**MVC**

**O MVC é utilizado em muitos projetos devido à 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 trás como benefício isolar as regras de negócios da lógica de apresentação, a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem que haja a necessidade da alteração das regras de negócios, proporcionando assim 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, e é a existência deste controlador que torna possível a separação entre as camadas. Quando um evento é executado na interface gráfica, como um clique em um botão, a interface irá se comunicar com o controlador que por sua vez se comunica com as regras de negócios.

Imagine uma aplicação financeira que realiza cálculos de diversos tipos, entre eles os de juros. Você pode inserir valores para os cálculos e também escolher que tipo de cálculo será realizado. Isto tudo você faz 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, você terá um botão no sistema que quando clicado gera um evento.

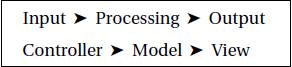
Este evento pode ser uma requisição para um tipo de cálculo específico como o de juros simples ou juros compostos. Fazem parte da requisição neste caso os valores digitados no formulário, como também a seleção do tipo de cálculo que o usuário quer executar sobre o valor informado. O evento do botão é como um pedido a um intermediador que prepara as informações para então enviá-las para o cálculo. Este intermediador nós chamamos de Controller. 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.

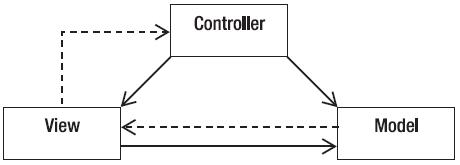
Confira nosso [**Artigo Completo sobre Padrão MVC**](https://www.devmedia.com.br/padrao-mvc-java-magazine/21995)

**O padrão arquitetural Model-View-Controller (MVC) é uma forma de quebrar uma aplicação, ou até mesmo um pedaço da interface de uma aplicação, em três partes: o modelo, a visão e o controlador**.

O MVC inicialmente foi desenvolvido no intuito de mapear o método tradicional de entrada, processamento, e saída que os diversos programas baseados em GUI utilizavam. No padrão MVC, teríamos então o mapeamento de cada uma dessas três partes para o padrão MVC conforme ilustra a imagem abaixo:

  
**Figura 1**: Mapeamento das três partes de uma aplicação para o MVC.

A figura abaixo demonstra que a entrada do usuário, a modelagem do mundo externo e o feedback visual para o usuário são separados e gerenciados pelos objetos Modelo (Model), Visão (View) e Controlador (Controller).

  
**Figura 2**: Objetos utilizados no MVC e suas interações.

Explicando cada um dos objetos do **padrão MVC** tem-se primeiramente o controlador (Controller) que interpreta as entradas do mouse ou do teclado enviado 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. 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 que está se tentando resolver. 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, tudo que ela realmente faz é receber instruções do controle e informações do modelo e então exibir elas. A visão também se comunica de volta com o modelo e com o controlador para reportar o seu estado.

Tão importante quanto explicar cada um dos **objetos do padrão arquitetural MVC** é explicar como é o seu fluxo tipicamente. Primeiramente o usuário interage com a interface (por exemplo, pressionando um botão) e o controlador gerenciar esse evento de entrada da interface do usuário. A interface do usuário é exibida pela visão (view), mas controlada pelo controlador. O controlador não tem nenhum conhecimento direto da View, ele apenas envia mensagens quando ela precisa de algo na tela atualizado. O controlador acessa o modelo, possivelmente atualizando ela de forma apropriada para as ações do usuário (por exemplo, o controlador solicita ao modelo que o carrinho de compras seja atualizado pelo modelo, pois o usuário incluiu um novo item). Isto normalmente causa uma alteração no estado do modelo tanto quanto nas informações. Por fim, a visão usa o modelo para gerar uma interface com o usuário apropriada. A visão recebe as informações do modelo. O modelo não tem conhecimento direto da visão. Ele apenas responde a requisições por informações de quem quer que seja e requisita por transformações nas informações feitas pelo controlador. Após isso, o controlador, como um gerenciador da interface do usuário, aguarda por mais interações do usuário, onde inicia novamente todo o ciclo.

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 apresentar apenas uma simples interface para o mundo. Entre as diversas **vantagens do padrão MVC** estão a possibilidade de reescrita da GUI ou do Controller sem alterar o nosso modelo, reutilização da GUI para diferentes aplicações com pouco esforço, facilidade na manutenção e adição de recursos, reaproveitamento de código, facilidade de manter o código sempre limpo, etc.

Rest

O que é REST?

REST é acrônimo de *Representational State Transfer*, e tem como objetivo primário a definição de características fundamentais para a construção de aplicações Web seguindo boas práticas.

A Web como a conhecemos hoje, funciona seguindo práticas REST, e para entendermos melhor, vamos considerar o exemplo abaixo:

1. Você entra com um endereço em seu navegador (Chrome, Firefox, Edge, etc).

2. Seu navegador estabelece uma conexão TCP/IP com o servidor de destino e envia uma requisição GET HTTP com o endereço que você digitou.

3. O servidor interpreta sua requisição e de acordo com o que foi solicitado, uma resposta HTTP é retornada ao seu navegador.

4. A resposta retornada pode ser de sucesso, contendo alguma representação em formato HTML, ou pode ser algum erro, como por exemplo o famoso 404 Not Found, que indica que o endereço/recurso que você solicitou não pôde ser encontrado.

5. Em caso de sucesso, o seu navegador interpreta o HTML e você consegue navegar pela página renderizada.

Todo esse processo é repetido enquanto você está navegando em alguma página Web. Cada link que você clica ou formulário que submete, efetua os passos que discutimos acima.

Se analisarmos detalhadamente o que discutimos, podemos extrair alguns elementos principais; esses elementos são responsáveis por permitirmos criar aplicações Web da forma que conhecemos hoje, e são esses elementos que vamos detalhar melhor abaixo.

Recursos

Um recurso é um elemento abstrato e que nos permite mapear qualquer coisa do mundo real como um elemento para acesso via Web.

Se analisarmos o primeiro passo da lista anterior, podemos relembrar que falamos de um endereço. Esse endereço possui basicamente uma parte que nos permite endereçar algo, e é o que chamamos de URL (*Universal Resource Locator*), um exemplo seria http://www.algaworks.com.

A partir desse endereço, estamos aptos a acessar algum recurso, que poderia ser por exemplo, cursos ou alunos.

Dito isso, podemos verificar que ao acessarmos por exemplo, o endereço http://www.algaworks.com/cursos, é nos retornado uma representação HTML do recurso “cursos” (faça um teste no seu próprio navegador).

Da mesma forma, se acessarmos por exemplo, http://www.algaworks.com/alunos, podemos verificar que esse recurso não existe e que uma página indicando a resposta 404 nos será mostrada.

Interfaces Uniforme

Para interagirmos com os recursos que aprendemos no tópico anterior, o HTTP nos fornece uma interface de operações padronizadas, permitindo que possamos criar, atualizar, pesquisar, remover e executar operações sob um determinado recurso.

Além de operações padronizadas, o HTTP fornece um conjunto de respostas para que os clientes (navegadores, APIs, etc) possam saber, de forma adequada, como agir perante uma determinada resposta.

Abaixo, veja uma breve descrição dos principais métodos e respostas.

GET

O método GET é utilizado quando existe a necessidade de se obter um recurso. Ao executar o método GET sob um recurso, uma representação será devolvida pelo servidor.

Em aplicações Web, normalmente é retornado uma representação HTML.

POST

Utilizamos o método POST quando desejamos criar algum recurso no servidor a partir de uma determinada representação. Exemplo disso é quando fazemos a submissão de algum formulário em uma página Web.

PUT

Semelhante ao método POST, a ideia básica do método PUT é permitir a atualização de um recuso no servidor.

DELETE

Como você já deve estar imaginando, o método DELETE é utilizado com o intuito de remover um recurso em um determinado servidor.

Respostas

Baseado nos métodos que discutimos, o servidor deve processar cada uma das requisições e retornar uma resposta adequada. Veja um resumo de cada uma dessas respostas.

1XX – Informações Gerais  
2XX – Sucesso  
3XX – Redirecionamento  
4XX – Erro no cliente  
5XX – Erro no servidor

Para cada tipo que você pode ver, existe uma série de respostas relacionadas. Por exemplo, se o servidor retornar um “200 OK”, significa que o recurso pedido foi retornado com sucesso.

Por outro lado, se o servidor retornar um “404 Not Found”, significa que o recurso que estamos pedindo não foi encontrado.

Apesar de muitas aplicações Web não seguirem o uso adequado de métodos e respostas, devemos sempre que possível, construir nossas aplicações utilizando-as da maneira mais adequada possível.

Reprentações

Em aplicações Web, a representação mais utilizada é o HTML. Essa representação é utilizada como forma de encapsular as informações relacionadas a um determinado recurso.

Além do HTML, podemos utilizar XML, JSON, ou algum outro formato que melhor atenda o cenário que estamos desenvolvendo.

É importante citar que um recurso pode ter mais de uma representação, ou seja, podemos construir aplicações que para determinados cenários, retornem representações diferentes, baseado nas necessidades de cada cliente.

JSON

JSON (JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição -Dezembro - 1999. JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados.

JSON está constituído em duas estruturas:

Uma coleção de pares nome/valor. Em várias linguagens, isto é caracterizado como um object, record, struct, dicionário, hash table, keyed list, ou arrays associativas.

Uma lista ordenada de valores. Na maioria das linguagens, isto é caracterizado como uma array, vetor, lista ou sequência.

Estas são estruturas de dados universais. Virtualmente todas as linguagens de programação modernas as suportam, de uma forma ou de outra. É aceitavel que um formato de troca de dados que seja independente de linguagem de programação se baseie nestas estruturas.

Em computação, JSON (pronúncia ['dʒejzən], J-son em inglês), um acrônimo de JavaScript Object Notation, é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (parsing) entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor (natureza auto-descritiva). Isto é, um modelo de transmissão de informações no formato texto, muito usado em web services que usa transferência de estado representacional (REST) e aplicações AJAX, substituindo o uso do XML. Padrão foi especificado em 2000 e, definido em 2013 nos dois padrões concorrentes, RFC 7159 e ECMA-404.

O JSON é um formato de troca de dados entre sistemas independente de linguagem de programação derivado do JavaScript.[1][2] Mas a partir de 2017 muitas linguagens de programação incluiram código para gerar, analisar sintaticamente dados em formato JSON e também converter para objetos da linguagem. O tipo de mídia da Internet oficial (MIME) para o JSON é application/jsone nomes de arquivos JSON usam a extensão .json.

O RFC 7493 define um perfil restrito de JSON, conhecido como I-JSON (abreviação de Internet JSON), que visa superar alguns problemas de interoperabilidade com o JSON. Cada documento I-JSON é um documento JSON válido, mas o contrário as vezes não é válido. Apesar de derivado do Javascript, alguns códigos JSON não são considerados Javascript.[3]

Hibernate

Hibernate é o framework para persistência de dados mais utilizado em projetos Java. Sendo uma das primeiras opções a implementar o conceito de mapeamento objeto-relacional (ORM), em pouco tempo se tornou referência entre os desenvolvedores, tendo influenciado, inclusive, a criação da especificação JPA.

Mas afinal, o que é a JPA? JPA é a especificação do Java que dita como os frameworks ORM devem ser implementados. Ela foi criada com o intuito de padronizar essas soluções. Antes de sua criação existiam diversos frameworks e bibliotecas que abstraiam os desafios da persistência com ORM em Java.

Ferramentas para auxiliar nesta tarefa tornaram-se populares entre os desenvolvedores Java e são conhecidas como ferramentas de mapeamento objeto-relacional (ORM). O Hibernate é uma ferramenta ORM open source e é a líder de mercado, sendo a inspiração para a especificação Java Persistence API (JPA). O Hibernate nasceu sem JPA mas hoje em dia é comum acessar o Hibernate pela especificação JPA. Como toda especificação, ela deve possuir implementações. Entre as implementações mais comuns, podemos citar: Hibernate da JBoss, EclipseLink da Eclipse Foundation e o OpenJPA da Apache. Apesar do Hibernate ter originado a JPA, o EclipseLink é a implementação referencial.

O Hibernate abstrai o seu código SQL, toda a camada JDBC e o SQL será gerado em tempo de execução. Mais que isso, ele vai gerar o SQL que serve para um determinado banco de dados, já que cada banco fala um "dialeto" diferente dessa linguagem. Assim há também a possibilidade de trocar de banco de dados sem ter de alterar código Java, já que isso fica como responsabilidade da ferramenta.

JPA

17

Marcar como lido

Anotar

Imprimir

JPA é um framework leve, baseado em POJOS (Plain Old Java Objects) para persistir objetos Java. A Java Persistence API, diferente do que muitos imaginam, não é apenas um framework para Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

Atualmente temos que praticamente todas as aplicações de grande porte utilizam JPA para persistir objetos Java. JPA provê diversas funcionalidades para os programadores, como será mais detalhadamente visto nas próximas seções. Inicialmente será visto a história por trás da JPA, a qual passou por algumas versões até chegar na sua versão atual.

História da Especificação

Após diversos anos de reclamações sobre a complexidade na construção de aplicações com Java, a especificação Java EE 5 teve como principal objetivo a facilidade para desenvolver aplicações JEE 5. O EJB 3 foi o grande percursor para essa mudança fazendo os Enterprise JavaBeans mais fáceis e mais produtivos de usar.

No caso dos Session Beans e Message-Driven Beans, a solução para questões de usabilidade foram alcançadas simplesmente removendo alguns dos mais onerosos requisitos de implementação e permitindo que os componentes sejam como Plain Java Objects ou POJOS.

Já os Entity Beans eram um problema muito mais sério. A solução foi começar do zero. Deixou-se os Entity Beans sozinhos e introduziu-se um novo modelo de persistência. A versão atual da JPA nasceu através das necessidades dos profissionais da área e das soluções proprietárias que já existiam para resolver os problemas com persistência. Com a ajuda dos desenvolvedores e de profissionais experientes que criaram outras ferramentas de persistência, chegou a uma versão muito melhor que é a que os desenvolvedores Java conhecem atualmente.

Dessa forma os líderes das soluções de mapeamento objetos-relacionais deram um passo adiante e padronizaram também os seus produtos. Hibernate e TopLink foram os primeiros a firmar com os fornecedores EJB.

O resultado final da especificação EJB finalizou com três documentos separados, sendo que o terceiro era o Java Persistence API. Essa especificação descrevia o modelo de persistência em ambos os ambientes Java SE e Java EE.

JPA 2.0

No momento em que a primeira versão do JPA foi iniciada, outros modelos de persistência ORM já haviam evoluído. Mesmo assim muitas características foram adicionadas nesta versão e outras foram deixadas para uma próxima versão.

A versão JPA 2.0 incluiu um grande número de características que não estavam na primeira versão, especialmente as mais requisitadas pelos usuários, entre elas a capacidade adicional de mapeamento, expansões para a Java Persistence Query Language (JPQL), a API Criteria para criação de consultas dinâmicas, entre outras características.

**Ionic**

Introdução ao Ionic

O desenvolvimento de aplicativos para dispositivos móveis nunca esteve tão em alta, e a perspectiva é que esses números continuem subindo. Se você quer iniciar nessa área, o devcast abaixo traz um bate papo sobre o assunto e um direcionamento para os seus estudos:

O Ionic é um framework open source para desenvolvimento de aplicativos móveis multiplataforma. Para isso, possibilita a implementação do app utilizando tecnologias comumente empregadas na construção do Front-end de soluções web: HTML, CSS e JavaScript. No entanto, como diferencial em relação ao frameworkque adota como base, o Apache Cordova, traz recursos que simplificam ainda mais o desenvolvimento e dão ao app um aspecto mais profissional.

Esses diferenciais estão relacionados ao conjunto de componentes visuais que podemos utilizar para construção do front-end da solução, assim como ao fato do Ionic trazer consigo outra linguagem e framework para prover uma solução de mais alto nível em termos de código e, consequentemente, projeto. Estamos falando do TypeScript e do Angular (Figura 1).

Assim, temos a inovação do Cordova, Orientação a Objetos em JavaScript, bem como as propostas que o Angular implementa em um só framework para construir apps mobile híbridas. Tudo isso faz do Ionic a principal opção quando o objetivo é criar apps mobile multiplataforma.

Primeiros passos no Ionic

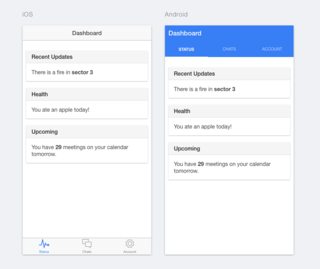
Conhecido o que é o Ionic, podemos começar a programar com ele e criar nossos primeiros aplicativos. Para isso, sugerimos o curso a seguir, que ensina como baixar e instalar o projeto, apresenta a estrutura do framework, assim como seus principais recursos. Você aprenderá sobre tudo isso e ao final também terá desenvolvido um CRUD básico.

Cordova

Bem resumidamente, o [Cordova](https://cordova.apache.org/) é responsável por fazer seu código Javascript acessar **recursos nativos** do dispositivo, como câmera, GPS, acelerômetro, dentre outros. Já o [Ionic](https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiIw4fP35vWAhXBipAKHdSJAPAQFggnMAA&url=https%3A%2F%2Fionicframework.com%2F&usg=AFQjCNFs_Vi2jRGpAkX2cqIN0vXtw-ERgA) é um frameworkque cuida da parte visual da aplicação, de como ela será apresentada para o usuário final, criando aparências muito similares ao de apps nativos.

Como, suponho eu que já tenha lido, o Ionic é construído sobre o Cordova, isto é, o Apache Cordova é a camada inferior, que cuida da parte de transformação do aplicativo em pacotes para cada sistema operacional.

Veja abaixo uma comparação usando o mesmo código, porém compilados para IOS e Android.

[](https://i.stack.imgur.com/508G8m.png)

Perceba que o Ionic faz o tratamento do código para deixar a aparência de acordo com o sistema que você usa, sendo ele Android, iOS, etc.

O Apache Cordova trabalha com outros frameworks além do Ionic, como [jQuery Mobile](https://jquerymobile.com/), [Dojo Mobile](https://dojotoolkit.org/reference-guide/1.10/dojox/mobile.html), [Sencha Touch](https://www.sencha.com/products/touch/), [Lungo](https://github.com/tapquo/Lungo.js/), [Mobile Angular UI](http://mobileangularui.com/).

A conclusão é que são duas ferramentas diferente, dependentes, sendo o Cordova uma ponte para tratar da comunicação com hardware, funcionando como "backend" e o Ionic para trabalhar em questões de aparência para o usuário final, como "frontend".

Typescript

Ao longo da nossa trajetória como desenvolvedores de sistemas web, nos deparamos com a necessidade de utilizarmos a linguagem de programação JavaScript em várias das nossas aplicações. Considerada uma das linguagens mais populares do mercado, ela ainda não possui, mesmo com tanto tempo no mercado, alguns recursos e características normalmente necessárias no desenvolvimento de software de grande escala.

Diferentemente do que ocorria alguns anos atrás, onde JavaScript era utilizada apenas no lado cliente das aplicações, atuando na validação de formulários e composição de elementos da interface, atualmente temos iniciativas que levam essa linguagem a representar a parte principal de aplicações mobile (em frameworks de desenvolvimento híbrido como Apache Cordova) e web (no lado servidor com Node.js). Nestes cenários, os projetos não se resumem mais a pequenos arquivos com algumas linhas de código, muitas vezes apenas utilizando outros frameworks front-end. Agora, passa a ser necessária a possibilidade de construir arquiteturas mais sólidas, onde se possa organizar adequadamente o código e aplicar as melhores práticas e técnicas de programação, como a Orientação a Objetos e, dentro desta, o uso de interfaces.

**TypeScript**

Criada pela Microsoft, TypeScript está provando ser uma escolha comum entre os desenvolvedores ASP.NET. Não se trata, na verdade, de uma linguagem completamente nova, mas sim um superset (ou superconjunto) do JavaScript.

Com TypeScript dispomos de recursos que melhor suportam o uso da Programação Orientada a Objetos, que tem como base quatro princípios fundamentais: encapsulamento, herança, abstração e polimorfismo, os quais veremos de forma mais detalhada a seguir. A POO sempre foi um problema ao ser aplicada em JavaScript, devido a sua sintaxe não permitir escrever classes, por exemplo, de forma tão clara, além da fraca tipagem de dados. O TypeScript oferece então uma forma de corrigir ou contornar esses problemas, adicionando funcionalidades que quando compiladas resultarão em código JavaScript novamente. Porém, agora o desenvolvedor lidará diretamente com uma sintaxe simplificada, mais clara e amplamente suportada por editores de código modernos.

O TypeScript, que no momento da publicação deste artigo encontrava-se em sua versão 1.8, teve como seu principal desenvolvedor o mesmo criador da linguagem C#, Anders Hejlsberg, e sua equipe na Microsoft. Visando aproveitar o máximo da linguagem e contar com a adesão e colaboração da comunidade técnica, o projeto é open source e baseia-se nos padrões ES6 (ECMAScript), que podem ser compilados para JavaScript.

Por ser um superconjunto do JavaScript, qualquer código dessa linguagem pode ser colocado em um arquivo TypeScript, que possui a sua extensão “.ts”, e utilizado diretamente. Este é um ponto bastante positivo, pois podemos utilizar códigos JavaScript já existentes, sem a necessidade de realizarmos grandes conversões. Uma vez que tenhamos um arquivo TypeScript salvo, podemos compilá-lo para JavaScript utilizando a ferramenta de compilação tsc.exe, ou mesmo utilizando task runners (**BOX 1**) como o Grunt ou mesmo o Gulp.

Mysql

Spring

**O que é o Spring Framework**

Muitos já ouviram falar no Spring Framework, mas a complexidade inicial afasta ou assusta bastante os iniciantes do mundo Java. Sim, a principio e **principalmente para um iniciante o Spring Framework pode parecer nada simples, mas para desenvolvedores mais experientes, logo pode-se perceber o encanto e a mágica do Spring Framework**. Nesse artigo vamos desvendar um pouco desse completíssimo framework.

[**Spring é um framework de código aberto (open source)**](http://www.devmedia.com.br/curso/introducao-ao-spring-framework/191), criado por Rod Johnson, em meados de 2002, e apresentado no seu livro Expert One-on-One: JEE Design and Development. Foi criado com o intuito simplificar a programação em Java, possibilitando construir aplicações que antes só era possível utilizando EJB’s.

O Spring atualmente possui diversos módulos como Spring Data (trata da persistência), Spring Security (trata da segurança da aplicação) entre outros módulos. Mas o principal (core) pode ser utilizado em qualquer aplicação Java, as principais funcionalidades são a injeção de dependência (CDI) e a programação orientada a aspectos (AOP), cabe ao desenvolvedor dizer ao Spring o que quer usar. O que faz dele uma poderosa ferramenta, pois não existe a necessidade de se arrastar todas as ferramentas do framework para criar uma aplicação simples.

O Spring Framework fornece um modelo abrangente de programação e configuração para aplicativos corporativos modernos baseados em Java - em qualquer tipo de plataforma de implementação.

Um elemento-chave do Spring é o suporte de infra-estrutura no nível do aplicativo: o Spring concentra-se no "encanamento" de aplicativos corporativos para que as equipes possam se concentrar na lógica de negócios no nível do aplicativo, sem vínculos desnecessários com ambientes de implementação específicos.

**Política de suporte e migração**

Para obter informações sobre requisitos mínimos, orientação sobre como atualizar de versões anteriores e políticas de suporte, confira [a página wiki oficial do Spring Framework](https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Versions)

**Características**

* [Principais tecnologias](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html) : injeção de dependência, eventos, recursos, i18n, validação, ligação de dados, conversão de tipo, SpEL, AOP.
* [Teste](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/testing.html) : objetos mock, framework TestContext, Spring MVC Test WebTestClient,.
* [Acesso a dados](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/data-access.html) : transações, suporte a DAO, JDBC, ORM, Marshalling XML.
* [Frameworks](https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html) web [Spring MVC](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html) e [Spring WebFlux](https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html) .
* [Integração](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/integration.html) : remoting, JMS, JCA, JMX, email, tarefas, agendamento, cache.
* [Idiomas](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/languages.html) : Kotlin, Groovy, linguagens dinâmicas.

**Palestras da SpringOnePlatform 2017**

Spring Boot

O Spring Boot é um projeto da Spring que veio para facilitar o processo de configuração e publicação de nossas aplicações. A intenção é ter o seu projeto rodando o mais rápido possível e sem complicação.

Ele consegue isso favorecendo a **convenção sobre a configuração**. Basta que você diga pra ele quais módulos deseja utilizar (WEB, Template, Persistência, Segurança, etc.) que ele vai reconhecer e configurar.

Você escolhe os módulos que deseja através dos *starters* que inclui no pom.xml do seu projeto. Eles, basicamente, são dependências que agrupam outras dependências. Inclusive, como temos esse grupo de dependências representadas pelo *starter*, nosso pom.xml acaba por ficar mais organizado.

Apesar do Spring Boot, através da convenção, já deixar tudo configurado, nada impede que você crie as suas customizações caso sejam necessárias.

O maior benefício do Spring Boot é que ele nos deixa mais livres para pensarmos nas regras de negócio da nossa aplicação.

Spring Data

Apesar das incríveis funcionalidades existentes na especificação JPA, muitas delas são repetitivas, fazendo com que o mesmo código seja reescrito diversas vezes, ou exigem muito código para operações simples. Tendo em vista esse cenário, o Spring criou o framework Spring Data JPA, cujo principal objetivo é permitir que o desenvolvimento com JPA se torne mais fácil, prático e menos repetitivo.

O Spring Data JPA é um framework que nasceu para facilitar a criação dos nossos repositórios.

Ele faz isso nos liberando de ter que implementar as interfaces referentes aos nossos repositórios (ou DAOs), e também já deixando pré-implementado algumas funcionalidades como, por exemplo, de ordenação das consultas e de paginação de registros.

Ele (o Spring Data JPA) é, na verdade, um projeto dentro de um outro maior que é o [Spring Data](http://projects.spring.io/spring-data/). O Spring Data tem por objetivo facilitar nosso trabalho com persistência de dados de uma forma geral. E além do Spring Data JPA, ele possui vários outros projetos:

* Spring Data Commons
* Spring Data Gemfire
* Spring Data KeyValue
* Spring Data LDAP
* Spring Data MongoDB
* Spring Data REST
* Spring Data Redis
* Spring Data for Apache Cassandra

…mas, provavelmente, o mais utilizado desses é o Spring Data JPA, que é o nosso assunto aqui.

Maven

**O que é o Maven?**

Em seu cerne, **o Maven é uma ferramenta de gerenciamento e automação de construção (*build*) de projetos**. Entretanto, por fornecer diversas funcionalidades adicionais através do uso de plugins e estimular o emprego de melhores práticas de organização, desenvolvimento e manutenção de projetos, é muito mais do que apenas uma ferramenta auxiliar.

Um desenvolvedor que seja alocado em um projeto Java EE que **utilize o Maven corretamente** não terá que saber de imediato quais dependências (bibliotecas) o projeto necessita para compilar e executar, não precisará descobrir onde obtê-las e nem irá se preocupar em como realizar a construção do pacote do aplicativo. Com um comando simples, como mvn install, na raiz do código-fonte do projeto, instruirá o Maven a gerar o código extra necessário (cliente de um web service, por exemplo), validar e compilar o projeto, testá-lo através de seus testes unitários e gerar o pacote com o código compilado. Outras etapas poderiam incluir auditoria de qualidade de código, documentação, geração de estatísticas, entre diversas possibilidades.

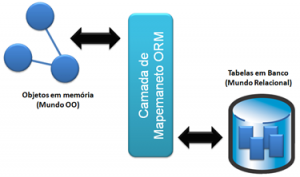
DevCast: [**O que é o Maven?**](https://www.devmedia.com.br/o-que-e-o-maven/36861)  
Neste DevCast vamos falar sobre uma tecnologia bastante interessante: O Maven! Descubra conosco o que é essa tecnologia!

Outra **característica do Maven** é estimular a adoção de boas práticas, porque uma das formas utilizadas por ele para reduzir o esforço de configuração do projeto é a utilização do conceito de *programação por convenção* (do inglês *convention over configuration*), em que a ferramenta assume que o seu usuário fará as coisas da forma como ela preconiza como ideais (estrutura de diretórios padrão, por exemplo), e o livra de ter que declarar algo que se repetirá em todo projeto. o incorporar as práticas aceitas pela comunidade Java como as mais indicadas para projetos Java EE, o **Maven acaba não só disseminando-as para novos desenvolvedores**, como também as padroniza entre os projetos em que ele é empregado, permitindo que novatos se localizem muito mais rapidamente dentro de projetos novos. Obviamente, pode-se definir manualmente o que é assumido como padrão, ao preço do aumento na carga de trabalho para a configuração inicial do projeto.

ORM

ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam. Ultimamente tem sido muito utilizada e vem crescendo bastante nos últimos anos.

Este crescimento tem se dado principalmente pelo fato de muitos desenvolvedores não se sentirem a vontade em escrever código SQL e pela produtividade que esta técnica nos proporciona. Existem ótimos ORM´s como Hibernate, NHibernate, Entity Framework e etc.

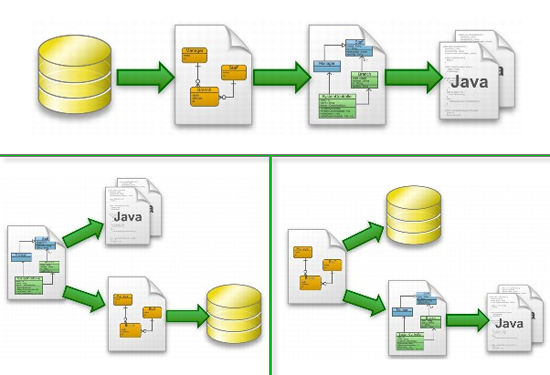
**Figura 1**. ORM

Tudo começa como mostrado na **Figura 1.**Existem dois mundos: o relacional e o orientado a objetos.

No mundo relacional prevalecem princípios matemáticos com a finalidade de armazenar e gerenciar corretamente os dados, de forma segura e se trabalha com a linguagem SQL que é utilizada para dizer o banco de dados “O QUE?” fazer e não como fazer.

Já no mundo orientado a objetos trabalhamos com classes e métodos, ou seja, trabalhamos fundamentados na engenharia de software e seus princípios que nos dizem “COMO” fazer. O ORM é justamente, a ponte entre estes dois mundos, ou seja, é ele quem vai permitir que você armazene os seus objetos no banco de dados.

Para isto precisamos fazer um mapeamento dos seus objetos para as tabelas do banco de dados.

**Figura 2**. Como o ORM trabalha

A **Figura 2** nos traz uma ideia de como o ORM trabalha. Ele faz o mapeamento da sua classe para o banco de dados e cada ORM tem suas particularidades para gerar o SQL referente a inserção do objeto que corresponde a uma tabela no banco de dados e realizar a operação. Utilizando um ORM, também se ganha produtividade, pois deixa-se de escrever os comando SQL para deixar que o próprio ORM, faça isto por você.

Bom pessoal, chegamos ao fim deste nosso artigo. Espero ter ajudado e contribuído de alguma forma para o crescimento profissional de cada leitor. Um abraço e até a próxima.

DTO

**Data Transfer Object** (DTO) ou simplesmente **Transfer Object** é um padrão de projetos bastante usado em Java para o transporte de dados entre diferentes componentes de um sistema, diferentes instâncias ou processos de um sistema distribuído ou diferentes sistemas via serialização.

A ideia consiste basicamente em agrupar um conjunto de atributos numa classe simples de forma a otimizar a comunicação.

Numa chamada remota, seria ineficiente passar cada atributo individualmente. Da mesma forma seria ineficiente ou até causaria erros passar uma entidade mais complexa.

Além disso, muitas vezes os dados usados na comunicação não refletem exatamente os atributos do seu modelo. Então, um DTO seria uma classe que provê exatamente aquilo que é necessário para um determinado processo.

Em alguns casos, usa-se o DTO ou TO para mapear informações obtidas do banco de dados e então usar numa View (MVC). Isso não é completamente errado, e até pode otimizar a apresentação dos dados, afinal o Controller já recebe as informações prontas para uso. Entretanto, isso pode também acabar em um modelo muito poluído com informações redundantes.

Quando tenho um domínio bem estruturado, prefiro criar beans que representam esse modelo. Esses beans geralmente são chamados de Entidades. Aí, em determinados casos (como em certas pesquisas no banco de dados baseadas em views ou que possuem joins ), crio um tipo de TO ou DTO para facilitar o transporte desses dados.

S3 da Amazon AWS

As empresas atuais precisam coletar, armazenar e analisar dados em escalas massivas com simplicidade e segurança. O Amazon S3 é um [armazenamento de objetos](https://aws.amazon.com/what-is-cloud-object-storage/) criado para armazenar e recuperar qualquer quantidade de dados de qualquer local: sites e aplicativos móveis, aplicativos corporativos e dados de sensores ou dispositivos da IoT. O serviço foi projetado para oferecer resiliência de 99,999999999% e armazena dados para milhões de aplicativos usados por líderes de mercado em todos os setores. O S3 oferece recursos abrangentes de segurança e conformidade que cumprem até os requisitos normativos mais rigorosos. Os clientes podem gerenciar de forma flexível dados de otimização de custo, controle de acesso e conformidade. O S3 oferece funcionalidade de consultas no local, o que permite executar diretamente análises avançadas com dados ociosos no S3. Além disso, o Amazon S3 é o serviço de armazenamento na nuvem que conta com o maior suporte no mercado, integrado à maior comunidade de soluções de terceiros e parceiros integradores de sistemas, bem como a outros serviços da AWS.

OAuth

O que é OAuth 2.0 e qual a sua utilidade?

O OAuth é um protocolo de autorização para API's web voltado a permitir que aplicações client acessem um recurso protegido em nome de um usuário.

Quando desenvolvemos uma API web temos em mente que ela seja consumida por aplicações client. A ideia é que a lógica de negócio em si e os dados da aplicação podem ser acessados por meio de uma API web. O que o usuário final vai realmente usar, por outro lado, é um exemplo de aplicação client.

Um exemplo disso seria com as aplicações de página única (SPA's) construídas em JavaScript, ou então um aplicativo móvel construído com android.

Além disso, as aplicações client não são necessariamente só as aplicações que interagem diretamente com o usuário final. Qualquer aplicação que para seu funcionamento precisa utilizar a API web construída é uma aplicação client dessa API.

Nesse sentido, quando utilizamos API's web é comum e necessário considerarmos que uma ou mais aplicações vão precisar interagir com essa API. Mais do que isso, é comum que aplicações de *terceiros* precisem usar essas API's.

Se existem recursos protegidos na API, ou seja, que devem ter o acesso controlado e só podem ser acessados por um usuário específico, então é preciso considerar como levar em conta essas diversas aplicações client, porque afinal, o usuário não vai a acessar o recurso diretamente, mas vai, na verdade, delegar essa tarefa a uma aplicação client.

Uma opção inicial seria que o usuário especificasse em cada aplicação client os seus dados de acesso (como login e senha), mas isso tem alguns problemas. Alguns desses problemas são listados na [própria especificação](https://tools.ietf.org/html/rfc6749), que estão abaixo em tradução livre:

1. As aplicações de terceiros precisam salvar as credenciais do usuário para uso futuro, tipicamente uma senha em texto claro.
2. Os servidores precisam suportar autenticação por senha, apesar das falhas de segurança inerentes a esse tipo de autenticação.
3. As aplicações de terceiros ganham um acesso muito amplo aos recursos protegidos do usuário, deixando o usuário sem a possibilidade de restringir a duração desse acesso ou limitar o acesso a um subconjunto dos recursos.
4. Os usuários não podem revogar o acesso a uma aplicação de terceiro de forma individualizada sem revogar o acesso a todas as aplicações de terceiros, e para fazer isso é necessário trocar a senha.

É por isso tudo que é necessário o OAuth para fazer autorização de forma segura e coerente no contexto de API's web.

O que o OAuth 2.0 faz?

O OAuth, sendo uma especificação, descreve de forma detalhada uma maneira de tratar todos os problemas considerados. Ele estabelece definições como:

* Resource Owner: a entidade que é capaz de controlar o acesso a um recurso protegido. É o "dono do recurso", mas nem sempre é uma pessoa. Quando ele é uma pessoa ele é o usuário final.
* Resource Server: o servidor que possui os recursos protegidos e recebe as requisições para acessar esses recursos.
* Authorization Server: é um servidor que gera tokens de acesso para permitir que o client acesse os recursos que o resource owner permitiu com o nível de acesso que o resource owner especificou.
* Client: é a aplicação que acessa os recursos no resource server em nome do usuário. O client pode ser qualquer tipo de aplicação que faça isso.

Com essas definições, o OAuth estabelece que quando uma aplicação client precisa acessar um recurso protegido no resource server ele deve obter um "token de acesso". Esse "token de acesso" é um token contendo as informações que caracterizam o acesso que o resource owner permitiu aos recursos protegidos.

Perceba que isso resolve os problemas especificados pois: o client não precisa das credenciais do resource owner, somente precisa de um token gerado pelo authorization server, que é uma aplicação confiável. Através do token, o resource owner pode especificar um acesso personalizado, e permitir ao client acesso somente a um subconjunto dos recursos protegidos. Além disso, é fácil revogar o acesso, pois basta invalidar o token concedido, sem falar que os tokens possuem um tempo de vida específico e não podem ser usados para sempre.

Além disso tudo, o OAuth especifica fluxos. Cada fluxo é para uma situação particular de interação entre cliente, authorization server e resource server. Esses fluxos são detalhados e dizem todos os passos necessários para, naquele caso, obter o acesso desejado. Para ver esses fluxos em detalhes é recomendável olhar na [especificação](https://tools.ietf.org/html/rfc6749#section-1.3).

Existem alternativas?

Existem outras formas de gerenciar autorização em API's web como o tradicional modo de usar usuário e senha, mas como já explicado, esses métodos mais tradicionais tendem a ter alguns problemas que o OAuth foi pensado para resolver.

Controle de Acesso

API’S

API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. A sigla API refere-se ao termo em inglês "Application Programming Interface" que significa em tradução para o português "Interface de Programação de Aplicativos".

Uma API é criada quando uma empresa de software tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço. Existem vários deles que disponibilizam seus códigos e instruções para serem usados em outros sites da maneira mais conveniente para seus usuários. O Google Maps é um dos grandes exemplos na área de APIs. Por meio de seu código original, muitos outros sites e aplicações utilizam os dados do Google Maps adaptando-o da melhor forma a fim de utilizar esse serviço.

Quando uma pessoa acessa uma página de um hotel, por exemplo, é possível visualizar dentro do próprio site o mapa do Google Maps para saber a localização do estabelecimento e verificar qual o melhor caminho para chegar até lá. Esse procedimento é realizado por meio de uma API, onde os desenvolvedores do site do hotel utilizam do código do Google Maps para inseri-lo em um determinado local de sua página.

Participe do nosso **GRUPO CANALTECH DE DESCONTOS** do [Whatsapp](https://bit.ly/2NREpI6" \t "_blank) e do [Facebook](https://bit.ly/2LqiPsV" \t "_blank) e garanta sempre o menor preço em suas compras de produtos de tecnologia.

Através das APIs, os aplicativos podem se comunicar uns com os outros sem conhecimento ou intervenção dos usuários. Elas funcionam através da comunicação de diversos códigos, definindo comportamentos específicos de determinado objeto em uma interface. A API liga as diversas funções em um site de maneira que possam ser utilizadas em outras aplicações. Sistemas de pagamento online são um bom exemplo de funcionalidade das APIs que rodam de maneira automática. De modo geral, a API é composta de uma série de funções acessíveis somente por meio de programação.

Recentemente, a utilização das APIs tem se espalhado nos plugins, que complementam a funcionalidade de um determinado programa. Os desenvolvedores de um programa principal criam uma API específica e fornecem a outros criadores, que desenvolvem plugins para aumentar o potencial e as funcionalidades do programa.

Os sistemas operacionais também possuem suas APIs com as mesmas funções descritas acima. Por exemplo, o Windows possui APIs como a Telephony API, Win16 API e Win32 API. Quando um usuário executa um programa que envolva algum processo do sistema operacional, é bem provável que o Windows faça uma conexão entre o software e alguma de suas APIs.

BCrypt

**bcrypt** é um método de criptografia do tipo [hash](https://pt.wikipedia.org/wiki/Hash" \o "Hash) para senhas baseado no [Blowfish](https://pt.wikipedia.org/wiki/Blowfish" \o "Blowfish). Foi criado por Niels Provos e David Mazières e apresentado na conferência da USENIX em 1999 [[1]](https://pt.wikipedia.org/wiki/Bcrypt#cite_note-1).

Este método apresenta uma segurança maior em relação à maioria dos outros métodos criptográficos que é a implementação da variável "custo" que é proporcional à quantidade de processamento necessária para criptografar a senha. O método é conhecido como hash adaptativo às melhorias futuras de hardware por ter esta característica, pois pode permanecer resistente à ataques do tipo "força-bruta" com o tempo usando custos maiores de processamento.

Ao contrário do método tradicional "crypt" (concebido em 1976), o algoritmo bcrypt não possui as restrições da época que eram pouco processamento e poucos espaço (bytes) para guardar o salt e o hash gerado da senha criptografada [[2]](https://pt.wikipedia.org/wiki/Bcrypt#cite_note-2).

O algoritmo bcrypt foi implementado em diversas linguagens como Python, Perl, Ruby, Java, C# e outras [[3]](https://pt.wikipedia.org/wiki/Bcrypt#cite_note-3), além de possuir implementação também para a função "crypt" do UNIX.

Biblioteca JWT

JWT autenticação

JWT (JSON Web Token), com o T em maiúsculo, ele é um sistema de transferência de dados que pode ser enviado via URL, POST ou em um cabeçalho HTTP (header) de maneira "segura", essa informação é assinada digitalmente, por exemplo assinada com o algoritmo [HMAC](https://pt.wikipedia.org/wiki/HMAC), ou um par de chaves pública/privada usando [RSA](https://pt.wikipedia.org/wiki/RSA).

Um exemplo de situação que ele pode ser usado é a autenticação, uma vez que o usuário está conectado, cada pedido que vier irá incluir o JWT, permitindo que o usuário continue acessando serviços e recursos que são foram liberados com tal token (Single sign-on (SSO)).

A estrutura do JWT é em 3 partes divididas por pontos:

* **Cabeçalho**

Consiste em 2 partes geralmente, o tipo do token, que é JWT e o tipo do algorítimo de HASH, como por exemplo *HMAC SHA256* ou *RSA*, exemplo:

{

"alg": "HS256",

"typ": "JWT"

}

Este será a primeira parte do JSON codificado em base64 para formar o JWT

* **Payload** (carga de dados ou os dados enviados)

Esta é a segunda parte do token, ele contem os "pedidos". Esse "pedidos" são declarações sobre uma entidade (geralmente, o usuário) e metadados adicionais e existem 3 tipos: *reserved*, *public*, e *private claims*. Um exemplo:

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

* **Assinatura**

Para gerar a assinatura você deve usar o Cabeçalho e o Payload codificando eles, usando o algorítimo definido no cabeçalho e assinar, no exemplo do Cabeçalho usamos HS256 (HMAC SHA256), então deve ficar algo como:

HMACSHA256(

base64UrlEncode(Cabeçalho) + "." +

base64UrlEncode(Payload),

secret)

A assinatura é usada para verificar que o emissor da "JWT", é realmente "quem" se diz ser e também para verificar se a mensagem não foi alterada ao longo do caminho.

O resultado seria algo como:

(Cabeçalho em base64).(Payload em base64).(Assinatura em base64)

Um exemplo de envio via header no HTTP:

GET /foo/bar HTTP/1.1

Host: www.exemplo.com

Authorization: Bearer (Cabeçalho em base64).(Payload em base64).(Assinatura em base64)

Ou seja por ser algo que é transmitido via HTTP pode ser usado com qualquer linguagem que suporte os requisitos mínimos para gerar o TOKEN e enviar uma requisição HTTP, como Java, C#, PHP, Python.

CRUD

**CRUD** ([acrónimo](https://pt.wikipedia.org/wiki/Acr%C3%B3nimo) de **C**reate, **R**ead, **U**pdate e **D**elete na [língua Inglesa](https://pt.wikipedia.org/wiki/L%C3%ADngua_Inglesa)) para as quatro operações básicas utilizadas em [bases de dados relacionais](https://pt.wikipedia.org/w/index.php?title=Sistema_de_gerenciamento_de_bases_de_dados&action=edit&redlink=1) (RDBMS) ou em interface para utilizadores para criação, consulta, atualização e destruição de dados.