Padrão MVC

1 – Introdução

Na fase de Projeto começamos a nos preocupar com a arquitetura da aplicação. Damos realmente valor a tecnologia, diferente da fase de análise onde ainda estamos esboçando o problema a ser resolvido. Definimos a plataforma e como os componentes do sistema se organizarão. Evidentemente que os requisitos ainda são importantes, pois, por exemplo, um Sistema Web ou então uma aplicação de tempo real deverá influenciar na arquitetura.

Mesmo não possuindo uma definição consensual, muitos autores definem a arquitetura de software de um sistema computacional como as suas estruturas, que são compostas de elementos de software, de propriedades externamente visíveis de seus componentes e do relacionamento entre eles. Ou seja, a arquitetura define os elementos de software e como eles interagem entre si.

Para realizar a arquitetura de uma aplicação não basta estar num dia inspirado ou acordar com bastante vontade de realizar uma arquitetura, muito pelo contrário, precisamos de bastante experiência em diferentes organizações, diferentes tipos de projetos, conhecimentos de diferentes arquiteturas etc. A experiência prática ainda é a melhor solução, pois o trabalho em equipe também é uma forma excelente de definir uma arquitetura. Muitas vezes, alguns programadores possuem outras experiências ou conhecimentos e a troca dessa experiência é sempre válida, mesmo quando temos um arquiteto bastante experiente na equipe.

A arquitetura de um sistema tem diversos elementos como: elementos utilitários, de interação, elementos que fazem parte do domínio do problema, elementos de conexão, de persistência etc. Dessa forma, na arquitetura sempre definimos os seus elementos que serão utilizados no software e como eles se conectam. Uma arquitetura complexa exige mais tempo para desenvolvimento, porém, através de geração automática de aplicações torna-se mais produtivo. Por isso algumas equipes definem um framework para uma determinada aplicação e assim podemos utilizar muitas coisas pré-prontas que facilitam o desenvolvimento.

2 – MVC (Model View Controller)

O MVC é utilizado em muitos projetos devido a arquitetura que possui, o que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.

A utilização do padrão MVC traz como benefício o isolamento das regras de negócios da lógica de apresentação, que é a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem a necessidade de alterar as regras de negócios, proporcionando muito mais flexibilidade e oportunidades de reuso das classes.

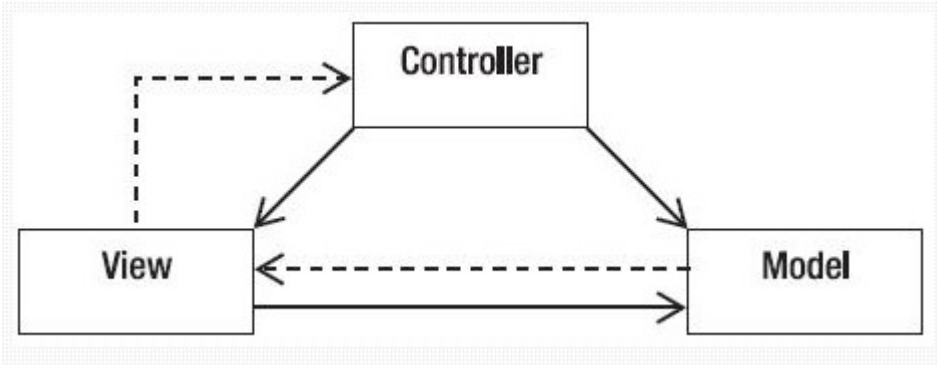
Uma das características de um padrão de projeto é poder aplicá-lo em sistemas distintos.O padrão MVC pode ser utilizado em vários tipos de projetos como, por exemplo, desktop, web e mobile.

A comunicação entre interfaces e regras de negócios é definida através de um controlador, que separa as camadas. Quando um evento é executado na interface gráfica, como um clique em um botão, a interface se comunicará com o controlador, que por sua vez se comunica com as regras de negócios.

Imagine uma aplicação financeira que realiza cálculos de diversos tipos, como os de juros. Você pode inserir valores para os cálculos e também escolher que tipo de cálculo será realizado. Isto tudo é feito pela interface gráfica, que para o modelo MVC é conhecida como View. No entanto, o sistema precisa saber que você está requisitando um cálculo, e para isso, terá um botão no sistema que quando clicado gera um evento.

Este evento pode ser uma requisição para um tipo de cálculo específico como o de juros simples ou juros compostos. Fazem parte da requisição os valores digitados no formulário e a seleção do tipo de cálculo que o usuário quer executar sobre o valor informado. O evento do botão é como um pedido a um intermediador (Controller) que prepara as informações para então enviá-las para o cálculo. O controlador é o único no sistema que conhece o responsável pela execução do cálculo, neste caso, a camada que contém as regras de negócios. Esta operação matemática será realizada pelo Model assim que ele receber um pedido do Controller.

O Model realiza a operação matemática e retorna o valor calculado para o Controller, que também é o único que possui conhecimento da existência da camada de visualização. Tendo o valor em mãos, o intermediador o repassa para a interface gráfica que exibirá para o usuário. Caso esta operação deva ser registrada em uma base de dados, o Model se encarrega também desta tarefa.



Explicando cada um dos objetos do padrão MVC tem-se:

1. Primeiramente o controlador (Controller), que interpreta as entradas do mouse ou do teclado enviadas pelo usuário e mapeia essas ações do usuário em comandos que são enviados para o modelo (Model) e/ou para a janela de visualização (View) para efetuar a alteração apropriada;
2. Por sua vez, o modelo (Model) gerencia um ou mais elementos de dados, responde a perguntas sobre o seu estado e responde a instruções para mudar de estado. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema a ser resolvido;
3. Por fim, a visão (View) gerencia a área retangular do display e é responsável por apresentar as informações para o usuário através de uma combinação de gráficos e textos. A visão não sabe nada sobre o que a aplicação está atualmente fazendo, pois tudo que ela realmente faz é receber instruções do controle e informações do modelo e então exibi-las. A visão também se comunica de volta com o modelo e com o controlador para reportar o seu estado.

Portanto, a principal ideia do padrão arquitetural MVC é a separação dos conceitos - e do código.O MVC é como a clássica programação orientada a objetos, ou seja, criar objetos que escondem as suas informações e como elas são manipuladas e então apresentadas em uma interface simples para o mundo. Entre as diversas vantagens do padrão MVC estão a possibilidade de reescrita da GUI ou do Controller sem alterar o modelo, reutilização da GUI para diferentes aplicações com pouco esforço, facilidade na manutenção e adição de recursos, reaproveitamento de código, facilidade na manutenção do código sempre limpo etc.

3 – Spring Boot

O Spring Boot é um framework Java open source que tem como objetivo facilitar esse processo em aplicações Java. Consequentemente, ele traz mais agilidade para o processo de desenvolvimento, uma vez que devs conseguem reduzir o tempo gasto com as configurações iniciais.

Com o Spring Boot conseguimos abstrair e facilitar a configuração de, por exemplo:

•Servidores;

•Gerenciamento de dependências;

•Configurações de bibliotecas;

•Métricas & health checks;

Para realizar todo esse processo o Spring Boot utiliza um conceito chamado convenção sobre configuração. É uma ferramenta que decide para você a melhor forma de se fazer algo. É o que chamamos de ferramenta opinativa, ela toma as decisões no nosso lugar baseado em convenções, aplicando configurações padrões e facilitando o trabalho.

No entanto ela não é inflexível e ainda permite uma configuração diferente da default caso o usuário assim deseje. Por exemplo, você pode alterar para que ele utilize o Jetty como servidor ao invés do Tomcat que é a configuração padrão.

Uma das maiores vantagens que o Spring Boot trouxe ao desenvolvimento é que toda essa configuração não necessita mais ser realizada pelos temidos XMLs, embora ele ainda suporte esse tipo de configuração. A maior parte da configuração pode ser feita de forma programática via anotações. O Spring Boot é composto por vários módulos que ajudam nesse processo. Alguns deles são:

Spring Boot

É o módulo principal que ajuda na configuração e integração dos outros módulos.

Spring Boot Starters

Os starters são dependências que agrupam outras dependências com um propósito em comum. Dessa forma, somente uma configuração é realizada no seu gerenciador de dependências.

Por exemplo, o spring-boot-starter-amqp, é um starter que permite a construção de soluções de mensageria baseadas em AMQP e RabbitMQ.Ao realizar a configuração no meu gerenciador de dependência se define somente o starter:

```
<dependencies>
   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-amqp</artifactId>
   </dependency>
</dependencies>
```

E internamente ele encapsula as dependências necessárias para utilização das features. No caso do spring-boot-starter-amqp, ele encapsula três dependências:

```
plugins {
 id "org.springframework.boot.starter"
}

description = "Starter for using Spring AMQP and Rabbit MQ"

dependencies {
 api(project(":spring-boot-project:spring-boot-starters:spring-boot-
starter"))
 api("org.springframework:spring-messaging")
 api("org.springframework.amqp:spring-rabbit")
}
```

Alguns exemplos de starters disponíveis:

•Spring Boot Starter Web: Auxilia na construção de aplicações web trazendo já disponíveis para uso Spring MVC, Rest e o Tomcat como servidor.

•Spring Boot Starter Test: Contém a maioria das dependências necessárias para realizar testes da sua aplicação: Junit, AssertJ, Hamcrest, Mockito, entre outros

•Spring Boot Starter Data JPA: Facilita a construção da nossa camada de persistência, ajudando na abstração do nosso banco de dados provendo uma série de facilidades para criação de repositories, escrita de queries, entre outros.

Spring Boot Autoconfigure

Como dito anteriormente o Spring Boot trabalha de forma opinativa, tomando decisões para você. Essas decisões padrões são baseadas através do conteúdo do seu classpath.

O Autoconfigure é responsável por ler este conteúdo e realizar as configurações necessárias para que a aplicação funcione. É ele quem gerencia todo o processo de configuração da aplicação.

Spring Boot Actuator

O Spring Boot Actuator é uma ferramenta que permite monitorar e gerenciar as aplicações implantadas. Dentre os recursos disponibilizados temos:

- Métricas: Obtém e disponibiliza diversos dados da nossa aplicação, como por exemplo, espaço em disco, memória, tempo de resposta etc.
- Logging: Facilita o acesso ao arquivo de log da aplicação por meio de um endpoint específico.
- HeathlChecks: Disponibiliza endpoints de health checks.
- Informações da Aplicação: Permite a disponibilização de informações da aplicação. Por exemplo, versão, informações do git etc.

Spring Boot Test

O Spring Boot Test contém funcionalidades úteis e anotações que facilitam e ajudam a testar sua aplicação.

Spring Boot Devtools

Spring Boot Devtools é um conjunto de funcionalidades que ajuda o trabalho de qualquer dev. Como, por exemplo, restart automático da aplicação quando ocorre alguma mudança no código.

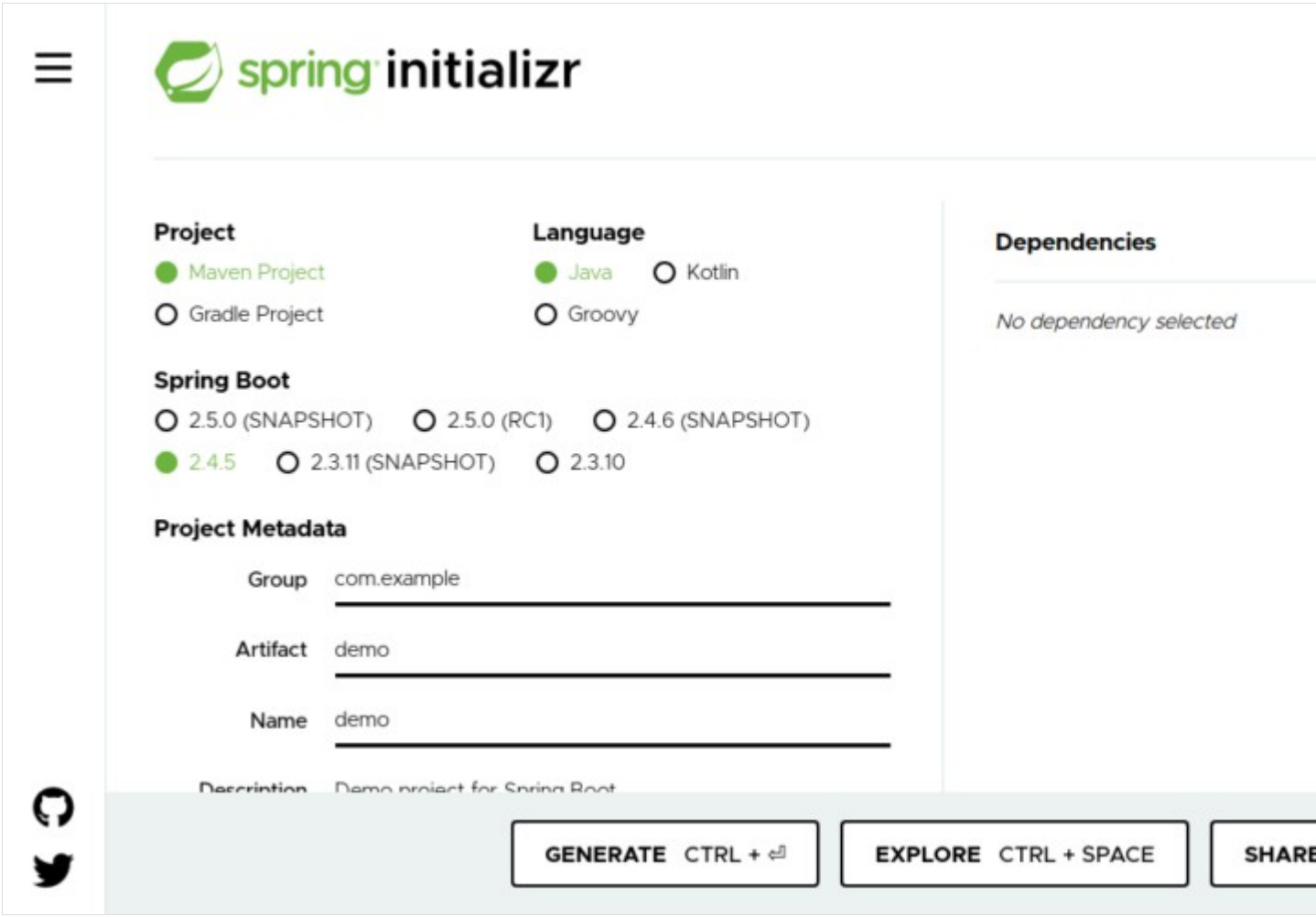
Spring Tool Suite

O Spring nos fornece uma IDE totalmente customizada para o desenvolvimento de aplicações do ecossistema spring: o Spring Tool Suite (STS).

O STS é uma IDE baseada em Eclipse que já vem com algumas funcionalidades facilitadoras para projetos Spring.

Spring Initializr

E para facilitar a criação de aplicações utilizando outras IDEs a Spring disponibilizou o Spring Initializr. Ele é uma UI que permite a criação de projetos Sprint Boot de forma facilitada.



Através dele definimos nome do projeto, pacotes, dependências (starters do spring e outros projetos), linguagem (Java, Groovy ou Kotlin).Uma vez definido é só clicar no botão Generate e o projeto será criado, gerando um zip pronto para ser importado na IDE de sua preferência.