**Introdução ao protocolo HTTP**

**Hypertext Transfer Protocol ou Protocolo de Transferência de Hipertexto:** é um protocolo de comunicação utilizado para sistemas de informação de hipermídia, distribuídos e colaborativo.

**Hipertexto** é o termo que remete a um texto ao qual se agregam outros conjuntos de informação na forma de blocos de textos, palavras, imagens ou sons, cujo acesso se dá através de referências específicas, no meio digital denominadas hiperlinks, ou simplesmente links.

O browser faz uma requisição para o servidor e o servidor envia uma resposta através do protocolo HTTP.

**HTTP é Stateless (Protocolo sem estado):** é um protocolo de comunicação que considera cada requisição como uma transação independente que não está relacionada a qualquer requisição anterior, de forma que a comunicação consista de pares de requisição e resposta independentes. Um protocolo sem estado não requer que o servidor retenha informação ou estado de sessão sobre cada parceiro de comunicação para a duração de várias requisições.

**Tipo de informações mandado por HTTP.**

**HTTP Request**

Verbo: GET, POST, PUT, DELTE, etc;

URL: Para onde será mandada a requisição;

Headers: Accpet, Accept-Language, etc. Qual a lingua que será aceita como resposta.

Parâmetros: Query, String, etc. Ex: atributos de um cadastro.

**HTTP Response**

Code: 200 OK, 404, 500;

Headers: Contect-type, etc; Ex: Página em HTML, PDF…

Resposta: [{“código”:1, “nome”: “São Paulo”, ...}] JSON

**MAVEN**

**Maven** é uma ferramenta que ajuda a gerenciar um projeto de software. A mais usada é o controle de dependência do projeto.

O Maven utiliza um arquivo XML (POM) para descrever o projeto de software sendo construído, suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios e plug-ins necessários.

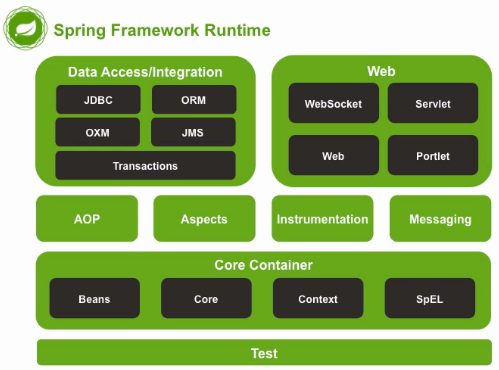
O Maven baixa bibliotecas Java e seus plug-ins dinamicamente de um ou mais repositórios, como o Maven 2 Central Repository, e armazena-os em uma área de cache local. O Maven é construído utilizando uma arquitetura baseada em plugin, que permite que ele faça uso de qualquer aplicação controlável através da entrada padrão. Teoricamente, isto permitiria qualquer um escrever plugins para fazer interface com ferramentas de construção (compiladores, ferramentas de teste de unidade, etc.) para qualquer outra linguagem.

**POM**

Um **Project Object Model**, ou **Modelo de Projeto de Objeto**, fornece todas as configurações para um único projeto. A configuração geral cobre o nome do projeto, seu proprietário e suas dependências de outros projetos. Também pode configurar fases individuais do processo de construção, que são implementados como plugins. Projetos maiores poderiam ser divididos em vários módulos, ou sub-projetos, cada um com seu próprio POM. Cada um pode então escrever um POM raiz através do qual ele pode compilar todos os módulos com um único comando.

**SPRING**

Plataforma Java.



Injeção de dependências fica na parte do Core Container.

**SPRING vs Java EE**

Vai da decisão de negócio. Tem suporte pago pra os dois. A curva de aprendizado é bem parecido. Ex: Java EE para injetar dependência é @Inject e no Spring é @Autowired. Facilidade no desenvolvimento. Ex: Spring Boot. Os dois tem uma boa documentação e suporte. Não é porque o Java EE é padrão que não posso usar outros frameworks. Atualização, no spring evolui mais rápido do que o Java EE porque ele não precisa de uma aprovação de um comitê de membro, assim que um tem uma ideia é criado um projeto é criado versões é lançado para testes e nesse mesmo momento são recebidos os feedbaks. Assim como é fácil criar também é fácil morrer esses projetos mais dinamicamente. Já no Java EE precisa de aprovações dos comitês e tudo mais. Java EE e Spring podem ser usados juntos. Ex: Spring Security com JPA

**EJB**

Enterprise JavaBeans (EJB) é um componente da plataforma JEE que roda em um container de um servidor de aplicação. Seu principal objetivo consiste em fornecer um desenvolvimento rápido e simplificado de aplicações Java, com base em componentes distribuídos, transacionais, seguros e portáveis.

**Padrão MVC**

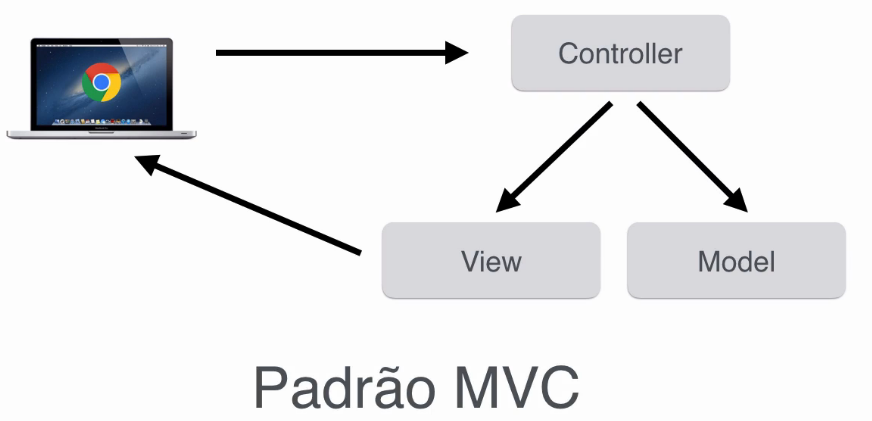
Ajuda a fazer o mapeamento. Ex:

Browser: É feito uma requisição na aplicação.

Controller: Recebe a requisição do browser. GET, POST, PUT, DELTE, etc; e delega para o Model.

Model: Faz as regras de negócio salva no banco e tudo mais. E devolve para o Controller e o controller indica qual View deve ser aberta.

View: A view é carregada e retornada de volta para o browser do usuário.



**Spring MVC vs JSF**

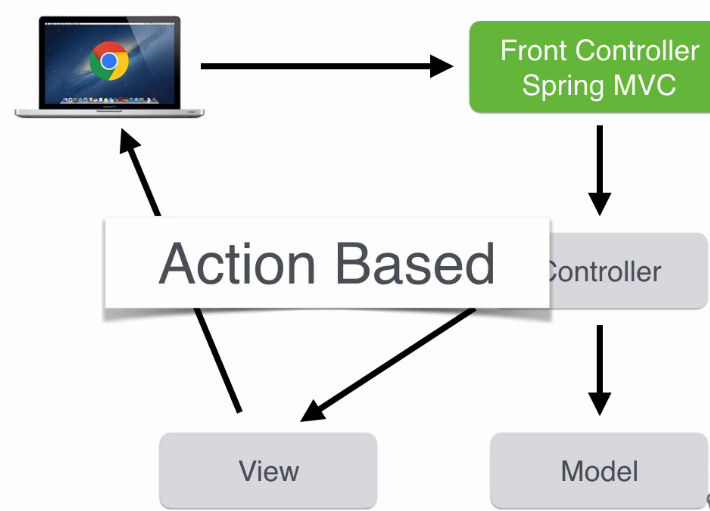
* Spring MVC

Browser: É feito uma requisição e vai para o Front Controller Spring MVC.

Front Controller Spring MVC (Framework): Manda para o Controller;

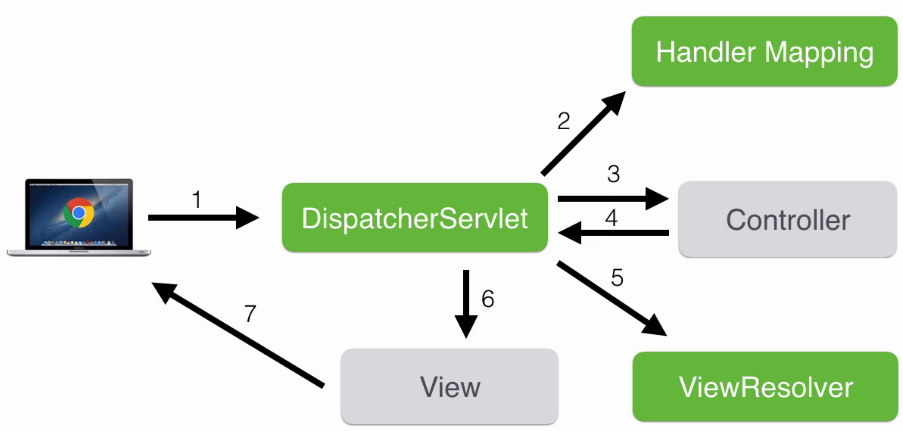
Controller (Nós que escrevemos.);

**Action Based:** O controle empurra/disponibiliza os dados para a view. Há uma flexibilidade grande para escrever a View porque o controller e a view são praticamente desacoplados;



**Detalhes do Spring MVC**

A requisição sai do browser e chega no Front Controller o Front Controller é um nome de um padrão que dentro do Spring MVC é chamado de DispatcherServlet. Depois ele precisa saber para qual controller ele precisa encaminhar e o Handler Mapping é responsável por isso. Logo depois o DispatcherServlet manda para o Controller. Depois de processado os dados a Controller devolve para o DispatcherServlet juntamente com o nome da View. Porém o DispatcherServlet também não sabe qual View ele deve mandar e o ViewResolver fica responsável por isso. Logo depois o DispatcherServlet consegue identificar a View e enviar os dados. A View vai desenhar todo o html e retornado para o browser.



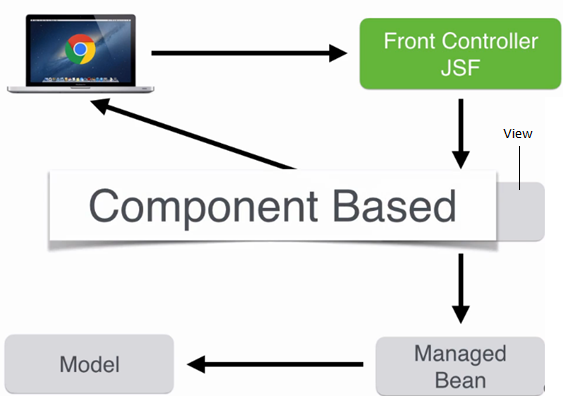
* JSF

Browser: É feito uma requisição e vai para o Front Controller JSF.

Front Controller JSF(Framework): O controller JSF que é o próprio frameworf manda direto para a view;

View: A view busca os dados do Managed Bean e o Managed Bean chama o model;

Model: O model tem toda a camada de negócio e acesso ao banco de dados; Feito isso ele devolve para o Managed Bean e disponibiliza para a View e a View retorna para o Browser;



Não é possível usar os dois frameworks juntos pois cada um tem seu próprio estilo de trabalhar.

**Component Based**: É possível criar componentes Java na View;

**Thymeleaf**

Template engine: Transforma objetos java em conteúdo em html.

@Autowired: Injetar uma dependência

@Qualifier(“nomeClass”): Usado para quando existir dois ou mais Beans que implementam o mesmo método. Nesse caso é preciso especificar de qual classe irá ser chamado o método