

Engenharia de Software

Prof. Vinicius Cardoso Garcia

vcg@cin.ufpe.br :: [@vinicius3w](https://twitter.com/vinicius3w) :: viniciusgarcia.me

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

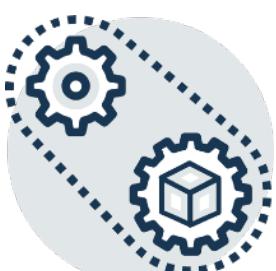
Licença do material

Este Trabalho foi licenciado com uma Licença
Creative Commons - Atribuição-NãoComercial-
Compartilhagual 3.0 Não Adaptada



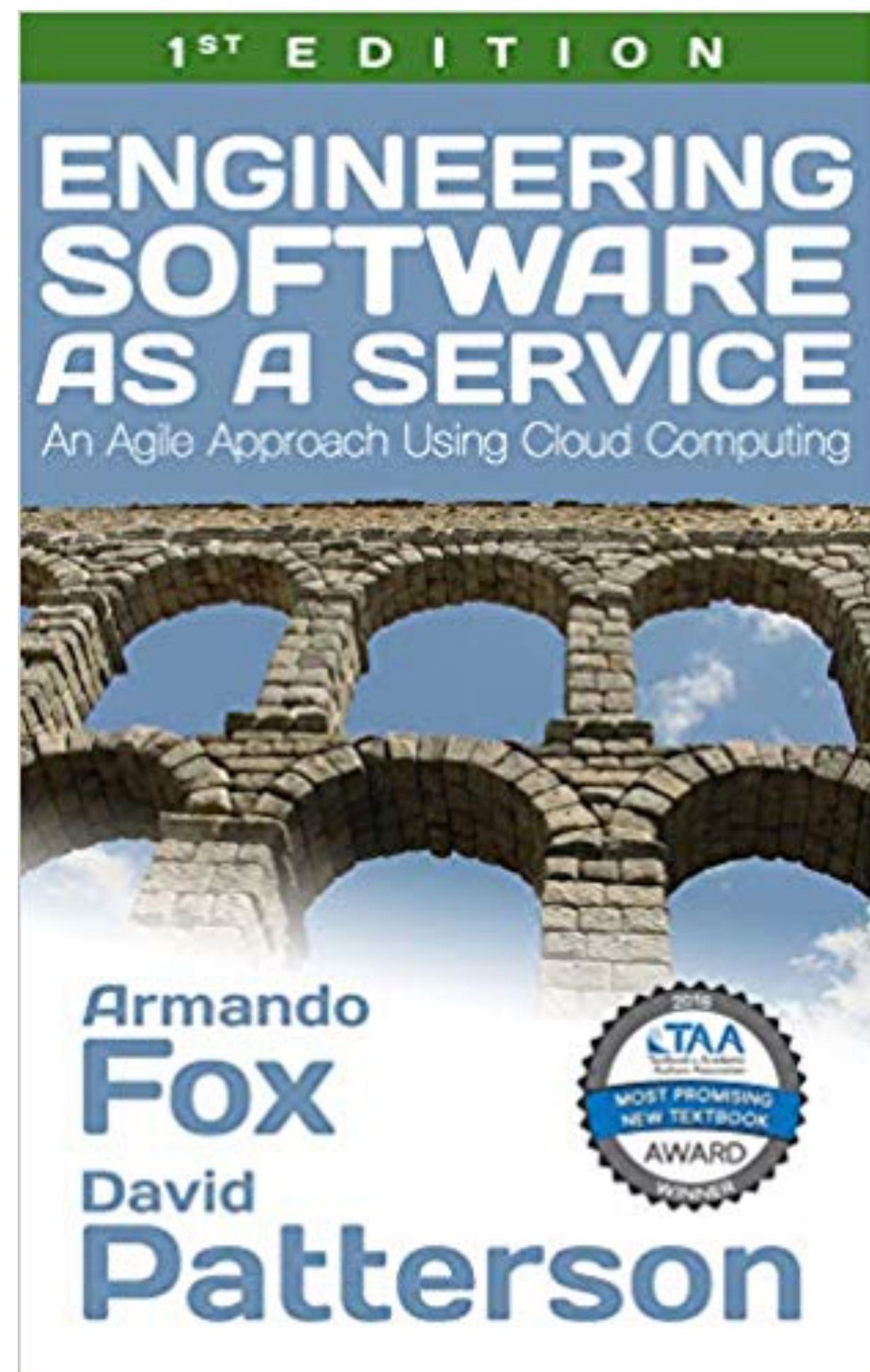
Mais informações visite

[http://creativecommons.org/licenses/by-nc-sa/
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



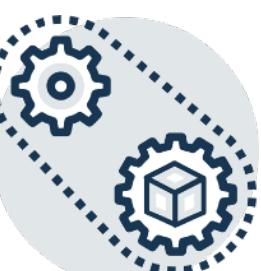
Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
 - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
 - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
 - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
 - <https://engsoftmoderna.info/>



Revisão do que vimos

- Documentação de API's revelam **método**, **caminho** e **argumentos** obrigatórios e opcionais para **cada** chamada da API
- Também a **URI base** que se aplica a **todas** as chamadas; geralmente inclui a versão da API
- A maioria das APIs requer **autenticação**; há várias formas de incluir **credenciais** de autenticação na solicitação
- Os resultados quase sempre são retornados como um único objeto JSON
 - E quanto a vários resultados, por ex. lista de imóveis disponíveis?

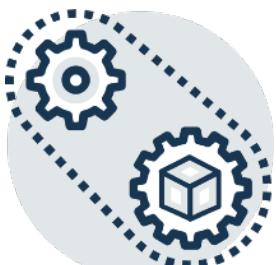


RESTful API

GET PUT POST DELETE

REST

Tudo é um Recurso

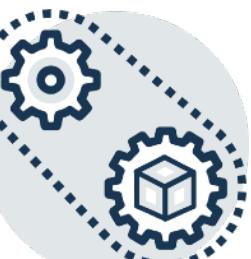


5



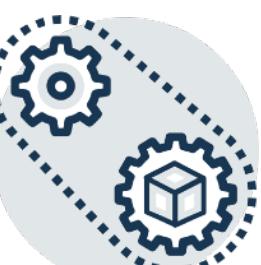
Qual o melhor API framework?

- SOAP? WSDL? UDDI? XML-RPC? CORBA?
- A Internet evolui pelo consenso dos desenvolvedores...
 - fácil para os desenvolvedores entenderem/usarem
 - fácil expressar casos de uso comuns, possível para fazer os raros
 - não é necessária uma licença pesada
 - promove design de código correspondido a padrões da Web
- E o vencedor é... **REST**



REST (Representational State Transfer) – Roy Fielding, 2000

- Route \neq página ou ação, mas recurso & operação
 1. Qual é o principal **recurso** afetado?
 2. Que **operação** deve ser feita e quais são os possíveis resultados e efeitos colaterais?
 3. Quais **outros dados** necessários para a operação e como são especificados?
- Um recurso pode ter **várias** representações possíveis
- As respostas podem incluir **hiperlinks** para recursos adicionais
- Um serviço RESTful ou API é aquele cujas **operações** seguem essas diretrizes



A ideia mais importante na aula de hoje ...



Apps Web como serviços RESTful

- Um aplicativo Web RESTful é um serviço cujos recursos e resultados de operação podem ter várias representações, sendo as mais comuns:
 - JSON (geralmente; raramente XML) para acesso programático
 - HTML para interface humana
- Assim, a questão-chave de projeto para esse aplicativo é: **quais são os recursos, quais são os relacionamentos entre eles e quais operações são permitidas?**
- Muitos frameworks, incluindo o Rails, Spring Boot, AdonisJS, ExpressJS, economizam muito tempo se você pensar cuidadosamente sobre essa decisão.

Operações básicas: CRUDI

GET /properties

```
{"properties":
```

```
[  
  {"id": 253, "title": "Beach Class"},  
  {"id": 254, "title": "Torres do Rosarinho"},  
  {"id": 255, "title": "Rio Tapajós"},  
  ...  
  {"id": 500, "title": "Mont'serrat"}  
]
```



10

Ação	Significado	Método HTTP
Create	Criar nova instância de recurso	POST
Read	Recuperar instância do recurso	GET
Update	Modificar instância de recurso existente em vigor PATCH or PUT	
Delete	Destruir a instância do recurso	DELETE
Index or List	Enumerar (filtrado?) uma coleção de recursos	GET



Exemplos: Possíveis rotas para manipular recursos de Imóveis

R: GET /properties/253

O argumento que identifica o recurso principal geralmente aparece como parte do caminho da URI

I: GET /properties?name=hidden+figures

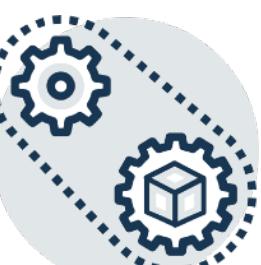
O recurso principal neste caso é o próprio catálogo de filmes; Params adicionais personalizam a operação

C: POST /properties → por quê sem id?

A. ele será **carregado** no corpo da solicitação **POST**, junto com dados sobre o novo filme

B. o **imóvel** ainda não foi criado, então ele não tem um **ID**

C. a rota está **incompleta**: deve ser algo como **POST /movies/:movie_id**



E o Update?

- Ações com **efeitos colaterais** não devem usar GET
- Convenção de Loose: POST **cria** um recurso, **atualizações** PUT ou PATCH, DELETE **exclui**

```
Create: POST /properties
{
  "title": "Holliday",
  "price": 120,
  "adress": "...",
  "latitude": -27.200368
  "longitude": -49.621018
}
```

```
Update: PATCH /properties/253
{
  "price": 145
}
```

- A resposta podem ser **cabeçalhos sem corpo** ou pode **incluir objetos JSON** representando recursos **recém-criados** ou **atualizados** recentemente



Outras operações além de CRUDI?

- Que tal "adicionar imóvel à minha lista de observação"?
- **Dica:** Quando parecer estranho criar uma operação em termos de recursos, se pergunte se existe outro tipo de recurso esperando para ser definido e qual relacionamento ele tem com o(s) recurso(s) existente(s)
- Às vezes, isso resultará em um projeto mais simples: novo recurso, mas somente o CRUDI o utiliza



Discussão

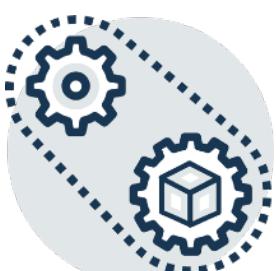
- Como seria uma rota RESTful para modelar a operação de um usuário adicionando um imóvel à lista de observação?
- Lembre-se das 3 questões do REST:
 1. Recurso primário?
 2. Operação e efeitos colaterais?
 3. Outros dados necessários?

A estrutura do URI sugere recurso "aninhado": uma lista de observação pertence a um determinado usuário

PUT /users/:user_id/watchlist/:property_id

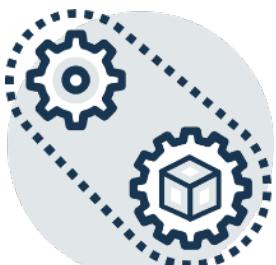
PUT /users/:user_id/watchlist?property_id=xxx

...ou PATCH ao invés de PUT



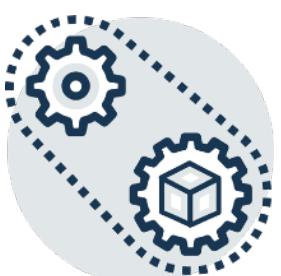
Discussão

- Quais outras especificações possíveis para a rota “adicionar imóvel à minha lista de observação” são um projeto RESTful razoável? (Dica: 3 perguntas REST)
 - A. PUT /watchlist/:watchlist_id?property_id=xxx
 - B. GET /watchlist/:watchlist_id?property_id=xxx
 - C. PUT /:watchlist_id/:property_id
 - D. PUT /user/:user_id?watchlist_id=yy&property_id=xx



Sumário: RESTful APIs (Representational State Transfer)

- Tudo é um recurso.
 - Operações básicas: criar, ler, atualizar, excluir, listar
 - Inábil para expressar a operação desejada? → talvez novos recursos precisem ser definidos
- A operação RESTful identifica o recurso, op para ser feita/efeitos colaterais e quaisquer outros dados necessários para concluir a operação
 - relação de recursos (por exemplo, A "possui" B), muitas vezes refletida na rota, por ex. [GET /a/:a_id/b/:b_id](#), mesmo que b_id identifique exclusivamente o recurso
- Recurso pode ter diferentes representações
 - Página HTML exibível para usuário humano
 - Estrutura de dados JSON ou XML
 - portanto, representacional em REST
- REST "ganhou" porque é simples e corresponde a restrições de engenharia de como a Web evoluiu, por exemplo, [HTTP sem estado](#)



Experimentando com APIs?

- Manualmente
 - Extensão do navegador [JSONview](#)
 - `curl` (padrão no Linux, MacOS; `curl.haxx.se` no Windows)
- O "API explorer" incorporado em muitos documentos de API permite que você tente chamadas ao vivo preenchendo valores de argumentos
 - [Swagger](#)
- Tópico suplementar/futuro: [OpenAPI](#)
- [Postman](#) (getpostman.com), uma ferramenta para experimentar e projetar APIs
 - [Insomnia](#): <https://insomnia.rest>

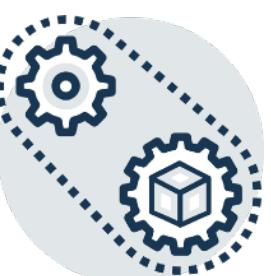


```
</form>
top:0px;
<br>
<div>
position: absolute;
color:#999;</h2>
<p> margin:0px; </html>
</span> text-align:center; <h2>
<td> <!DOCTYPE html>
font-style
<html> </div> margin:4px;
<head> </div>
<body> <tr><img>
<div> <span> <h1><body><ul>
```

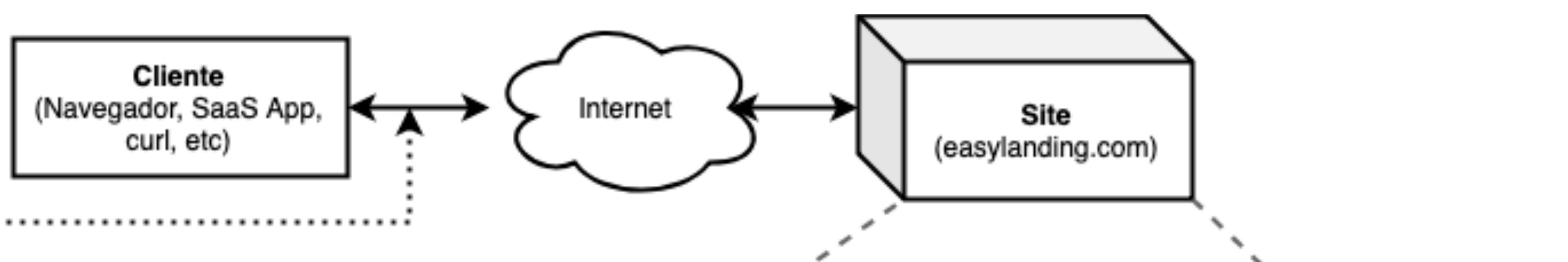
HTML & CSS



HTML & CSS

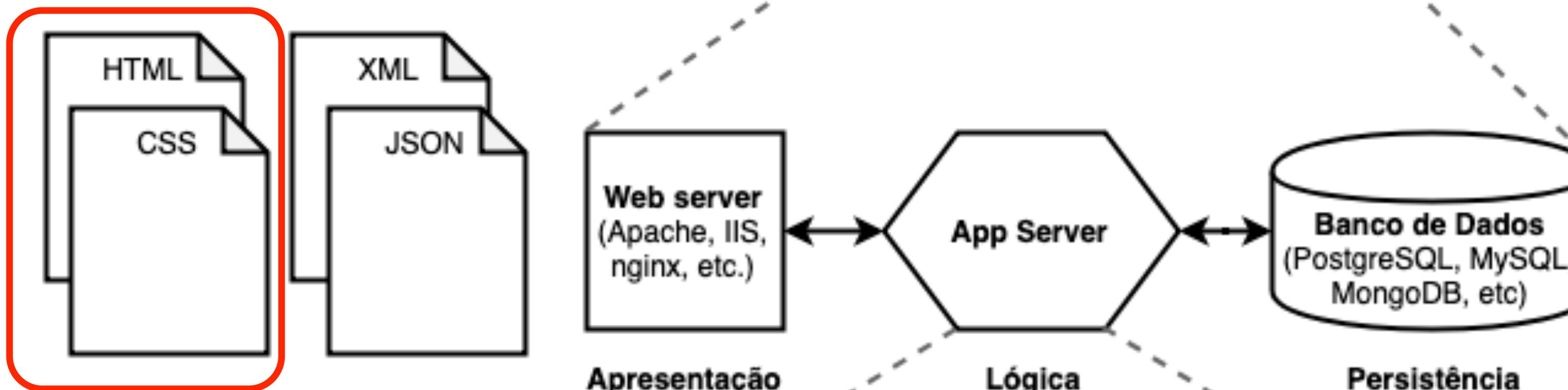


Cliente-Servidor vs P2P



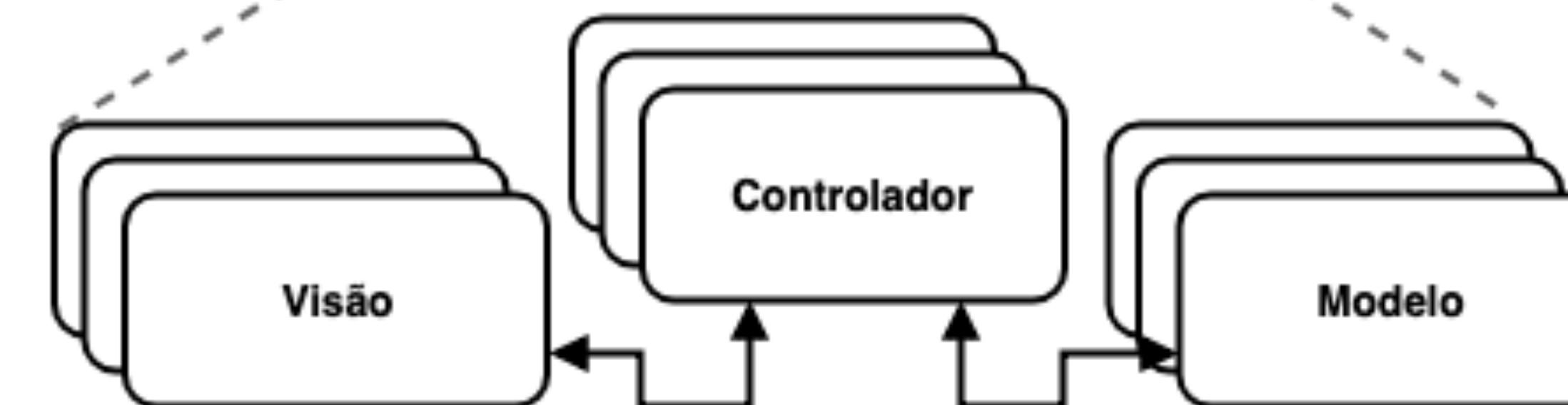
Arquitetura 3 camadas

Escalabilidade



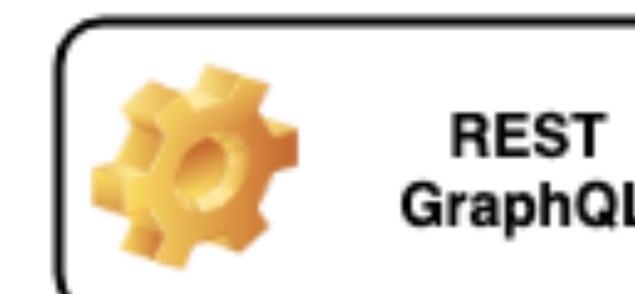
Padrões Arquiteturais

Model-View- Controller

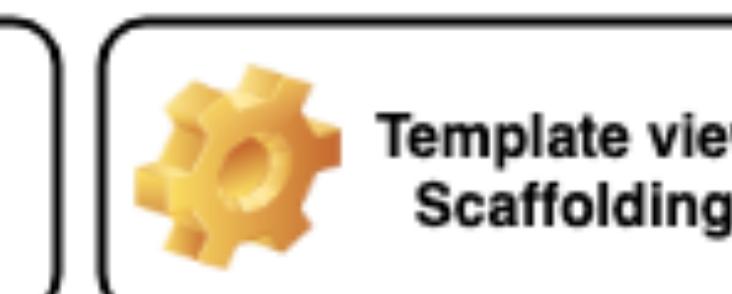


Padrões de Projeto

Idiomas



REST
GraphQL



Template view
Scaffolding

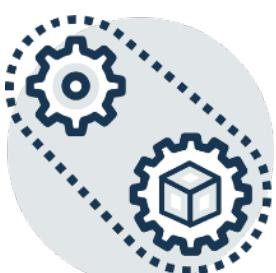


ORM
Active Record
Data Mapper



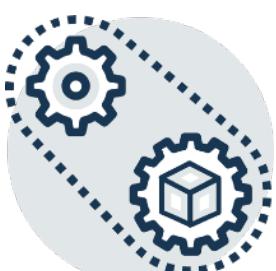
HTML & CSS

- Não objetivo: orientá-lo em HTML e CSS
- Objetivo: tratar os 20% dos conceitos que ajudarão você a entender 80% dos tutoriais rapidamente
- Não objetivo: torná-lo especialista em design de front-end
- Objetivo: evitar que suas páginas pareçam ter sido criadas por vikings bárbaros



Fundamentos conceituais

- HyperText Markup Language (HTML) == hierarquia de elementos aninhados
 - texto, imagens, formulários de preenchimento, divisões estruturais
 - cada elemento pode ter um ID (exclusivo dentro do documento) e zero ou mais classes
- Cascading Style Sheets (CSS) associam atributos visuais a elementos específicos
 - geralmente com base no ID e/ou classe do elemento
- **Bootstrap** é um conjunto de estilos visuais limpos e bem-elaborados que você pode usar como ponto de partida
 - Centenas de "modelos de página" do Bootstrap disponíveis também



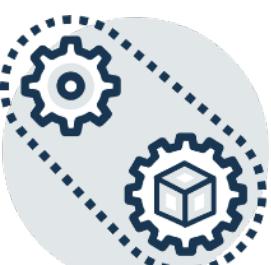
Introduction

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes three hard-to-swallow claims about penguins:

First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish

Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

**Recommended: Mozilla Foundation
HTML tutorial (under Resources on
course home page)**



<h1>Introduction</h1>

<p>

This article is a review of the book

<i>Dietary Preferences of Penguins</i>,

by Alice Jones and Bill Smith. Jones and Smith's controversial work makes three hard-to-swallow claims about penguins:

</p>

First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish

Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

Introduction

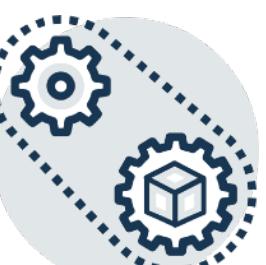
This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes two hard-to-swallow claims about penguins:

- First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish
- Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

```
<h1>Introduction</h1>
<p>
  This article is a review of the book
  <i>Dietary Preferences of Penguins</i>,
  by Alice Jones and Bill Smith. Jones
  and Smith's controversial work makes
  three hard-to-swallow claims about
  penguins:
<ul>
<li>
  First, ...

```



Hypertext Markup Language

- Documento = Hierarquia de elementos delimitados por `<tag>....</tag>`
 - texto (títulos, tabelas, listas, parágrafos)
 - mídia incorporada (imagens, JavaScript)
 - formulários (caixa de texto, botões de rádio/verificação, menus ...)

`<div>`

Bloco de coisas, possivelmente
incluso `elementos incorporados`

`</div>`

- Elementos
 - Pode ter conteúdo: `<p>Bora BAÊA!</p>`
 - E/ou atributos opcionais/obrigatórios:
``



Cascading Style Sheets (CSS)

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="http://.../style.css"/>
  </head>
  <body>
    ...main page content here...
  </body>
</html>
```

- Atributos mais importantes: `id` & `class`

- `id` deve ser único na página
- Uma mesma `class` pode ser anexada a muitos elementos

```
<div class="row">
  <div class="col-md-3 offset-md-3 text-center">
    Eu sou Vinicius. Eu leciono a IF977 e faço pesquisa...
  </div>
</div>
```



Bootstrap



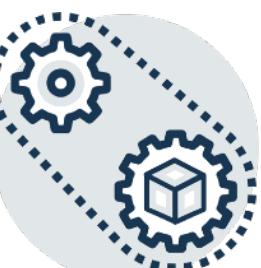
BootstrapCDN

When you only need to include Bootstrap's compiled CSS or JS, you can use [BootstrapCDN](#).

CSS only

```
<link rel="stylesheet" href="https://st...
```

- Estilos CSS personalizáveis de código aberto e comportamentos JavaScript para estilo de página
 - Fácil: baixe diretamente dos próprios servidores do [Bootstrap](#), substitua/adicione alguns estilos, conforme necessário
- Os principais recursos do Bootstrap:
 1. Componentes de nível superior: “Jumbotron”, “card”, etc., estilizando elementos HTML subjacentes ([`<p>`](#), [`<div>`](#), etc.)
 2. Layout de grade: até 12 colunas por linha
 3. Responsivo: comportamento razoável em vários tamanhos de exibição
- Difícil de fazer um layout **realmente ruim** com o Bootstrap.



Seletores CSS identificam elementos específicos para estilo

- tag name: h1
- class name: .pageFrame
- element ID: #pageHead
- tag name & class: div.pageFrame
- tag name & id: img#welcome (**normalmente redundante**)
- descendant relationship: div.custName
- Atributos herdam padrões do browser a menos que sejam substituídos
- Objetivo: **Marcações HTML não contém informações visuais de estilo**

```
<div class="pageFrame" id="pageHead">
  <h1>
    Welcome,
    <span id="custName">Vinicius</span>
  </h1>
  
</div>
```

