Unisanta Science and Technology, **2018, 3, July**
Published Online 2018 Vol.7 N°1 http://periodicos.unisanta.br/index.php/sat)

UNISANTA
Science and Technology
ISSN 2317-1316

Um estudo Aplicado sobre *Framework* Angular 2

Luiz Carlos da Fonseca Junior, Marcio Roberto da Fonseca e Hélio Augusto de Lima Rangel

UNISANTA – Universidade Santa Cecília – Departamento de Sistemas de Informação
Rua Oswaldo Cruz, 266 - Santos-SP, Brasil - CEP: 11045-100

E-mail*:* luizmew@hotmail.com
Received April, 2017

Resumo: O objetivo deste artigo é demonstrar através de uma aplicação exemplo, alguns dos principais recursos do *framework* Angular 2, sua capacidade de produzir aplicações web, bem como servir de base para diversas ferramentas de desenvolvimento para plataformas móveis.

Palavras chave: Sistema de Informação, Programação, framework de desenvolvimento web; angular 2.

An applied Study on the Angular Framework 2

Abstract: The purpose of this article is to demonstrate through an example application some of the key features of the Angular 2 framework, its ability to produce web applications as well as serve as the basis for various development tools for mobile platforms.

Keywords: Information System, Programming, web development framework; angular 2.

1. Introdução

O *framework* de desenvolvimento web Angular 2 - que na edição deste artigo se apresenta em sua versão 2.1- é fruto de uma parceria da Google com a Microsoft e baseia-se em *TypeScript*, linguagem oriunda da Microsoft que embora possa trabalhar diretamente com outras linguagens, recursos como *EcmaScript* 5 e *EcmaScript* 6 além de Dart, faz uso do conceito de SPA (*Single Page Applications*) ou aplicações de uma única página.

Frameworks são desenvolvidos com o objetivo de se aumentar a produtividade, muitas vezes oferecendo soluções para problemas comuns ao desenvolvimento.

> "[...]outra vantagem da grande parte dos *framework*s é que tarefas repetitivas podem ser automatizadas, é o conceito conhecido como *dry – don't repeat yourself*, que pode ser traduzido para "não se repita". Em uma aplicação que tenha de manipular dados vindos de uma tabela na base de dados, as operações de inclusão, exclusão e alteração são praticamente iguais para todas as tabelas envolvidas. Não teria sentido repetir o esforço para desenvolver esse código-fonte várias vezes, e a geração dessas funções poderia ser automatizada por alguma ferramenta contida no *framework*." (MINETTO, 2007, p. 18).

Com a evolução do HTML e de toda a estrutura que o suporta e auxilia como *Javascript* e CSS, chega-se a solução na qual se baseia também o *framework* Angular 2, que embora carregue em seu nome o "2", foi totalmente reescrito, deixando os problemas de *performance* para sua versão anterior. Essa tríade de desenvolvimento (HTML5, *Javascript* e CSS) aliada a recursos como o gerenciador de pacotes NPM que fornecem diversas opções de pacotes e configurações sem custo, tornam o Angular 2 uma promissora ferramenta para o desenvolvimento de aplicações web.

2. Ferramentas

O desenvolvimento com Angular 2, mesmo não sendo obrigatório, pode ser feito através do *Node* e do seu gerenciador de pacotes oficial, o NPM, que oferece benefícios como servir a aplicação e a *transpilação* (conversão de *Typescript* em *Javascript*) automática do código.

O *Node*, como servidor web suporta o desenvolvimento de aplicações *server-side* baseadas em rede utilizando J*avascript* e o V8 *Javascript* (interpretador

desenvolvido em C++ pelo Google e usado no *Chrome*, que possui a característica única de poder ser baixado e utilizado em qualquer aplicação), ou seja, o *Node* se utiliza desse mecanismo para poder usar *Javascript* no lado do servidor.

> Em linguagens como Java e PHP, cada conexão inicia um novo encadeamento que potencialmente é acompanhado de 2 MB de memória. Em um sistema que tenha 8 GB de RAM, suporta teoricamente um número máximo de 4.000 conexões simultâneas. [...]se necessitasse que sua aplicação web suportasse mais conexões, teria que adicionar mais servidores. Os custos com servidores, tráfego de dados, suporte e questões técnicas, naturalmente aumentariam. (ABERNETHY, 2011, tradução nossa)

O *Node* altera a forma como a conexão é feita no servidor iniciando um processo para cada conexão ao invés de um novo encadeamento eliminando a necessidade de se alocar a memória correspondente, possibilitando que o programa trate dezenas de milhares de conexões simultâneas. Um exemplo de seu potencial pode ser visto na figura 1, na qual foi realizado um *benchmark* com Node em ambiente *multicore*.

O estudo do Angular 2 é um ótimo retorno custo benefício, sua curva de aprendizagem para o desenvolvimento de aplicações é base fundamental para diversos *frameworks* como IONIC, por exemplo, que oferece um *framework* de desenvolvimento para aplicações móveis híbridas que operam tanto sobre plataforma Android como IOs. Para facilitar a construção de aplicações com Angular 2, dispomos de uma ferramenta muito útil, o Angular CLI (*Command Line Interface)* que entre outras funções, resolve todas as dependências da aplicação criada. Sua instalação se dá através do comando: "**npm install -g angular-cli**".

3. Aplicação Exemplo

Com o propósito de exemplificar o uso do Angular 2, é proposta a criação de uma aplicação que consiste de um campo para digitação do valor e um botão de envio para a ação. O valor digitado será usado para consulta através de acesso ao *web service* do site www.viacep.com.br.

3.1 Desenho do Estudo

Como aplicação do estudo desse artigo propõe a criação de uma aplicação básica composta de um campo para a inserção de um CEP (Código de Endereçamento Postal) e um botão para executar uma ação. Esta ação consiste no consumo do *webservice* www.viacep.com.br através da consulta do CEP digitado que fornecerá as informações de logradouro, complemento, bairro, localidade, uf, unidade, números do IBGE (Instituto Brasileiro de Geografia e Estatística) e da GIA (Guia de Informação e Apuração do ICMS).

3.2 Procedimentos e Estrutura

Concluída a instalação do Node, que pode ser obtido em https://nodejs.org/en/, procedemos a criação da aplicação. Utilizando a ferramenta Angular CLI, através do comando "**npm install -g angular-cli**", pelo *prompt* de comando, vamos proceder para criação de nossa aplicação aqui nomeada de WebCEP. Após selecionar o local de criação, executa-se o seguinte comando: "**ng new WebCEP**", esse comando criará a estrutura de arquivos necessária para o funcionamento da aplicação. A estrutura obtida é a mostrada na figura 2.

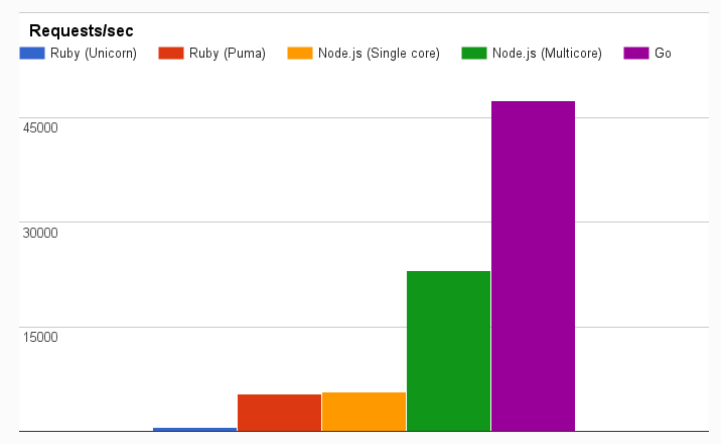

Figura 1. Comparação entre os servidores em relação à quantidade de requisições por segundo que atendem.
Fonte: BIAZUS & VITTI, 2016.

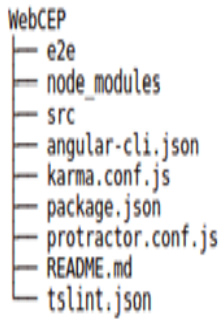

Figura 2. Estrutura de diretórios e arquivos da aplicação.

Na estrutura apresentada na figura 2, encontram-se as pastas *e2e* (*end to end*) que contém arquivos para a realização de testes de ponta a ponta. A pasta *node_modules* contém todos os arquivos necessários para o funcionamento do Angular 2 e qualquer outro módulo que seja adicionado ao projeto. A pasta *src* contém um subpasta de nome *App* onde são armazenados todos os arquivos da aplicação, ainda na raiz da pasta *src*, existem outros arquivos importantes da aplicação:

a. *index.html* – página inicial;
b. *main.ts* – configura a inicialização;
c. *style.css* – folha de estilos global;
d. *tsconfig.json* – responsável pelas configurações de complicação do TypeScript;

Como boa prática recomenda-se criar uma nova pasta para cada módulo criado. Além das pastas citadas, a estrutura criada pelo Angular CLI também cria alguns arquivos na raiz da pasta do projeto, são eles:

a. *angular-cli.json* – inclui a versão e nome da aplicação, diretório de saída dos arquivos gerados e ambientes, entre outras configurações;
b. *karma.conf.js* – arquivo de configuração de testes;
c. *package.json* – configura os módulos necessários para a aplicação funcionar, diferenciando por ambiente, pois nem todos os módulos necessários em desenvolvimento, são necessários no ambiente de produção;
d. *protractor.conf.js* – configura testes automatizados com o *Protractor;*
e. *tslint.js* – responsável per verificar o código TypeScript com regras que podem ser configuradas;

Também fazendo o uso do Angular CLI, foi adicionado o recurso necessário para fazer uso do *framework* Bootstrap através do comando "npm install ng2-bootstrap -save". Incluímos no *head* da página index.html o *link* para o arquivo de estilo <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">.

A figura 3 representa o fluxo de dados e a interação com os recursos do Angular 2:

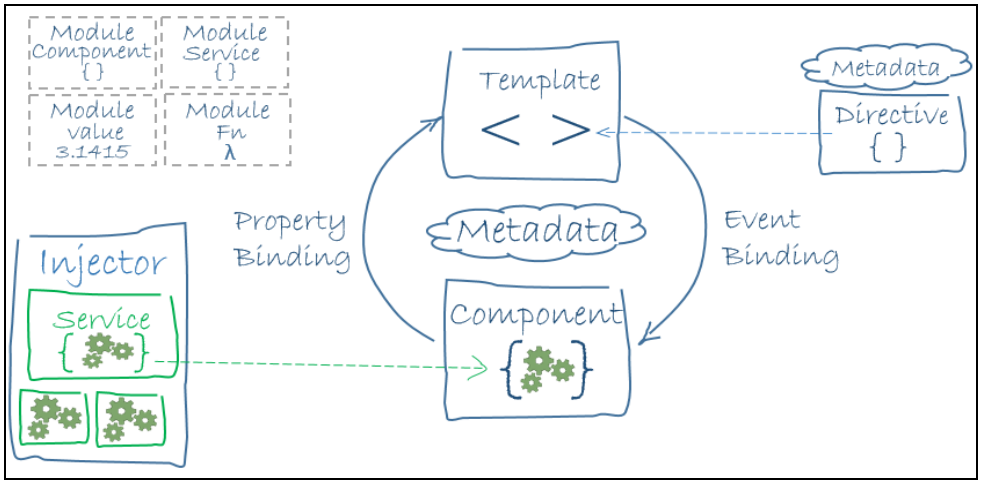

Figura 3. Representação do fluxo de dados e informação entre os componentes do Angular 2. Fonte: (ANGULAR FEATURES, 2016)

3.2.1 Módulos

Módulos são blocos de código com um determinado propósito, como uma classe, uma função ou até mesmo uma constante. Um conjunto de módulos compõe uma aplicação. Como a aplicação fica desmembrada em pequenas partes, se torna mais fácil realizar os testes e as manutenções. Por convenção cada módulo deve ter sua própria pasta contendo arquivos relacionados.

3.2.2 Componentes

Componente faz o papel da *view* e encapsula os módulos *Template*, *Metadata* e *DataBinding*. Praticamente tudo em uma aplicação Angular 2 é um componente, com exceção das regras de negócio que devem estar em um serviço.

O Angular CLI também fornece os meios necessários para a geração de componentes através do comando "**ng g component [nome_do_componente]**", este comando cria os arquivos dos tipos css, html e ts para o funcionamento do componente em uma pasta nomeada com nome atribuído na execução do comando, porém, considerando o tamanho da aplicação deste artigo usaremos o componente que é criado durante o comando "**ng new**", o *app.component* construído como a estrutura representada na figura 4.

```
import { Component } from '@angular/core';

 @Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],

 })
  export class AppComponent {
  title = 'app works!';
 }
```

Figura 4. Estrutura do app.component.ts, imagem .

O componente gerado e representado na figura 4 possui três blocos distintos que são: declaração, decorador e definição. *Export* e *Class* são definições do *TypeScript*. Quando o arquivo for compilado irá gerar um código *Javascript* que criará objetos públicos acessíveis aos outros módulos da aplicação pelo *meta* comando *import***.** A notação *@Component* vista na figura 4 representa o decorador do componente que possui três atributos:

- *selector* - representa a *tag* HTML onde será exibido o componente;
- *templateUrl* - representa o caminho para o *template* deste componente para "renderizar";
- *styleUrls* - representa o caminho até a folha de estilo;

3.2.3 Diretivas

Os *templates* em Angular 2 são dinâmicos, logo, eles precisam de instruções armazenadas em *metadados* de diretivas. O componente representado na figura 4 é uma diretiva. Existem uma série de outras diretivas a disposição do Angular 2, assim como o ***ngFor** que nos permite iterar sobre um objeto.

3.2.4 Roteamento

Embora o Angular 2 trabalhe com o conceito SPA (*Single Page Application*), nada impede que tenhamos diversas telas em nossa aplicação. O *Router* é o responsável pela navegação no Angular 2, a exemplo do *browser* que nos permite interagir com endereços, *links* e botões que avançam ou retornam as páginas de um website. Esse módulo de roteamento se faz necessário quando o projeto é complexo, incluindo serviços especializados como controle de acesso, por exemplo. Na aplicação objeto de estudo deste artigo não se faz necessário seu uso.

3.2.5 *Templates*

É a *view* de um determinado componente que contém basicamente marcações HTML e algumas próprias do Angular que, após processadas, são exibidas no navegador. O *template* da nossa aplicação até aqui, tem apenas a *tag* H1 envolvendo a marcação **{{title}}**, utilizada tanto no Angular, como no Angular 2. Para o nosso propósito vamos alterar o *template* app.compontent.html para o formato representado na figura 5.

Percebe-se que as notações próprias do Angular como o uso de colchetes, parênteses e das chaves duplas, são notações que permitem ao Angular 2 realizar o *binding*[1] de eventos e propriedades.

3.2.6 Metadados

Instruem o Angular 2 a como processar uma classe. Na representação da figura 4, o decorador *@Component* possui os metadados sobre qual *tag* deve ser manipulada, o caminho para o *template* e para a folha de estilos.

[1] Refere-se a conversão de algo para a programação, na aplicação exemplo, o Angular 2, consegue captar o click do mouse e substituir os atributos do input.


```
<h1>
  {{title}}
</h1>
<div class="container">
    <div class="row">
        <div class="col-md-6">
            <pre class="card card-block card-header">Digite o CEP:</pre>
            <input type="text" [(ngModel)]="nrcep" class="form-control" maxlength="8">
            <button type="button" class="btn btn-primary" (click)="consultarCEP()">
                Consultar CEP
            </button><br>
        </div>
    </div>
    <div class="row">
        <h3 class="col-md-6">Informações</h3>
    </div>
    <div class="row">
        <div class="col-md-3"> CEP:</div>
        <div class="col-md-3">{{result?.cep}}</div>
    </div>
    <div class="row">
        <div class="col-md-3"> LOGRADOURO:</div>
        <div class="col-md-3">{{result?.logradouro}}</div>
    </div>
    <div class="row">
        <div class="col-md-3"> BAIRRO:</div>
        <div class="col-md-3">{{result?.bairro}}</div>
    </div>
    <div class="row">
        <div class="col-md-3"> LOCALIDADE:</div>
        <div class="col-md-3">{{result?.localidade}}</div>
    </div>
    <div class="row">
        <div class="col-md-3"> IBGE:</div>
        <div class="col-md-3">{{result?.ibge}}</div>
    </div>
    <div class="row">
        <div class="col-md-3"> GIA:</div>
        <div class="col-md-3">{{result?.gia}}</div>
    </div>
</div>
```

Figura 5. Estrutura atualizada do *template* app.component.html.

3.2.7 *DataBinding*

DataBinding é como o Angular 2 vincula os dados na *view* com os dados do componente. Existem quatro maneiras:

- *Bind* por interpolação representada na aplicação por "{{title}}" onde apenas é feita a apresentação dos dados;
- *Bind* por propriedade onde um valor é associado a uma variável visível no escopo do Angular 2;
- *Bind* por evento onde se associa uma ação a um elemento e está representada na aplicação pelo trecho "(click)";
- Combinação do *bind* por evento e do *bind* por propriedade, essa forma denominada de *Two-Way-DataBinding* sincroniza os valores de ambos os lados assim, sempre que o valor for alterado tanto no *template* como no componente ele será o mesmo em ambos os lados. Esta forma também é representada na aplicação pelo trecho "[(ngModel)]", muito usada em formulários.

3.2.8 Serviços

Serviço representa uma classe, função ou valor que possa ser usado por outro componente e caso haja necessidade, armazenará toda a regra de negócio que corresponda ao componente a qual ele irá servir.

> Uma regra de negócio é: "Uma declaração que define alguns aspectos do negócio. Ela deve ser uma condição ou um fato construído a partir de uma afirmação, uma limitação construída a partir de uma ação ou uma derivação. Ela deve ser atômica não podendo ser quebrada ou decomposta em regras de negócio mais detalhadas" (Hay, D., Healy, K. A., 1995 apud NETTO et al., 2007)

Para a aplicação criada nesse artigo será criado um serviço que fará acesso ao *webservice*. Novamente faremos uso do Angular CLI através do comando "**ng g service app**". Este comando gerará o arquivo representado na figura 6.

```
import { Injectable } from '@angular/core';

@Injectable()
export class AppService {

 constructor() { }

}
```

Figura 6. Conteúdo do arquivo app.service.ts, imagem.

3.2.9 Injeção de Dependência

Um serviço é uma classe consumida por um ou mais componentes. Para evitar que o componente gerencie as classes da qual ele depende o Angular 2 nos oferece o recurso de injeção de dependência, dessa forma nós podemos explicitamente indicar as dependências da qual ele necessita. O Angular CLI já cria os serviços com o decorador *@Injectable* e o respectivo *import.* Completando o projeto com os recursos necessários teremos o código representado na figura 7. Para a chamada ao *webservice* é necessária a importação das classes *Http*, *Headers* e *Response* do módulo HTTP do Angular 2.

A comunicação é feita através de dados no formato JSON, o *headers* foi configurado para esse formato, definimos também um *websocket Observable*, recurso que faz parte do módulo RXJS.

```
import { Injectable } from '@angular/core';
import { Http, Headers, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';

@Injectable()
export class AppService {
  private baseUrl: string = 'https://viacep.com.br/ws/';
  resultado : any;
  constructor(private http : Http){}

  getAll(cep:string): Observable<any>{
    let consulta$ = this.http
      .get(`${this.baseUrl}${cep}/json`, {headers: this.getHeaders()});
      return consulta$;  }

  private getHeaders(){
    let headers = new Headers({ 'Accept': 'application/json' });
    return headers;
  }
}
```

Figura 7. Código demonstra a injeção de dependência.

Ainda no serviço, foi definido os métodos *getHeaders e getAll*, este último é responsável por retornar os valores da consulta.

Para finalizar, retornamos ao componente da nossa aplicação e incluímos a chamada ao serviço e configuramos a injeção de dependência.

A injeção de dependência do *AppService* se dá pela configuração do *providers* e pelo parâmetro no método construtor.

O conteúdo completo do componente está representado na figura 8.

4. Executando a Aplicação

Para a execução do projeto, basta acessar o diretório da aplicação via *prompt* de comando ou terminal e emitir o comando **ng serve**. Isso provoca o *build* da aplicação e, caso tudo ocorra sem problemas, um servidor deverá estar pronto e ativo, podendo ser acessado via endereço http://localhost:4200.

A aplicação deverá mostrar a saída como representada na figura 9. Vale lembrar que qualquer modificação no código da aplicação é refletida automaticamente na página enquanto o servidor estiver ativo.

```
import { Component } from '@angular/core';
import { AppService } from './app.service';
import { OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers : [AppService]
})
export class AppComponent implements OnInit {

  public resultado: Array<string>;
  public nrcep: string;

  constructor(private service: AppService) { }

  ngOnInit() { }

  consultarCEP() {
    this.service
      .getAll(this.nrcep)
      .subscribe(data => { this.resultado = data._body; })
  }
}
```

Figura 8. Conteúdo do arquivo app.component com a chamada ao serviço app.service.ts.

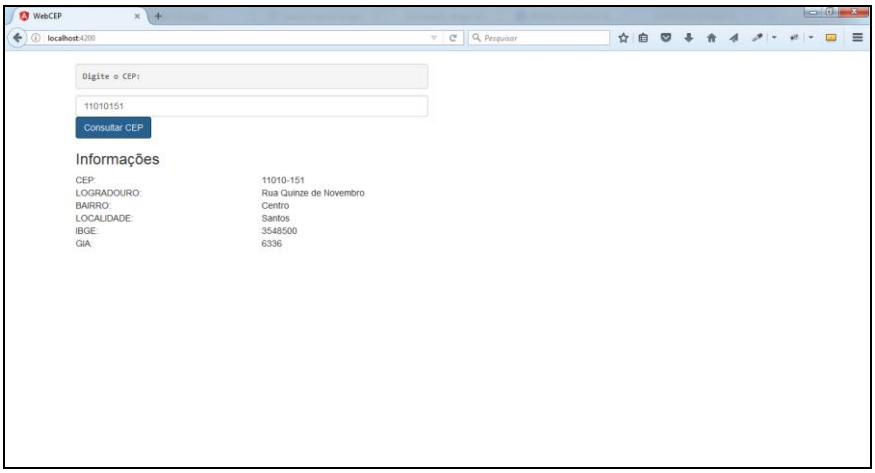

Figura 9. Tela do sistema demonstrando o resultando de uma consulta de CEP.

5. Conclusão

Esse artigo atingiu a proposta que consistia em demonstrar através da construção de uma pequena aplicação completa, a sintaxe simplificada do Angular 2 e o uso do *TypeScript*, evidenciamos que o desenvolvimento modularizado padrão do Angular 2 facilita manutenções, apresenta boa escalabilidade, confiabilidade e robustez. Deixamos a cargo do *Node*, por intermédio do NPM, a instalação de bibliotecas como a ferramenta Angular CLI e o gerenciamento das dependências do projeto, usamos também o servidor web *Node* para hospedar nossa aplicação, onde fizemos uso do *websocket* para executarmos as requisições. E finalmente, todas as vantagens e recursos apresentados fizeram do Angular 2 a base para construção de outros *frameworks*, como o Ionic, utilizado para desenvolvimento de aplicações híbridas.

Referências

ABERNETHY, M., **Just what is Node.js?**, IBM developersWorks, 2011. Disponível em: <https://kyulingcompany.files.wordpress.com/2012/06/os-nodejs-pdf.pdf>. Acesso em 11 out. 2016, 11:12:36.

BIAZUS, G & VITTI, P. Benchmark: Unicorn vs Puma vs Node.js vs Go disponível em http://shipit.resultadosdigitais.com.br/blog/benchmark-unicorn-vs-puma-vs-node-dot-js-vs-go/ , acesso em 20/10/2016.

ANGULAR Features: https://angular.io/resources/images/devguide/architecture/overview2.png>, acesso em 20/10/2016.

ANGULAR Framework. Projeto desenvolvido em parceria Google e Microsoft. Versão 2.1. Disponível em: <https://angular.io/>. Acesso em 16 out. 2016, 14:37:43.

NODE.js. Node.js Foundation. Versão 6.8.1. Disponível em: <https://nodejs.org/en/>. Acesso em 23 out. 2016, 09:21:15.

MINETTO, E. L., **Frameworks para Desenvolvimento em PHP**. 1. ed. São Paulo, Novatec, 2007. p. 18.

NETTO S.E.C. et al., **Desenvolvimento de Sistemas de Informação Flexíveis Utilizando *Framework* com Separação de Regras de Negócio**, IME: Série Informática, Rio de Janeiro, v. 24, set. 2007. Disponível em: <http://www.e-publicacoes.uerj.br/index.php/cadinf/article/viewFile/6520/4638>. Acesso em: 29 out. 2016, 17:51:32.

