Desenvolvimento de Sistemas

Arquitetura *web* em camadas

Introdução

Aplicações *web* são, por fundamento, baseadas nas interações cliente-servidor, que ocorrem de forma recorrente (cliente solicita, servidor responde). Nesse modelo, é possível desenvolver aplicações *web* de diferentes formas, destacando as estáticas e as dinâmicas.

Um sistema *web* estático, em grande parte, é menos complexo, pois o conteúdo retornado é igual para todos os clientes que fizerem solicitações ao servidor, ou seja, retorna uma página fixa, sem informações personalizadas ou mutáveis ao usuário. A página usará HTML (marcação), CSS (estilização) e JavaScript (eventos, interações e animações). Veja a seguir um exemplo de interação com um sistema estático.

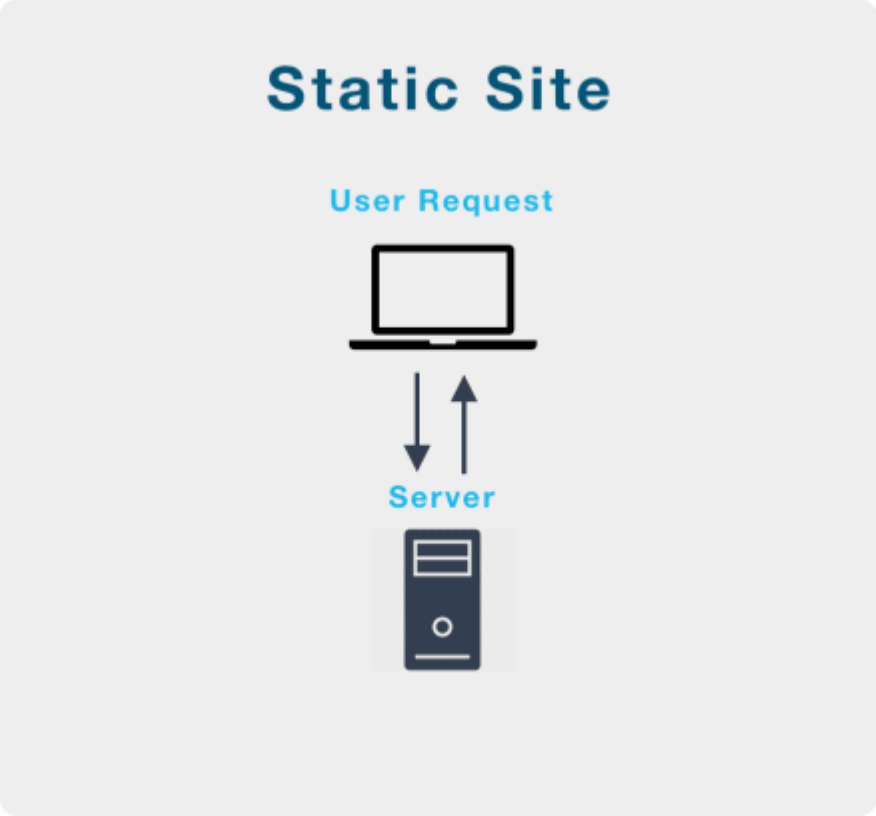

Figura 1 – Funcionamento de um *site* estático

Fonte: Nethmal (2020)

Em contrapartida, os chamados *sites* dinâmicos são aqueles em que parte do seu conteúdo é produzida dinamicamente para cada cliente que interage com o servidor. São sistemas que contam, geralmente, com o auxílio de uma base de dados. Além das **linguagens de** *front-end* (HTML, CSS, JS), precisa contar com uma linguagem de *back-end* capaz de tratar a requisição, processar as informações, eventualmente conectar a um banco de dados, montar e retornar a página como reposta ao usuário. Veja a seguir um exemplo de interação com um sistema dinâmico.

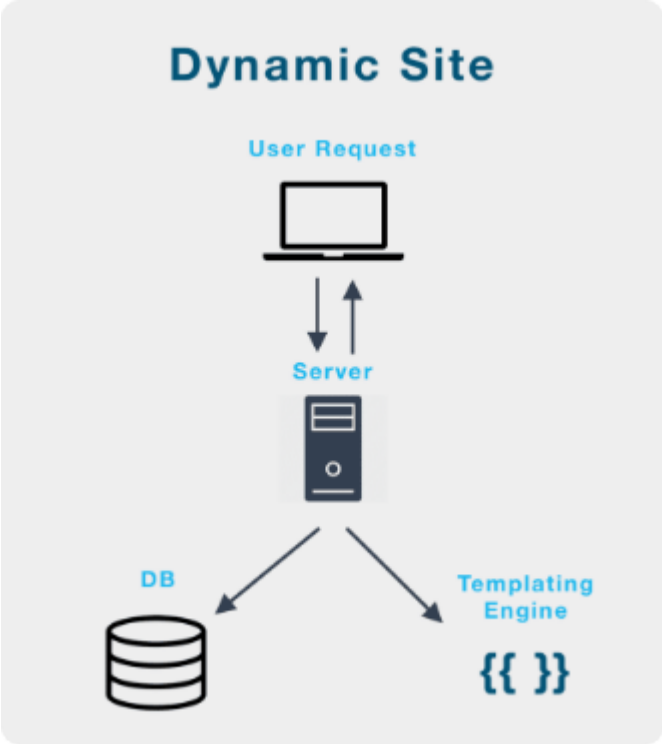

Figura 2 – Funcionamento de um *site* dinâmico

Fonte: Nethmal (2020)

Conceitos de camadas *front-end* e *back-end*

A camada *front-end* é tudo o que está sendo apresentado na tela do usuário, ou seja, o que pode ser visualizado por ele ao acessar determinada página. É toda a parte visível de um *site*, como formulários, imagens, menus, botões, listas e tabelas. Ainda pertencente à camada do *front-end*, pode-se ressaltar toda a interatividade presente em uma página, havendo *sites* desenvolvidos apenas com a utilização dessa camada visual. Os chamados *sites* estáticos são aqueles em que não há interação com o banco de dados. As principais linguagens para o desenvolvimento *front-end* são:

- HTML
- CSS

- JavaScript

De forma oposta à camada de *front-end*, a **camada** *back-end* restringe interações diretas com o usuário, no entanto é nessa camada que os dados são tratados e as interações com o banco de dados são feitas e retornadas ao usuário quando solicitadas. Em sistemas dinâmicos, essas consultas ocorrem de maneira constante, por exemplo: determinado usuário realiza o *login* em determinado sistema e os dados são tratados e comparados com os presentes na base de dados em questão. Uma vez verificado, o acesso à área restrita do sistema é liberado ou bloqueado. Como é possível perceber, é nessa camada também que são encontrados os mecanismos mais consistentes para a segurança de um sistema. Entre as linguagens mais utilizadas para a programação *back-end* estão:

- Java
- Python
- C#
- PHP

Protocolo HTTP e comunicação entre *front-end* e *back-end*

O protocolo HTTP é o responsável pela comunicação entre duas máquinas na *web*, uma máquina cliente e outra servidor (onde está armazenado determinado *site*). Ou seja, é por meio do protocolo HTTP que se comunica a camada de *front-end* com a camada de *back-end*.

Essa comunicação ocorre da seguinte maneira: o cliente faz uma requisição para o servidor (*request*), com os dados dessa requisição, o servidor então apresenta uma resposta ao cliente (*response*) de acordo com o solicitado. Esse conteúdo será visto a seguir. Também serão abordados os principais verbos utilizados no HTTP (GET e POST).

Figura 3 – Ilustração didática do protocolo HTTP

Fonte: Souza (2019)

Request

Em tradução direta, a palavra *request* significa requisição, ou seja, é um pedido que o cliente realiza ao servidor. Nesse pedido, encontram-se todos os dados necessários para atender à solicitação do cliente, podendo ser desde dados estáticos (já presentes na página) até dados de formulários (digitados pelo próprio cliente).

O *request* acontece a cada interação com uma página. Por exemplo: ao clicar em uma categoria de filmes de uma plataforma *streaming*, você está fazendo uma requisição ao servidor, que devolve os dados da página solicitada (*response*). Isso também acontece ao cadastrar um novo determinado recurso. Nessa situação, é enviada a requisição juntamente com os dados do formulário. Vale destacar que toda informação em HTTP é transportada como texto.

O protocolo ainda define verbos ou métodos, que são modos de interpretar essas requisições. Os mais comuns são GET e POST.

- GET – Neste verbo, os parâmetros são enviados por meio do cabeçalho da requisição, ficando visíveis os dados transmitidos pela URL.
- POST – Neste verbo, os dados são enviados no corpo da requisição, sendo assim, esses parâmetros não estão visíveis para o usuário explicitamente em primeiro momento, porém podem ser facilmente acessados utilizando recursos básicos.

Além desses verbos, ainda existem outros, como PUT, DELETE e PATCH, normalmente utilizados com serviços *web*.

Response

Como visto anteriormente, o *request* é uma requisição feita ao servidor com dados que o cliente deseja receber do servidor. A partir do processamento em servidor, é enviada a resposta – ou *response* – contendo os dados solicitados. Por exemplo, ao solicitar uma página como “www.ead.senac.br”, um *request* é enviado ao servidor, que fará processamentos, e um *response* é enviado de volta pelo servidor com os dados da página (o próprio código HTML dela).

Junto com os dados de resposta (o corpo da resposta), um código de *status* é enviado. Trata-se de um número entre 100 e 599, em que cada centena tem uma categoria para os *status*, sendo elas:

- 100 – 199 – Resposta de informação
- 200 – 299 – Resposta de sucesso
- 300 – 399 – Redirecionamentos
- 400 – 499 – Erros no lado do cliente
- 500 – 599 – Erros no lado do servidor

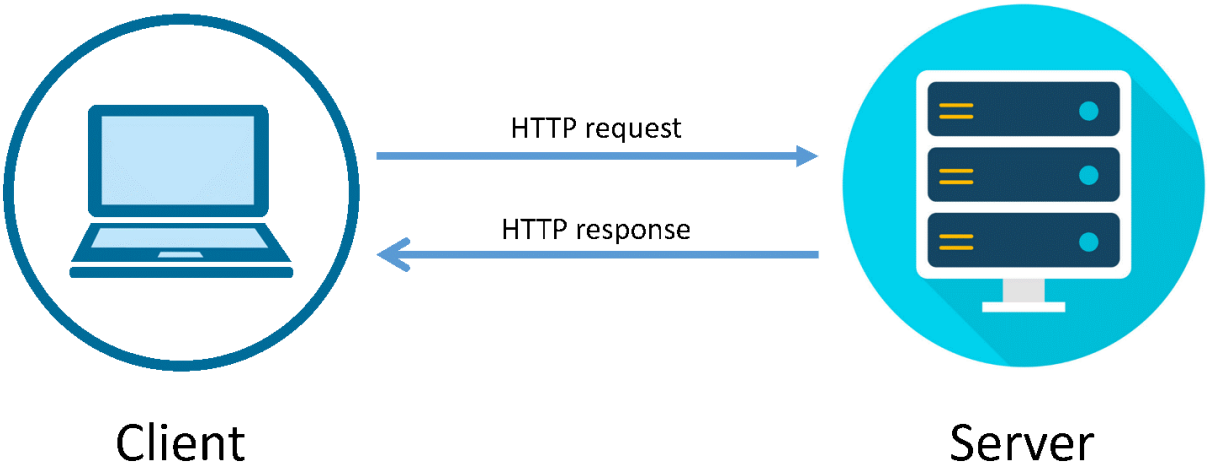

Figura 4 – Modelo cliente-servidor, com requisição e resposta

Fonte: Wan (2022)

Visto essas situações, é possível definir o protocolo HTTP como um protocolo sem estado (*stateless*), e como não guarda estados, cada par de requisição/resposta é único e independente. Sendo assim, o servidor não armazena informações referentes

às diversas requisições que recebe, o que simplifica a estrutura necessária de servidor, uma vez que não há necessidade de armazenamento adicional para lidar com as interações em andamento.

Request

Em tradução direta, a palavra *request* significa requisição, ou seja, é um pedido que o cliente realiza ao servidor. Nesse pedido, encontram-se todos os dados necessários para atender à solicitação do cliente, podendo ser desde dados estáticos (já presentes na página) até dados de formulários (digitados pelo próprio cliente).

O *request* acontece a cada interação com uma página. Por exemplo: ao clicar em uma categoria de filmes de uma plataforma *streaming*, você está fazendo uma requisição ao servidor, que devolve os dados da página solicitada (*response*). Isso também acontece ao cadastrar um novo determinado recurso. Nessa situação, é enviada a requisição juntamente com os dados do formulário. Vale destacar que toda informação em HTTP é transportada como texto.

O protocolo ainda define verbos ou métodos, que são modos de interpretar essas requisições. Os mais comuns são GET e POST.

- GET – Neste verbo, os parâmetros são enviados por meio do cabeçalho da requisição, ficando visíveis os dados transmitidos pela URL.
- POST – Neste verbo, os dados são enviados no corpo da requisição, sendo assim, esses parâmetros não estão visíveis para o usuário explicitamente em primeiro momento, porém podem ser facilmente acessados utilizando recursos básicos.

Além desses verbos, ainda existem outros, como PUT, DELETE e PATCH, normalmente utilizados com serviços *web*.

Response

Como visto anteriormente, o *request* é uma requisição feita ao servidor com dados que o cliente deseja receber do servidor. A partir do processamento em servidor, é enviada a resposta – ou *response* – contendo os dados solicitados. Por exemplo, ao solicitar uma

página como “www.ead.senac.br”, um *request* é enviado ao servidor, que fará processamentos, e um *response* é enviado de volta pelo servidor com os dados da página (o próprio código HTML dela).

Junto com os dados de resposta (o corpo da resposta), um código de *status* é enviado. Trata-se de um número entre 100 e 599, em que cada centena tem uma categoria para os *status*, sendo elas:

- 100 – 199 – Resposta de informação
- 200 – 299 – Resposta de sucesso
- 300 – 399 – Redirecionamentos
- 400 – 499 – Erros no lado do cliente
- 500 – 599 – Erros no lado do servidor

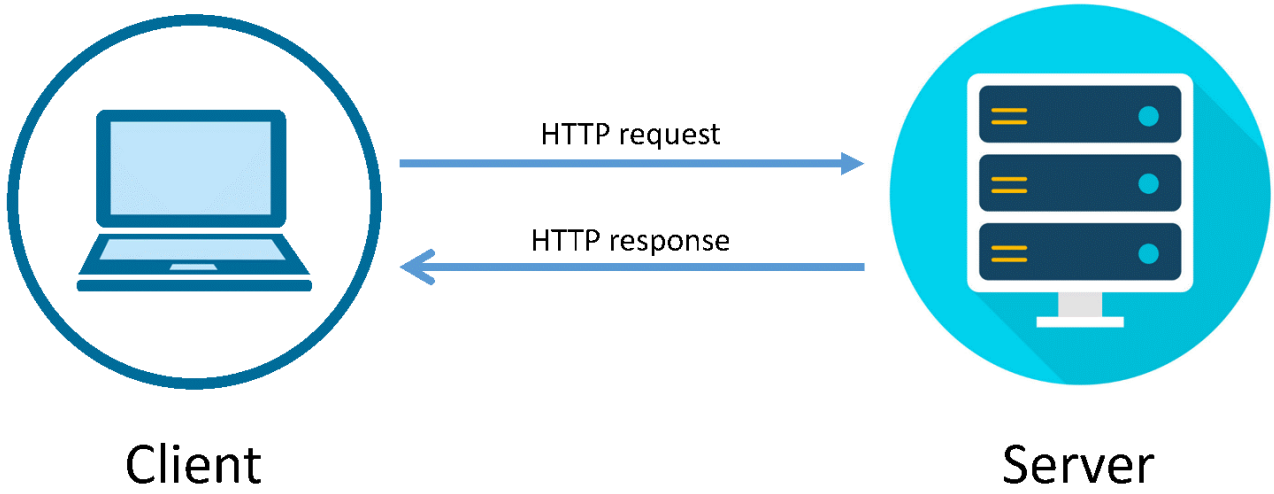

Figura 4 – Modelo cliente-servidor, com requisição e resposta

Fonte: Wan (2022)

Visto essas situações, é possível definir o protocolo HTTP como um protocolo sem estado (*stateless*), e como não guarda estados, cada par de requisição/resposta é único e independente. Sendo assim, o servidor não armazena informações referentes às diversas requisições que recebe, o que simplifica a estrutura necessária de servidor, uma vez que não há necessidade de armazenamento adicional para lidar com as interações em andamento.

Web Server e URL

Um *web server* – ou servidor *web* – é um *software* capaz de processar a requisição de um cliente e enviar de volta a resposta a ele. O Apache é um dos servidores *web* mais usados no mercado. Outro também importante é o Tomcat, servidor *web* que se integra com Java.

O servidor *web* roda em uma máquina física ou virtual, conectada à internet e com acessos externos liberados por meio de portas. Em suma, uma porta é uma numeração específica que acessa um *software* específico em um servidor. No caso do Apache *web Server*, a porta utilizada é a de número 80, padrão para *web*. Já no Tomcat, se usa por padrão 8080.

Quando o usuário requisita algo ao servidor através de URL, o cliente (um navegador como Chrome ou Firefox, por exemplo) monta uma requisição e envia ao servidor, que aciona o *web server*, que processa a requisição montando uma resposta, muitas vezes, com base em arquivos HTML mapeados pelo *software*. Essa resposta é enviada ao cliente, que a decodifica e a apresenta ao usuário.

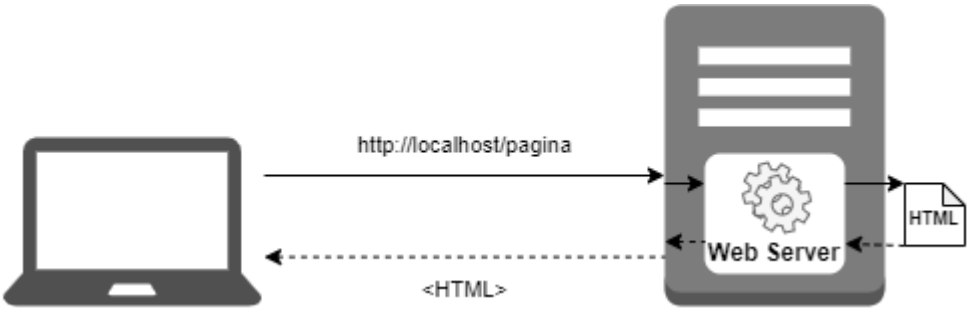

Figura 5 – Requisição de uma página estática

Fonte: Senac EAD (2023)

A URL (sigla para *Uniform Resource Locator* ou Localizador Uniforme de Recursos) é o endereço por meio do qual se alcança um servidor *web*. Observe a URL http://localhost:8080/meu-site?param=abc a seguir.

http:// define que está sendo usado o protocolo HTTP.

localhost é o chamado “domínio” da internet. Cada máquina é endereçada por um IP, uma numeração separada em quatro seções. Por exemplo, 142.250.218.78 ou 187.51.127.188. Para facilitar a localização, há o DNS (Sistema de Nomes de Domínios) ou simplesmente Domínios, que traduz esses IPs para nomes mais inteligíveis e memoráveis.

Assim, 187.51.127.188 pode ser referenciado por “google”, em “google.com”, e 187.51.127.188 por “ead.senac”, em “ead.senac.br”. Localhost é um domínio universal relativo ao IP 127.0.0.1, que é o IP usado para uma máquina local – ou seja, seu próprio computador.

:8080 é a porta que deverá ser usada para acessar o servidor *web*. Quando ela não é informada, assume-se a porta 80.

/meu-site é o recurso requisitado do servidor, que geralmente resultará em uma página *web*.

?param=abc são parâmetros opcionais enviados na requisição. Esses parâmetros podem ser processados pelo servidor como dados de entrada do usuário.

Web servers como o Apache sozinhos são ótimos para fornecer páginas estáticas. No entanto, para um sistema *web* dinâmico, é necessário integrar o servidor *web* a linguagens como PHP, Python, Ruby, C# ou Java. No caso do Apache, o mais comum é usá-lo com PHP. Para Java, há opções como o Tomcat e o Glassfish.

Java para *web*

Java conta com uma série de especificações para diferentes finalidades de *softwares*. Uma das mais notáveis é o **Java EE** (Java Enterprise Edition), que especifica e fornece bibliotecas para o desenvolvimento de aplicações para *web*. A partir das bibliotecas de Java EE – ou de *frameworks* que se utilizem delas, como será visto adiante – é possível programar *back-end* em uma aplicação *web* e integrar a camada de *front-end*, montando páginas com informações processadas.

A plataforma Java EE é extensa e cobre funcionalidades de persistência, XML, trocas de mensagens, autenticação e autorização. Em relação ao desenvolvimento *web*, destacam-se os seguintes conceitos:

Spring Framework

O Spring é um extenso *framework* Java, que visa facilitar o desenvolvimento com base em técnicas de injeção de dependências e inversão de controle. O Spring é hoje o *framework* Java mais popular do mundo. Esse *framework* tem como objetivo facilitar muitos

aspectos da programação Java, tornando-a mais fácil, rápida e segura, contando com aproximadamente 20 módulos para diferentes propósitos.

Figura 6 – Logo do Spring Framework

Fonte: Spring (c2023)

Entre muitos aspectos, o Spring ainda conta com um módulo de banco de dados de forma integrada, o Spring Data. Vale ressaltar que esse *framework* ainda conta com um módulo MVC (*Model*, *View* e *Controller*), que facilita o desenvolvimento de aplicação *web* utilizando esse padrão.

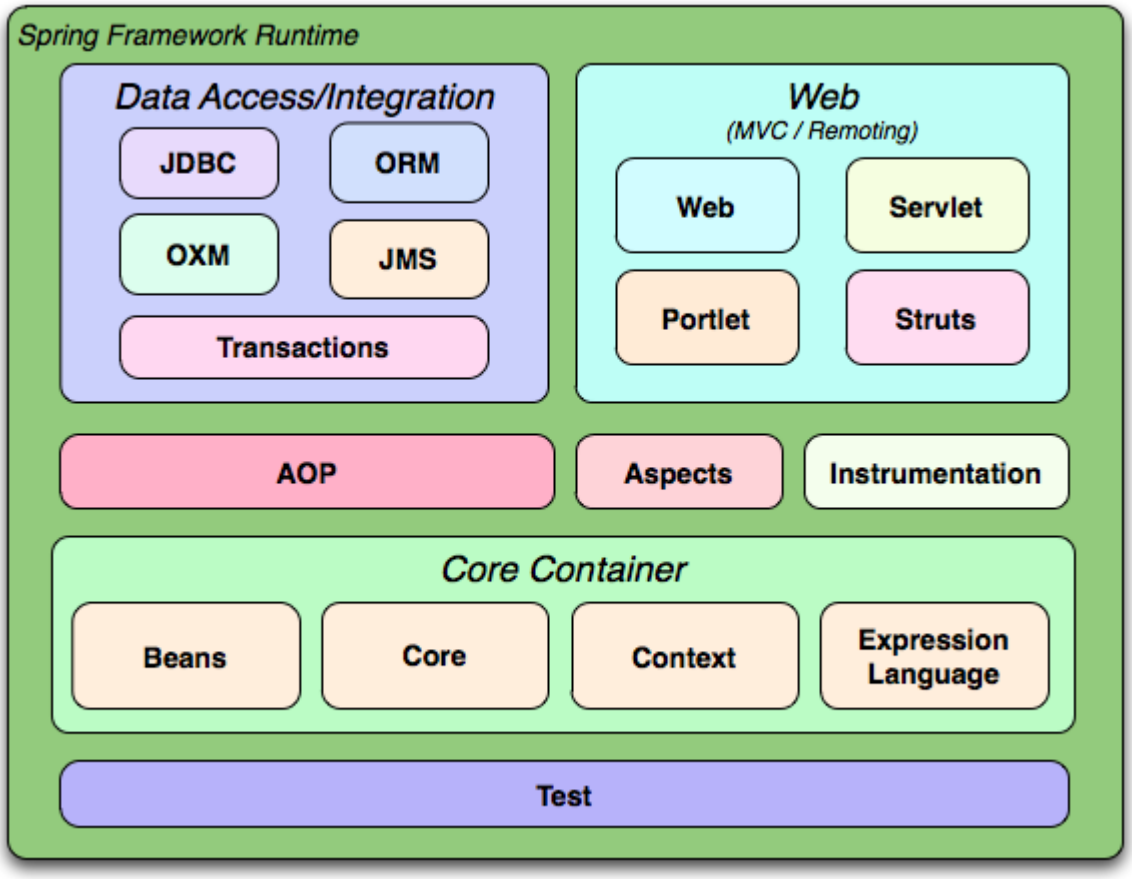

Figura 7 – Arquitetura geral do Spring Framework

Fonte: Spring (c2023)

Por fim, para facilitar a utilização do Spring, há a extensão Spring Boot, que visa facilitar e acelerar o processo de manutenção e configuração do projeto, sendo aconselhável em grande parte da utilização do Spring.

Mecanismos do Spring Framework

Para entender como funciona o Spring Framework, é necessário se familiarizar com alguns termos e compreender o que está acontecendo por baixo da execução do Spring *Web*, o módulo responsável pelo desenvolvimento de sistemas *web*.

Quando o servidor *web* é iniciado, o Spring Boot procura componentes e configurações do Spring no seu projeto e cria o contexto do aplicativo Spring. Em seguida, o Spring Boot cria e registra vários beans (objetos Java para fins específicos) no contexto do aplicativo, incluindo controladores, serviços, repositórios etc. Esses objetos serão gerenciados pelo contêiner do Spring e injetados em outras partes do aplicativo quando necessário. Nesse processo, três conceitos são importantes para Spring: a Inversão de Controle, a Injeção de Dependência e os Java Beans.

Inversão de Controle (IoC – Inversion of Control) é um princípio de design no qual um objeto em vez de criar suas próprias dependências (ou seja, outros objetos de que ele precisa para funcionar), ele delega essa tarefa a um "contêiner" especializado em gerenciar essas dependências. Por exemplo, se um objeto A precisa de outro objeto B para funcionar, em vez de A criar uma nova instância de B, o contêiner IoC cria a instância de B e a fornece a A.

Imagine que você precisa construir uma casa e, para isso, precisa de um encanador, um eletricista e um pedreiro. Em vez de você mesmo procurar e contratar cada um desses profissionais, você contrata uma empresa de construção que cuida de todos os aspectos da construção, incluindo a contratação desses profissionais. Isso é semelhante ao IoC, em que você delega a tarefa de gerenciar as dependências do seu programa a um contêiner especializado em gerenciar essas dependências.

Injeção de Dependência (DI – Dependency Injection) é a técnica utilizada pelo Spring Framework para implementar a Inversão de Controle. Ela define como as classes serão criadas e onde serão usadas em um programa. Por exemplo, se uma classe A precisa de uma instância da classe B para executar seu método, a classe A pode indicar que precisa de uma instância de B por meio de um ponto de injeção, como um construtor. Quando a classe A for executada, o contêiner do Spring Framework criará uma instância da classe B e a injetará na classe A, permitindo que o método b.metodoB() seja executado.

Podemos pensar na Injeção de Dependência como um serviço de entrega que traz exatamente o que você precisa, quando você precisa. Se você precisa de leite, pão e manteiga para o café da manhã, você pode simplesmente indicar isso para o serviço de entrega, e eles trarão esses itens exatamente no momento em que você precisa deles. Da mesma forma, a Injeção de Dependência traz exatamente as instâncias de classes que um objeto precisa para funcionar, quando ele precisa delas.

“Beans” são objetos que são criados, gerenciados e mantidos pelo contêiner do Spring. Eles representam as diferentes instâncias de classes que são definidas pelo programador e que são gerenciadas pelo contêiner do Spring. Por meio da configuração de Beans, os desenvolvedores podem definir as dependências e as características de cada objeto, e o contêiner do Spring se encarrega de gerenciar todo o ciclo de vida desses objetos.

Os Beans estão intimamente relacionados com a Inversão de Controle e a Injeção de Dependência, pois eles são criados e gerenciados pelo contêiner do Spring, que é responsável por injetar as dependências necessárias em cada objeto, conforme definido na configuração do programa. Isso permite que os desenvolvedores se concentrem em criar a lógica de negócios de suas classes, sem se preocupar com a criação e o gerenciamento de objetos manualmente.

Além disso, o contêiner do Spring também oferece recursos para gerenciar a configuração dos Beans, como a criação e a destruição de objetos, a configuração de propriedades e dependências, a gestão de escopos, entre outros recursos.

Em resumo, os Beans do Spring Framework são objetos que são criados e gerenciados pelo contêiner do Spring, permitindo que os desenvolvedores se concentrem na lógica de negócios e deixem a criação e o gerenciamento de objetos por conta do contêiner. Eles são uma parte importante da IoC e da DI, que fazem parte da filosofia do Spring de oferecer um ambiente de desenvolvimento mais modular, escalável e eficiente.

Em resumo, o Spring implementa a Inversão de Controle e a Injeção de Dependência para tornar o desenvolvimento de aplicativos mais fácil e modular. Ele fornece um ambiente gerenciado que centraliza a configuração de objetos e suas dependências, permitindo que os desenvolvedores se concentrem mais na lógica de negócios do que na infraestrutura do programa. Essa configuração toda seria feita manualmente com as implementações clássicas de Servlets e JSP de Java.

Com a IoC e a DI, os desenvolvedores podem definir as dependências de seus objetos de forma clara e simples, sem se preocupar em criá-las manualmente. O resultado é um código mais limpo, modular e escalável, que pode ser facilmente mantido e expandido.

Encerramento

Retomamos aqui conceitos fundamentais de aplicações na *web*. Ao desenvolver um sistema *web*, é importante ter em mente, acima de tudo, o mecanismo de *Request* e *Response*. Em conteúdos posteriores nesta unidade, será explorado mais o desenvolvimento *web* com Java e aplicado, para isso, o Spring Framework.