolvedor REST.
- REST é apenas um padrão de arquitetura.
- REST prescreve o uso de padrões:
  - HTTP, URL
  - XML/HTML/JPEG/etc. (Representações de recursos)
  - texto/xml, texto/html, imagem/gif, imagem/jpeg, etc. (Tipos de recursos, tipos MIME)

# Verbos em REST

- Verbos (vagamente) descrevem ações que são aplicáveis a substantivos
- Usar verbos diferentes para cada substantivo impossibilitaria a comunicação generalizada
- Alguns verbos se aplicam apenas a alguns substantivos
  - Na programação chamamos isso de “polimorfismo”
- Em REST, usamos verbos universais
  - Todos os serviços RESTful oferecem a mesma interface.
  - Com base em solicitações e respostas HTTP.

# Verbos adicionais

- HEAD
  - Prove os metadados
- TRACE
  - Verificar mudanças realizadas
- OPTION
  - Verifique quais verbos estão implementados para um recursos específico.
- PATH
  - Modifica parcialmente um recurso.

# Verbos

- GET, OPTIONS, TRACE e HEAD são operações seguras.
  - Eles não devem alterar o recurso de forma alguma.
- PUT e DELETE são idempotentes.
  - Solicitações repetidas devem ter o mesmo efeito que uma único pedido.
  - As operações seguras também são idempotentes.
  - POST não é idempotente.

# Fundamentos REST

- Os serviços oferecem recursos com os quais se pode interagir.
- Todos os recursos têm um URI exclusivo.
  - Frequentemente endereços da web (blah.com/form.html)
  - Não necessariamente HTML - os recursos são apenas conceitos.
  - URIs informam a um cliente que há um conceito em algum lugar.
    - Os clientes podem então solicitar uma tipo específico do conceito das representações que o servidor disponibiliza.
- Os verbos HTTP são usados para recuperar ou manipular recursos de maneira clara e universal.

# Formatos dos recursos

- Tanto o cliente quanto o servidor devem ser capazes de compreender o formato.
  - O conteúdo estruturado geralmente é JSON ou XML
- Uma representação deve representar completamente um recurso.
  - Se houver necessidade de representar parcialmente um recurso, quebrar o recurso em recursos filho.
  - Representação menor = mais fácil de transferir = menos tempo necessário para criar a representação = mais rápido o Serviço.
- A representação deve ser capaz de vincular recursos entre si por meio de URIs.

# Spring



spring

# Spring

- O Spring Framework é um framework de desenvolvimento de aplicativos Java que fornece um ambiente abrangente para construir diversos tipos de aplicativos, desde aplicativos corporativos até aplicativos web, desktop e móveis.
- Ele foi criado para abordar muitos dos desafios e complexidades encontrados no desenvolvimento Java, tornando-o mais modular, flexível e fácil de manter.
- O Spring Framework é uma ferramenta muito popular e influente na comunidade Java.
- Ele fornece uma ampla gama de recursos que ajudam os desenvolvedores a criar aplicativos robustos, flexíveis e de alta qualidade.
- O framework está em constante evolução e é amplamente adotado para o desenvolvimento de projetos Java em diversas áreas da indústria.

# Spring - principais aspectos

- Inversão de Controle (IoC): Uma característica central do Spring é a IoC, onde o controle sobre a criação e gerenciamento de objetos é transferido para o próprio framework. Isso reduz o acoplamento entre componentes e facilita a substituição de partes do sistema.
- Injeção de Dependência: A IoC é frequentemente realizada por meio da injeção de dependência, onde as dependências de um objeto são injetadas nele, em vez de serem criadas dentro do objeto. Isso permite maior flexibilidade, testabilidade e reutilização de código.
- Aspect-Oriented Programming (AOP): O Spring oferece suporte ao desenvolvimento baseado em aspectos, permitindo a modularização de preocupações transversais, como logging, segurança e transações, de forma mais limpa e separada do código principal.

# Spring - principais aspectos

- Módulos: O Spring é dividido em vários módulos que podem ser utilizados de acordo com as necessidades do projeto. Alguns exemplos incluem módulos para IoC, AOP, acesso a dados (JDBC, ORM), web (MVC), segurança, entre outros.
- Spring MVC: O módulo Spring MVC oferece suporte para desenvolver aplicativos web baseados no padrão MVC (Model-View-Controller). Ele facilita a criação de controladores, visualizações e integração com tecnologias web.
- Spring Data: O Spring Data simplifica o acesso a bancos de dados ao fornecer abstrações para diferentes tecnologias de armazenamento de dados, como JDBC, JPA (Java Persistence API) e NoSQL.

# Spring - principais aspectos

- Spring Security: O Spring Security oferece recursos abrangentes para lidar com autenticação, autorização e proteção de aplicativos. Ele é usado para implementar recursos de segurança em aplicativos web e não web.
- Spring Boot: Como mencionado anteriormente, o Spring Boot é uma extensão do Spring Framework que visa simplificar o processo de configuração e desenvolvimento de aplicativos, proporcionando configurações padrão e recursos de autoconfiguração.
- Event Handling: O Spring Framework oferece um sistema de eventos que permite que objetos se comuniquem por meio de eventos, permitindo o desacoplamento entre componentes.

# Por que Spring?

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



# Spring Framework Modules

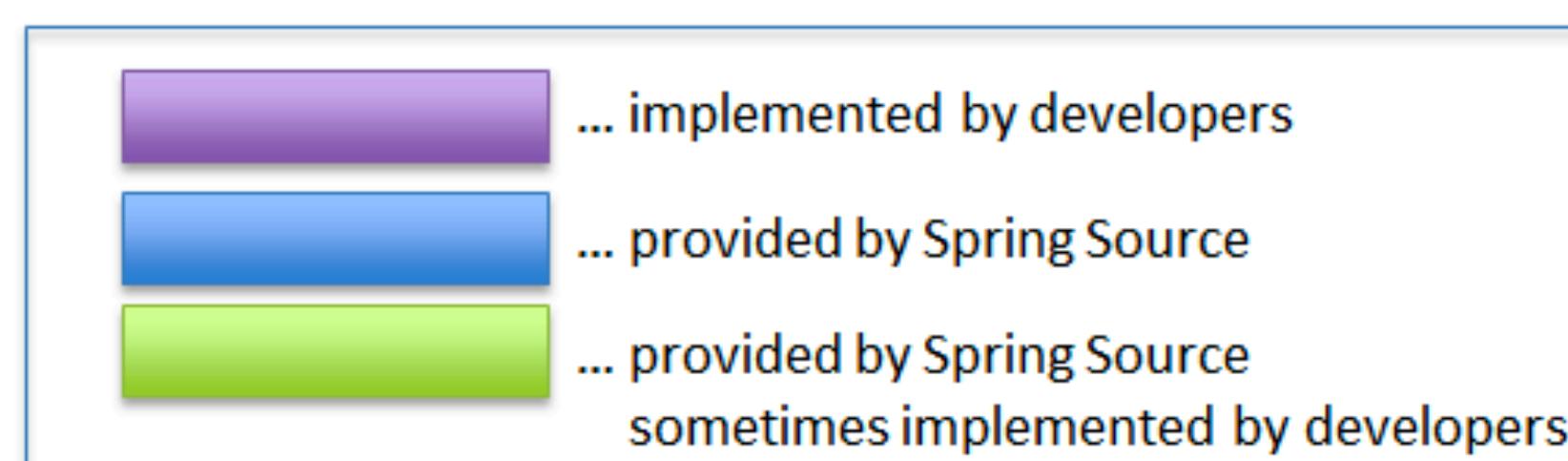
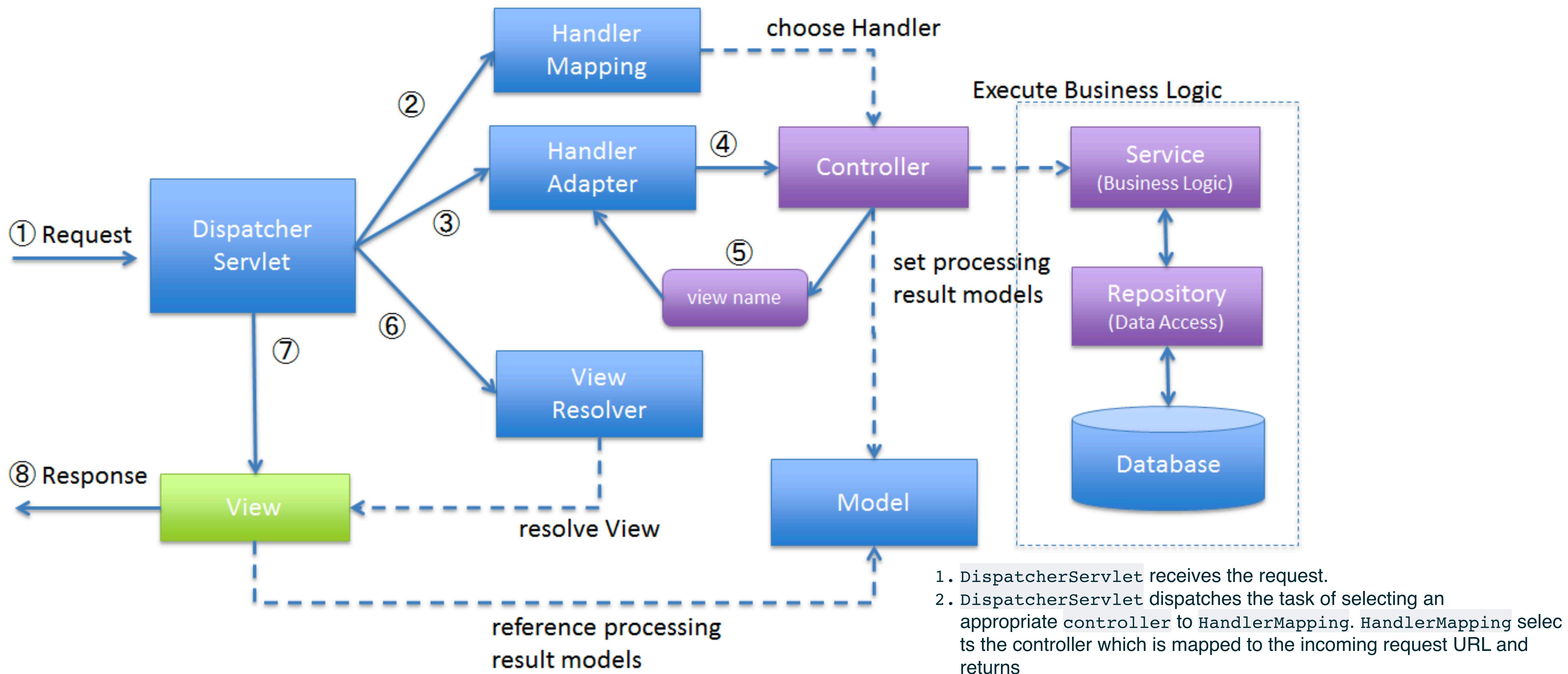
**Dependency Injection (DI)**

**Inversion of Control (IoC)**

**Aspect Oriented Programming (AOP)**

**Model View Controller (MVC)**

**Transaction Management**



1. **DispatcherServlet** receives the request.
2. **DispatcherServlet** dispatches the task of selecting an appropriate controller to **HandlerMapping**. **HandlerMapping** selects the controller which is mapped to the incoming request URL and returns the (**selected Handler**) and **Controller** to **DispatcherServlet**.
3. **DispatcherServlet** dispatches the task of executing of business logic of **Controller** to **HandlerAdapter**.
4. **HandlerAdapter** calls the business logic process of **Controller**.
5. **Controller** executes the business logic, sets the processing result in **Model** and returns the logical name of view to **HandlerAdapter**.
6. **DispatcherServlet** dispatches the task of resolving the **View** corresponding to the **View name** to **ViewResolver**. **ViewResolver** returns the **View** mapped to **View name**.
7. **DispatcherServlet** dispatches the rendering process to returned **View**.
8. **View** renders **Model** data and returns the response.



# Tutorial

# SpringBoot Criando o projeto

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

<https://start.spring.io/#!type=maven-project&language=kotlin&platformVersion=3.1.2&packaging=jar&jvmVersion=17&groupId=com.example&artifactId=demo&name=demo&description=Demo%20project%20for%20Spring%20Boot&packageName=com.example.demo&dependencies=web.thymeleaf>

# SpringBoot Criando o projeto

 **spring initializr**

---

**Project**  
 Gradle - Groovy    Gradle - Kotlin  
 Maven

**Language**  
 Java    Kotlin    Groovy

**Dependencies** [ADD DEPENDENCIES... ⌘ + B](#)

---

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

---

**Thymeleaf** TEMPLATE ENGINES  
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

---

**Project Metadata**

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 20 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

---

[GENERATE ⌘ + ↵](#) [EXPLORE CTRL + SPACE](#) [SHARE...](#)

# Tipo de projeto

- Projetos Gradle com Groovy:
- Linguagem de Configuração: No Gradle com Groovy, a configuração do projeto é feita usando a linguagem de programação Groovy. A sintaxe é mais parecida com uma linguagem de script.
- Flexibilidade: O Groovy oferece uma sintaxe flexível e dinâmica, o que pode ser vantajoso para configurações complexas e personalizações específicas.
- Curva de Aprendizado: Se você já está familiarizado com Groovy ou com linguagens de script em geral, pode encontrar mais facilidade em usar essa abordagem.

# Tipo de projeto

- Projetos Gradle com Kotlin:
- Linguagem de Configuração: O Gradle com Kotlin permite que você configure seus projetos usando a linguagem de programação Kotlin. Kotlin é uma linguagem mais moderna que visa melhorar a expressividade e a segurança em relação ao Java e Groovy.
- Tipagem Estática: Kotlin é uma linguagem de tipagem estática, o que pode resultar em menos erros de configuração, pois muitos erros são capturados em tempo de compilação.
- Concisa e Expressiva: Kotlin é conhecido por sua sintaxe concisa e expressiva, o que pode tornar a configuração do Gradle mais legível.
- Curva de Aprendizado: Se você estiver familiarizado com Kotlin ou gostaria de experimentar uma linguagem moderna, pode encontrar vantagens em usar essa abordagem.

# Tipo de projeto

- Projetos Maven:
- XML como Linguagem de Configuração: O Maven utiliza XML como a linguagem de configuração principal. A configuração é realizada através da edição de arquivos POM (Project Object Model).
- Convenção sobre Configuração: O Maven enfatiza a convenção sobre configuração, o que significa que ele fornece uma estrutura padrão para projetos, tornando a configuração inicial mais fácil.
- Gerenciamento de Dependências: O Maven tem um mecanismo robusto para gerenciar as dependências do projeto, usando o arquivo POM para definir essas dependências.
- Ampla Adoção: O Maven é amplamente adotado na comunidade Java e é usado por muitos projetos de código aberto e empresariais.

# Empacotamento

- Formato "JAR" (Java ARchive):
- Finalidade: O formato JAR é usado principalmente para empacotar bibliotecas Java (classes, recursos, etc.) em um único arquivo. Essas bibliotecas são geralmente reutilizáveis e podem ser compartilhadas entre diferentes projetos.
- Conteúdo: Um arquivo JAR contém classes compiladas, arquivos de recursos, metadados (geralmente no arquivo MANIFEST.MF) e outros recursos necessários para a biblioteca funcionar.
- Aplicação: Os arquivos JAR são frequentemente utilizados para distribuir e compartilhar bibliotecas, frameworks, utilitários e componentes reutilizáveis.
- Exemplo: Se você está desenvolvendo uma biblioteca de utilitários ou um conjunto de classes que serão usadas por outros projetos, um arquivo JAR é uma opção adequada.

# Empacotamento

- Formato "WAR" (Web ARchive):
- Finalidade: O formato WAR é projetado especificamente para empacotar aplicativos web Java. Ele contém todos os recursos necessários para executar um aplicativo web, incluindo classes, recursos, páginas JSP, arquivos HTML, arquivos de configuração e outros artefatos relacionados.
- Conteúdo: Além de conter as classes e recursos da aplicação, um arquivo WAR também inclui pastas específicas para estruturas web, como "WEB-INF" e "META-INF". O diretório "WEB-INF" contém configurações, classes específicas do aplicativo e arquivos JSP. O arquivo "WEB-INF/web.xml" é o arquivo de configuração da aplicação.
- Aplicação: O formato WAR é usado quando você está desenvolvendo aplicativos web Java, como sites, portais, serviços web, etc. É uma maneira de empacotar um aplicativo completo para ser implantado em um contêiner de servlet, como o Apache Tomcat.
- Exemplo: Se você está desenvolvendo um aplicativo web que usa tecnologias Java como Servlets, JSP e outros componentes web, um arquivo WAR é a escolha apropriada para empacotar e implantar o aplicativo.

# Exercícios

# Exercício 1

- Faça um form/serviço onde o usuário possa colocar duas datas (data inicio e data fim) e o servidor retorne;
  - A diferença de dias entre as datas;
  - A diferença de semana entre as datas;
  - A diferença de meses entre as datas;

# Exercício 2

- Faça um form/serviço onde o usuário possa entrar com números separados por ';' e o servidor retorne;
  - Os números em ordem crescente;
  - Os números em ordem decrescente;
  - Os números pares;



## Exercício 3 - Mimificador

Crie um form/serviço que dado um texto transforme para mimimi