Arquitetura front-end com AngularJS

Leonardo Zanivan
Michel Graciano

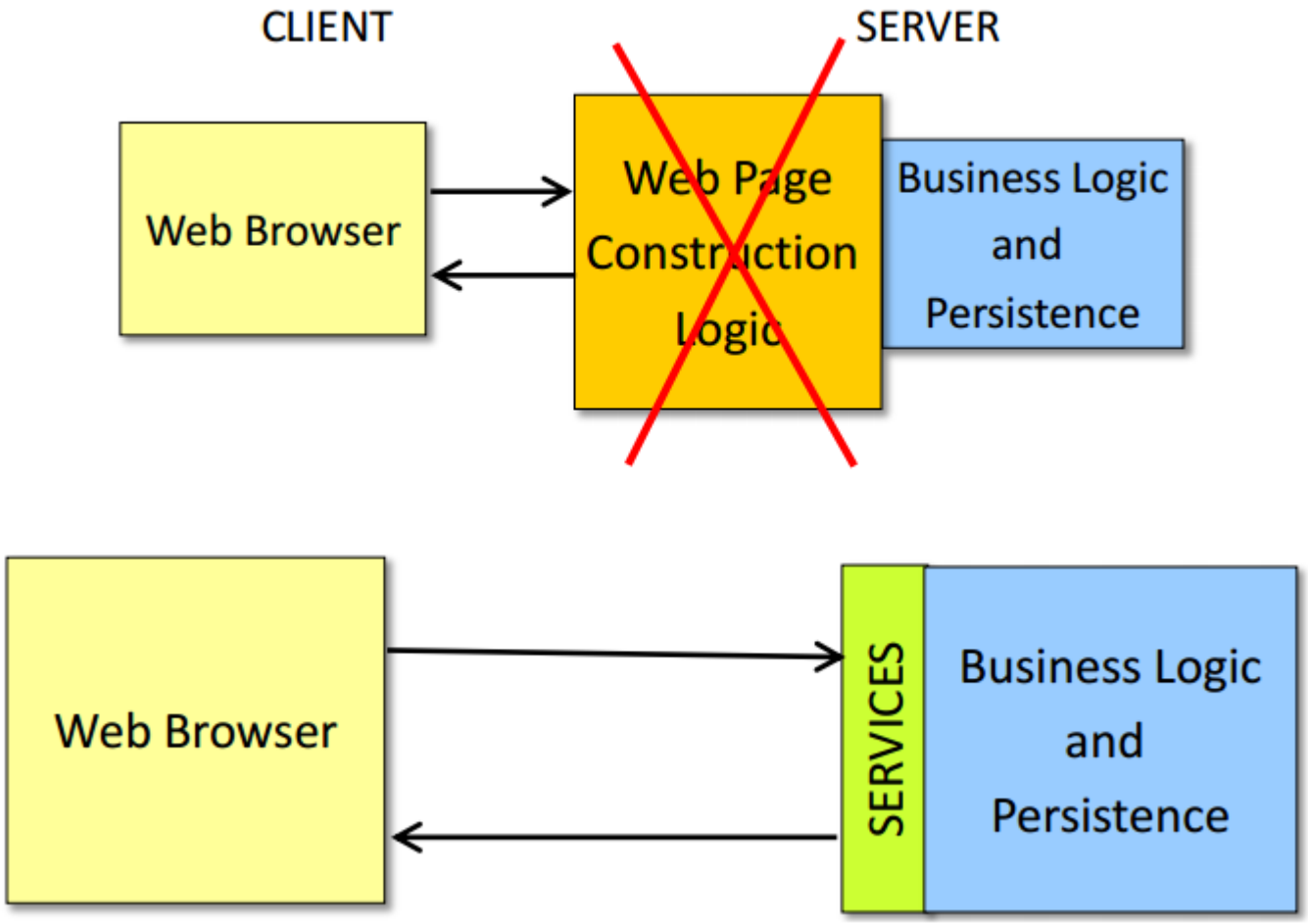
Agenda

- SOFEA
- AngularJS

SOFEA

- **S**ervice **O**riented **F**ront **E**nd **A**rchitecture
- Sinônimo de Thin Server Architecture
- Estilo arquitetural
- Descrito em 2007 por Ganesh Prasad, Rajat Taneja, Vikrant Todankar no artigo “Life above the Service Tier”
- Mas afinal, o que é SOFEA?

O que é SOFEA?


CLIENT
SERVER
Web Browser
Web Page Construction Logic
Business Logic and Persistence
Web Browser
SERVICES
Business Logic and Persistence

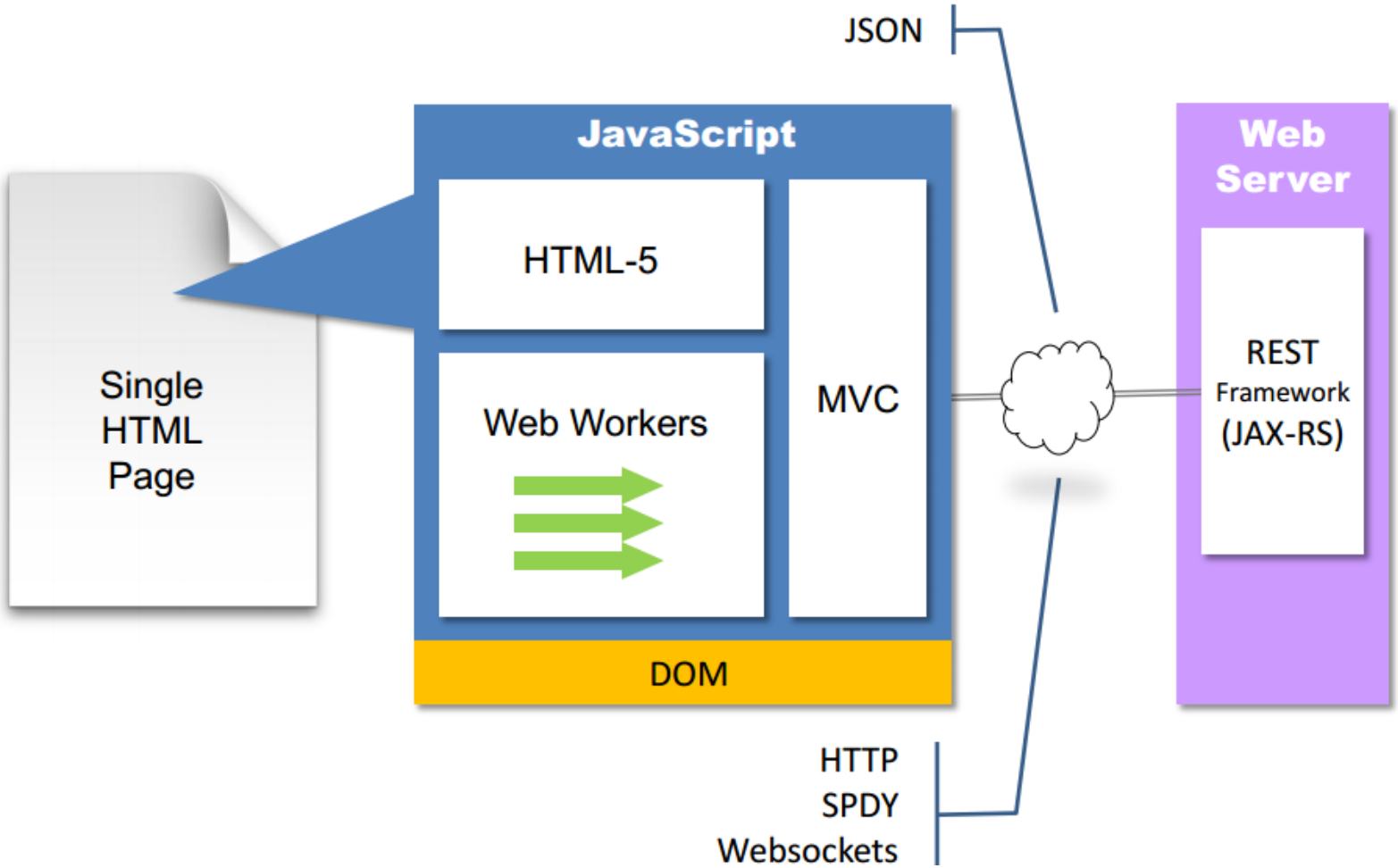
Princípios do SOFEA

- Download da aplicação, troca de dados e fluxo de apresentação devem ser desacoplados do servidor
- Fluxo de apresentação é de responsabilidade do cliente
- Todas as comunicações com o servidor devem ser via web services (REST, SOAP, etc.)
- Componentes do servidor devem focar na lógica do negócio e serem expostos em forma de serviços (SOA)

Benefícios do SOFEA

- Escalabilidade (processing, stateless, caching)
- Interopeabilidade (BaaS - Back-end as a Service)
- Maior ROI por LOC
- Se encaixa em ambientes SOA e Cloud
- Melhor organização e separação da lógica
- Maior responsividade para o usuário
- Desenvolvimento assíncrono do front-end e back-end
- Aplicações offline

Implementação do SOFEA



Migrando do MVC server

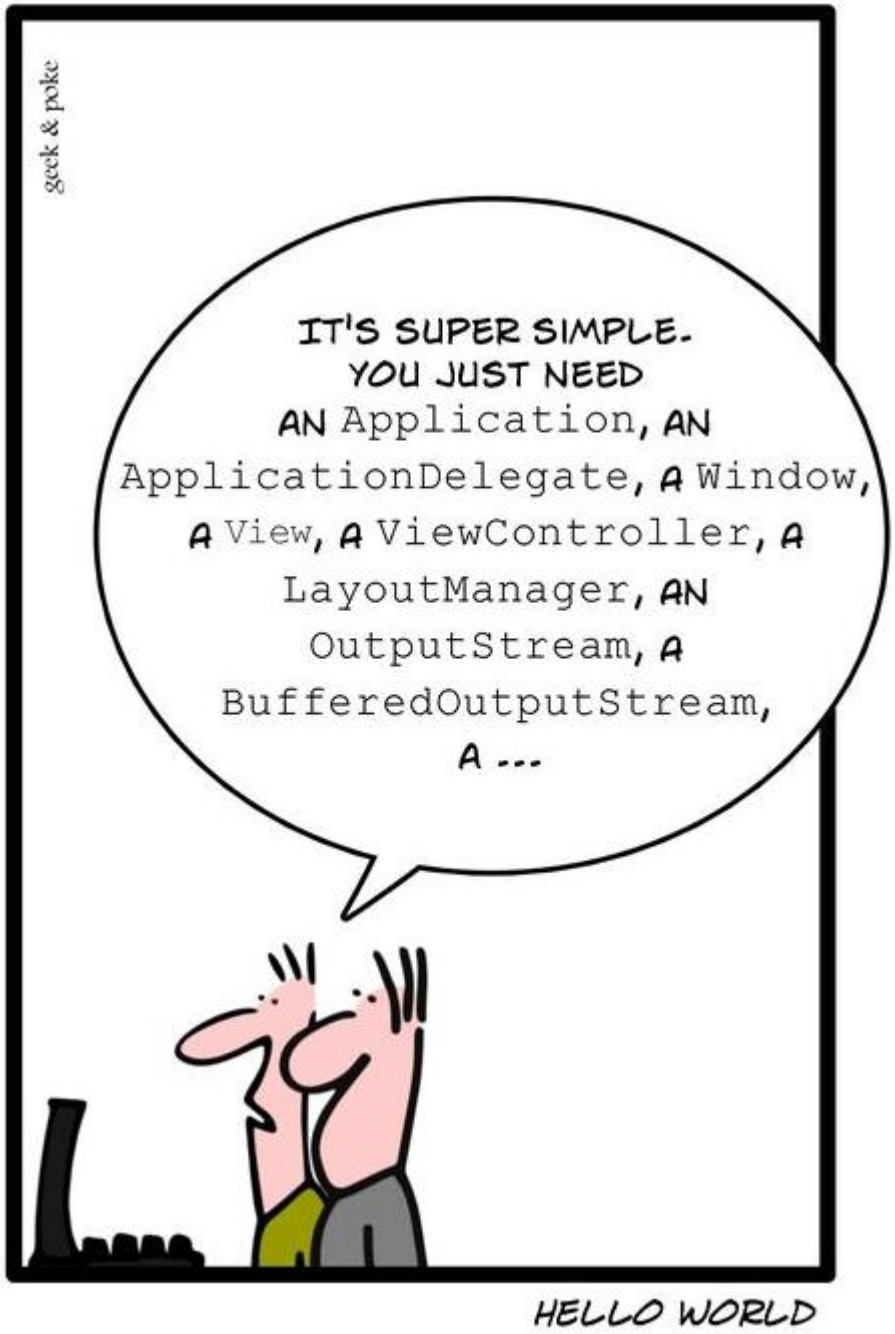
Os frameworks MVC server-side (como JSF) e engines de template (como JSP) são completamente substituídos.

- **Model**: Objetos de sessão passam a ser objetos JavaScript no cliente
- **View**: Widgets e templates são substituídos por HTML e CSS puros
- **Controller**: As classes responsáveis pelo fluxo da apresentação são substituídas por arquivos JS

Por que só agora?

- Redução da complexidade para a criação de web apps
  - Evolução dos web browsers e specs
  - Evolução dos frameworks client-side MVW
  - Simplicidade no cliente (HTML, JS e CSS)
- Heterogeneidade da equipe
- Maturidade do SOA e do Cloud
- Any Device (PC, mobile, embedded, wearables)

Redução da complexidade



SOFEA dentro da equipe

- **Designers**: Criam apenas arquivos HTML, CSS e imagens que serão utilizados diretamente pelo front
- **Front-end developers**: Mantêm apenas arquivos HTML dos designers e criam JS para implementar a UI
- **Back-end developers**: Focam somente na lógica de negócio, na persistência dos dados e nos serviços
- **Benefícios extras**: Front-end developers fazem mock dos serviços criando um contrato para desenvolver de forma assíncrona

AngularJS

- Framework JavaScript MVW* client-side para desenvolver aplicações web modernas e dinâmicas
- A primeira versão open-source foi liberada em 2010 e desde então ele é mantido pelo Google e comunidade
- Aprox. ~2 releases mensais, projeto altamente ativo
- O que faz o AngularJS ser especial?

Diferenciais do AngularJS

- Construído para criar aplicações grandes
  - Organização da bagunça no client-side
  - Reutilização de código e modularidade (DRY)
- Killer Features
  - Dependency Injection, Two-way data binding e Directives
- Integração natural com arquitetura REST, SOA, SOFEA
- Testabilidade

Organizando o código

- Guidelines de desenvolvimento
- HTML views
- Modules
- Controllers
- Services, Providers, Factories, Values e Constants
- Directives e Filters
- Bootstrap Configuration e Decorators

Dependency Injection

- AngularJS foi pioneiro na implementação de DI no JS
- Alta coesão e baixo acoplamento
- Possibilita substituir as implementações e até a criação de mocks para testes (Exemplo: $httpBackend)

Serviços injetáveis

- São singletons instanciados apenas uma vez pela app, somente quando necessários (lazy loaded), que fornecem funções ou mantém o estado de algo
- Tipos: Service, Factory, Provider, Constant e Value
- Podem ser utilizados em qualquer componente gerenciado pelo AngularJS

Exemplo de serviço

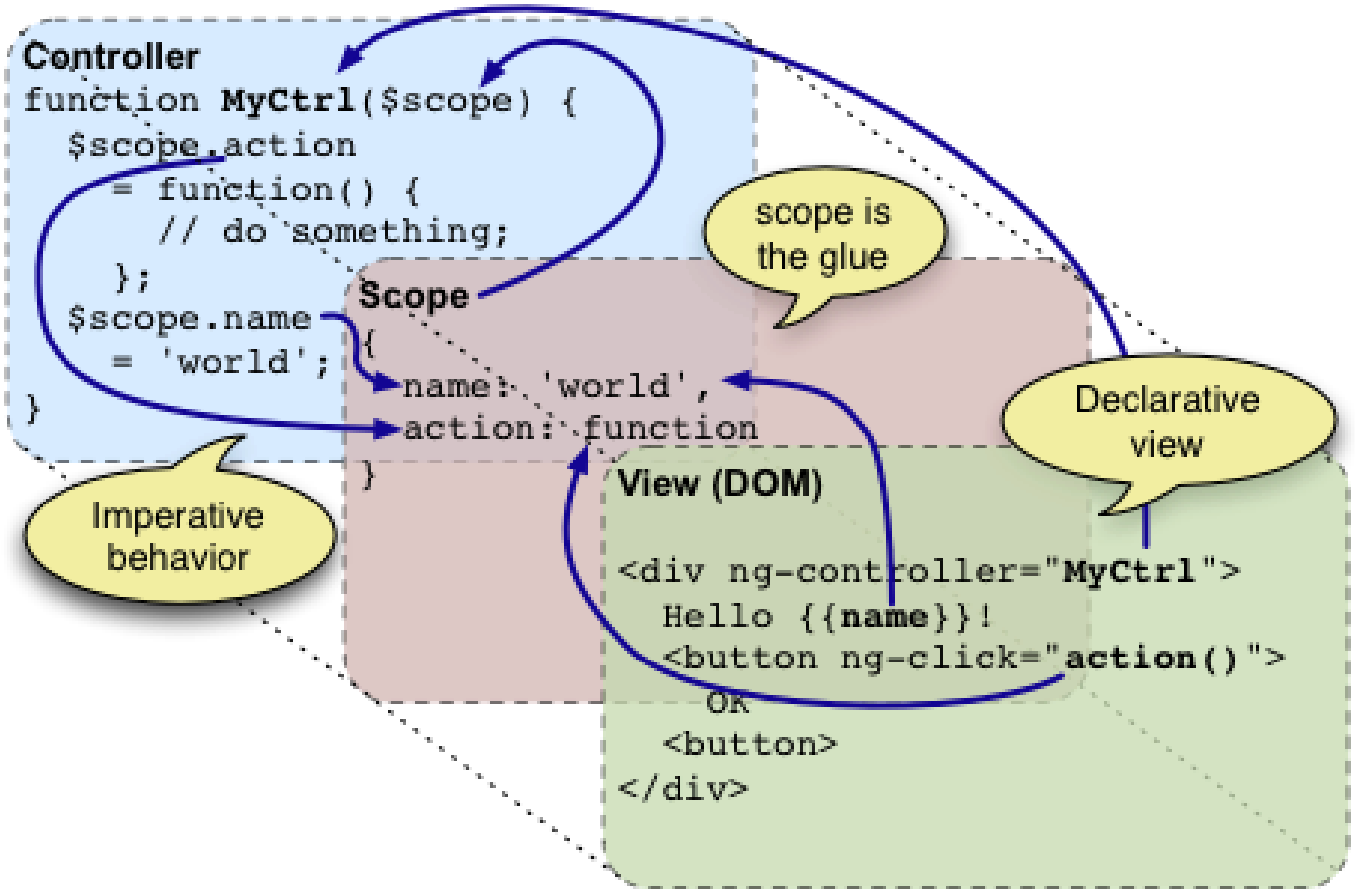
```
var services = angular.module('myApp.services',
    ['ngResources']);

services.factory('AuthService', function($resource) {
    return $resource(auth, {}, {
        authenticate: {
            method: 'POST',
            params: { 'action': 'authenticate' },
            headers: { 'Content-Type':
                       aplication/x-www-form-urlencoded }
        }
    });
});
```

Controllers

- Responsabilidades
  - Inicialização do modelo da aplicação
  - Expôr o modelo e as funções para a view (UI template) utilizando o $scope
  - Observar o modelo por mudanças e executar as atualizações onde for necessário (two-way data binding)
- Devem ser gerenciáveis e testáveis

Two-way data binding



Exemplo de controller

```
function SaudacaoController($scope) {
    $scope.saudacao = 'Olá!';
}

<div ng-controller="SaudacaoController">
    {{ saudacao }}
</div>
```

Directives

- HTML templating engine para criação de componentes
- Extensão do HTML (tags, attr, class)
- Extensão de componentes de terceiros (Ex.: Bootstrap)
- É considerado o recurso mais avançado do AngularJS
- Os componentes resultantes geralmente são reutilizáveis e distribuíveis (Ex.: AngularUI)

Exemplo de directive

```
<div ng-repeat"noticia in noticias">
    <span ng-bind="noticia.titulo"></span>
    <button ng-click="delete($index)">
        Excluir
    </button>
</div>
```

Consumindo Web Services

- $http
- $resource
- Restangular

Testando de ponta a ponta

- Unit (Karma & Jasmine)
- E2E (Protractor & Jasmine + Selenium WebDriver)

Funcionalidades extras

- Promises ($q)
- Event System ($emit, $broadcast)
- Routing (ngRoute, ui-router)
- Caching ($cache)
- Animations ($animate)
- i18n e l10n (ngPluralize, $locale, Translate)
- HTTP Interceptors (auth, redirect, log)

Cuidando da performance

- Cache
- Bind Once
- Fast Watchers
- Invisible Watchers (ngIf vs ngShow)
- Lazy Loading (infinite scroll, paginação)
- Repeat Track By (1.2.x+)
- Model Debounce (1.3.x+)

Futuro do AngularJS

- AngularJS 2.0
  - Mobile-first
  - Escrito em ES6, Annotations
  - Traceur compiler
  - Modularização (DI.js)
- Object.observe (40x faster)


Q&A

Obrigado

@leonardopanga

@mgraciano