

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



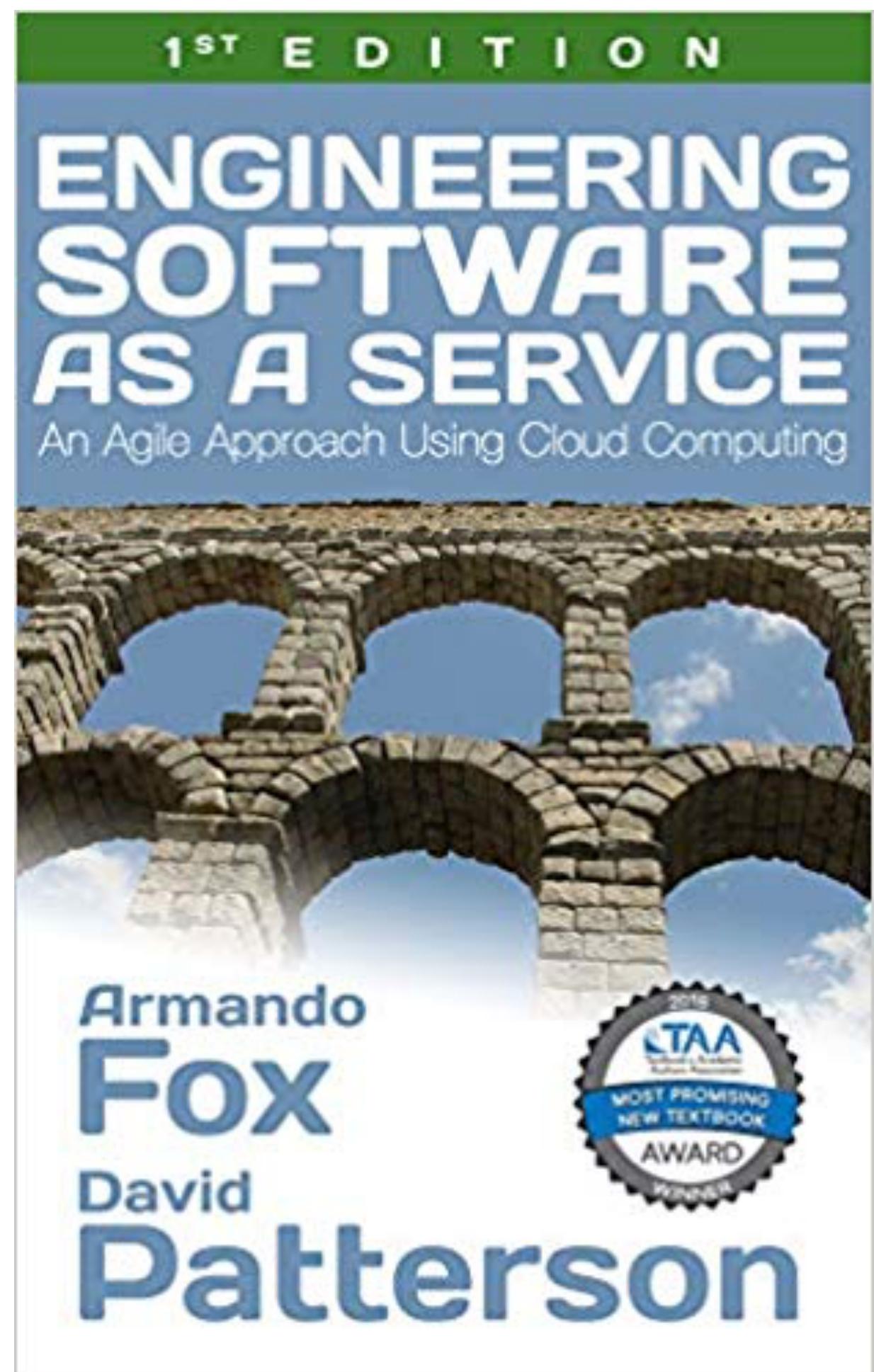
Mais informações visite

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



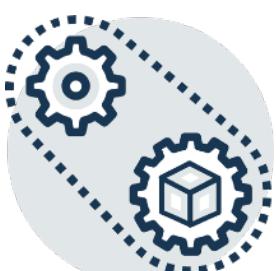
Referências

- 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/>





Models, Databases and Patterns

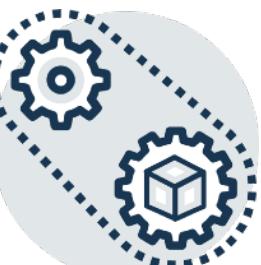


4

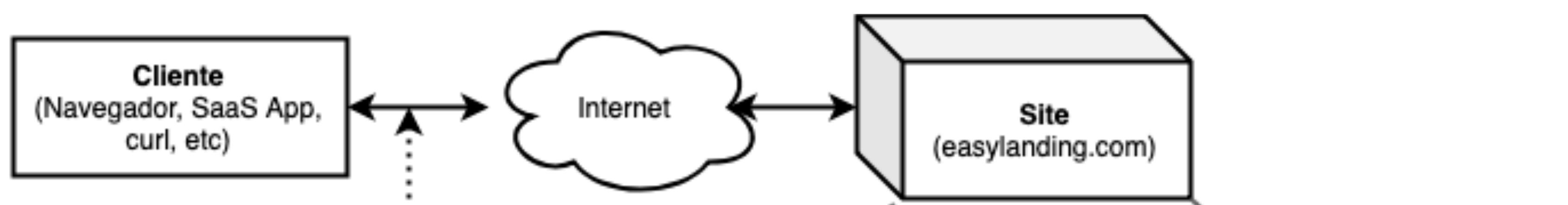


Warm up

- Como deveríamos armazenar e recuperar estruturas de dados orientadas a registros?
- Qual a relação entre dado armazenado e dado manipulado em uma linguagem de programação?

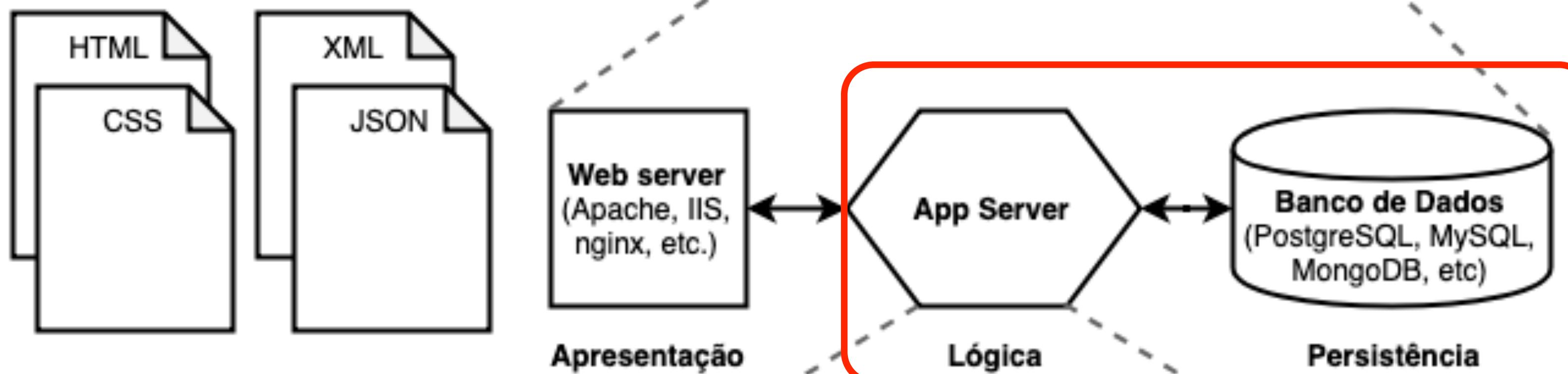


Cliente-Servidor vs P2P



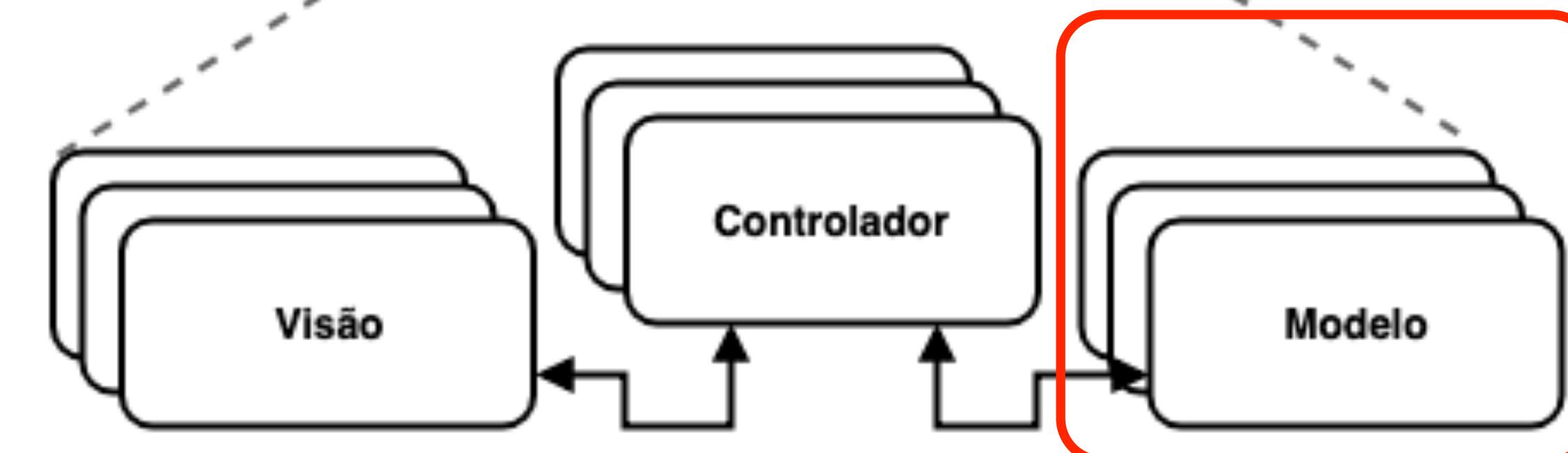
Arquitetura 3 camadas

Escalabilidade



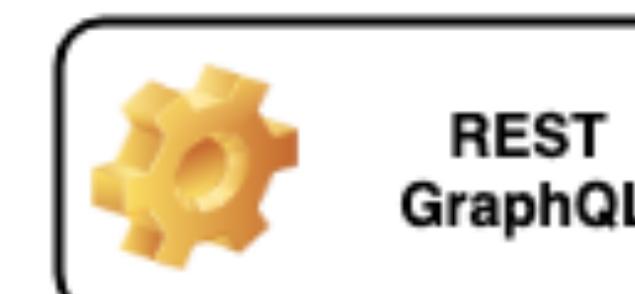
Padrões Arquiteturais

Model-View- Controller

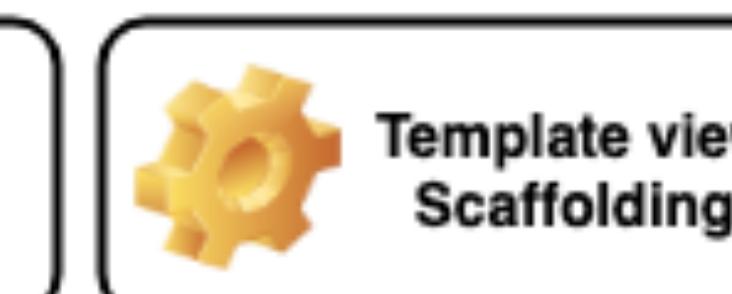


Padrões de Projeto

Idiomas



REST
GraphQL



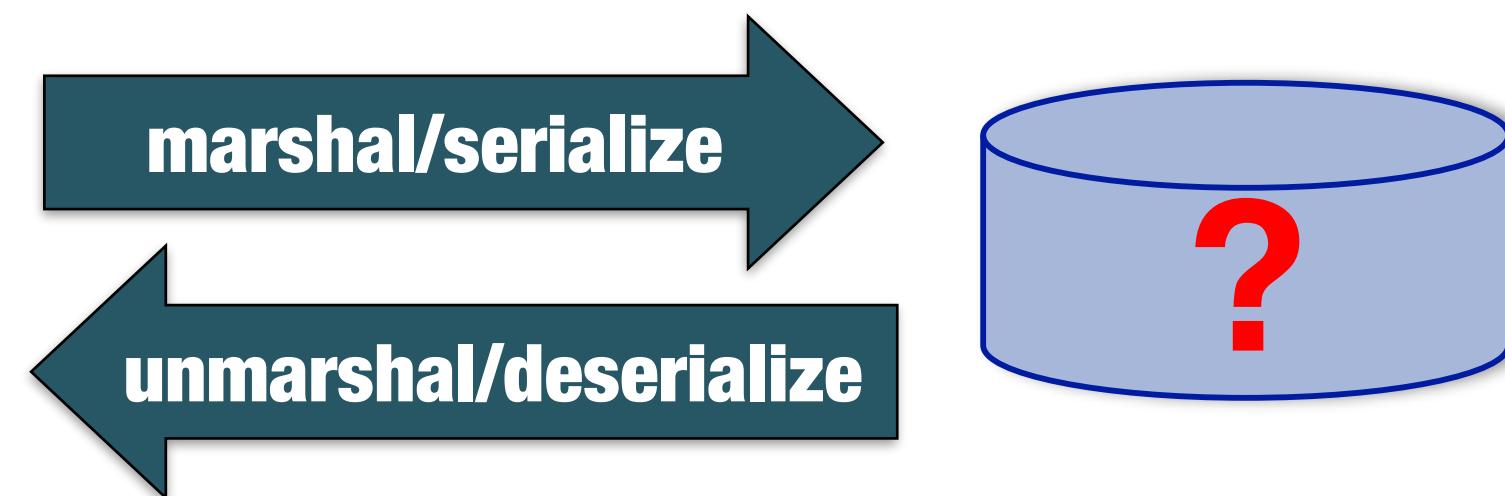
Template view
Scaffolding



ORM
Active Record
Data Mapper

In-Memory vs. In-Storage objects

```
#<Property:0x1295580>  
p.title, p.price, ...  
#<Property:0x32ffe416>  
p.title, p.price, ...
```



- Como representar **objetos persistidos** em armazenamento
 - Exemplo: Imóveis e as suas Fotos
- Operações básicas no objeto: **CRUDI** (Create, Read, Update, Delete, Index)
- **ActiveRecord**: todo modelo conhece como fazer o CRUD em si mesmo, utilizando mecanismos comuns
 - [Lucid](#) é a implementação do ActiveRecord do AdonisJS



Modelos ORM Armazenam Dados em RDBMS

- Cada tipo de model recupera sua própria tabela no BD
 - Todas as linhas em uma tabela possuem estrutura idêntica
 - 1 linha na tabela == uma instância do modelo
 - Cada coluna armazena o valor de um atributo do modelo
 - Cada linha possui um valor único para primary key (por convenção, em ORM é um inteiro chamado ID)

id	title	price	address
1	Beach Class	330	Av. Conselheiro Aguiar...
12	Hollyday	80	Av. Conselheiro Aguiar...
...
42	Flat na Praia	420	Oca Residende...

- Schema: Coleção de todas as tabelas e suas estruturas

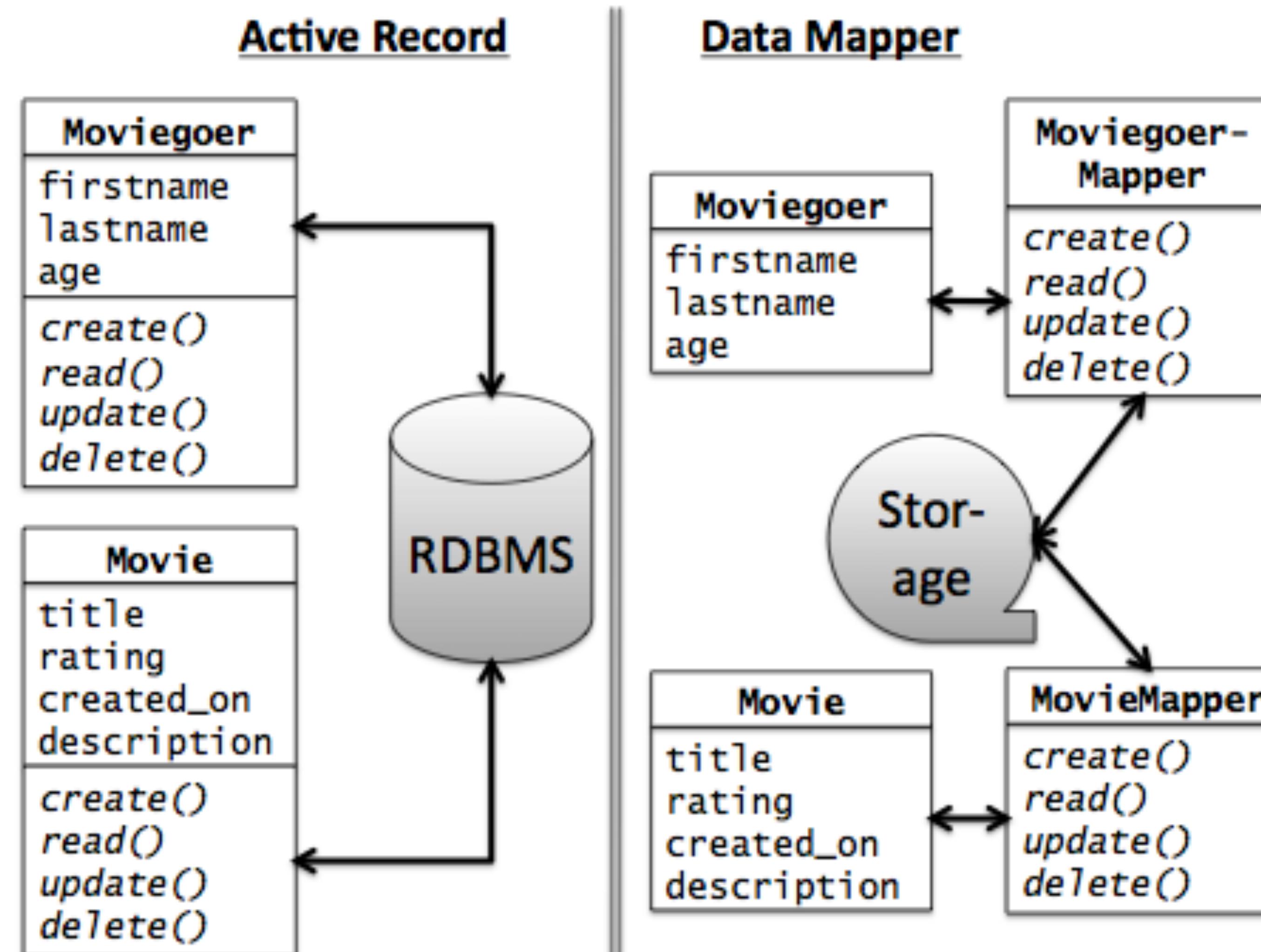


Alternativa: DataMapper

- **Data Mapper** associa mapeamentos separados para cada modelo
 - Ideia: manter mapeamentos **independentes** de armazenamentos de dados => trabalha melhor com mais tipos de BDs
 - Usado pelo **Google AppEngine**
 - **Cons:** não pode explorar os recursos de RDBMS para simplificar as consultas e relações complexas

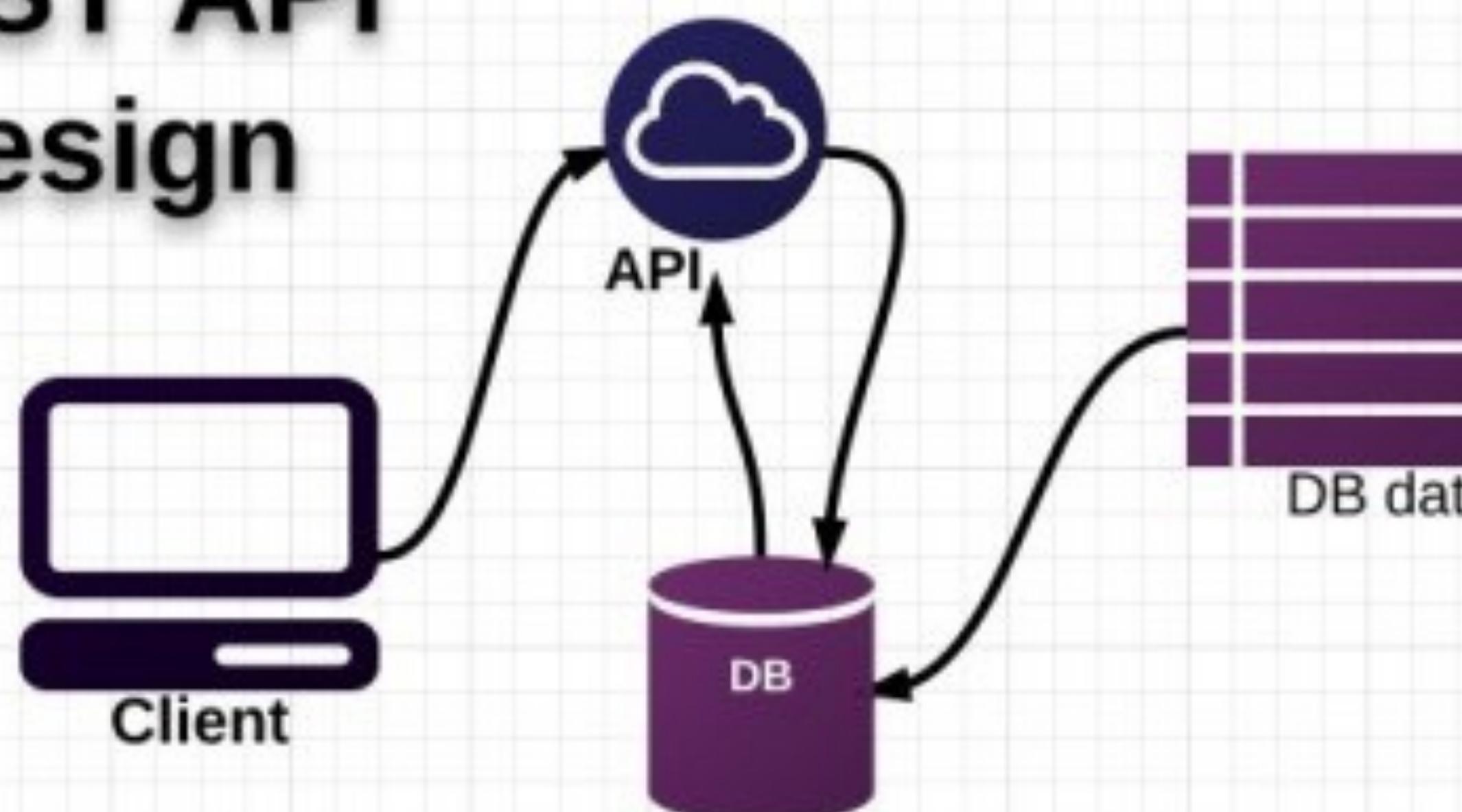


Active Record vs Data Mapper

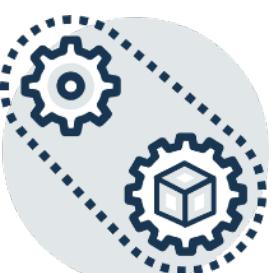


REST API Design

GET	/tasks - display all tasks
POST	/tasks - create a new task
GET	/tasks/{id} - display a task by ID
PUT	/tasks/{id} - update a task by ID
DELETE	/tasks/{id} - delete a task by ID



Controllers, Routes, and RESTfulness

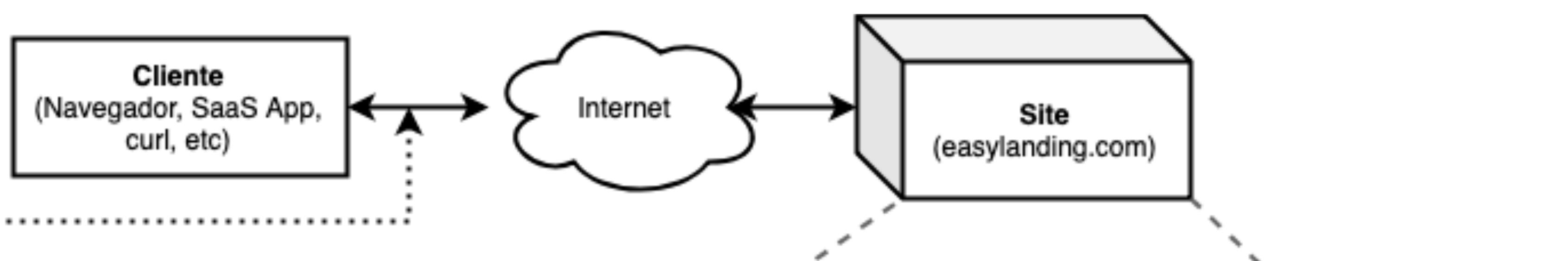


Warm up

- Quais decisões de projeto vão permitir que nosso aplicativo dê suporte a Arquitetura Orientada a Serviço?

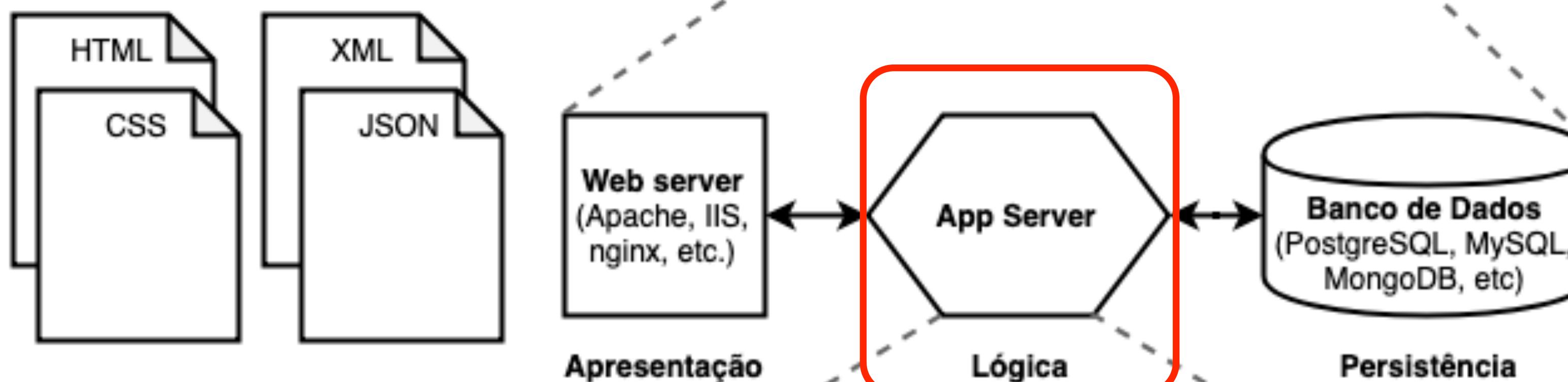


Cliente-Servidor vs P2P



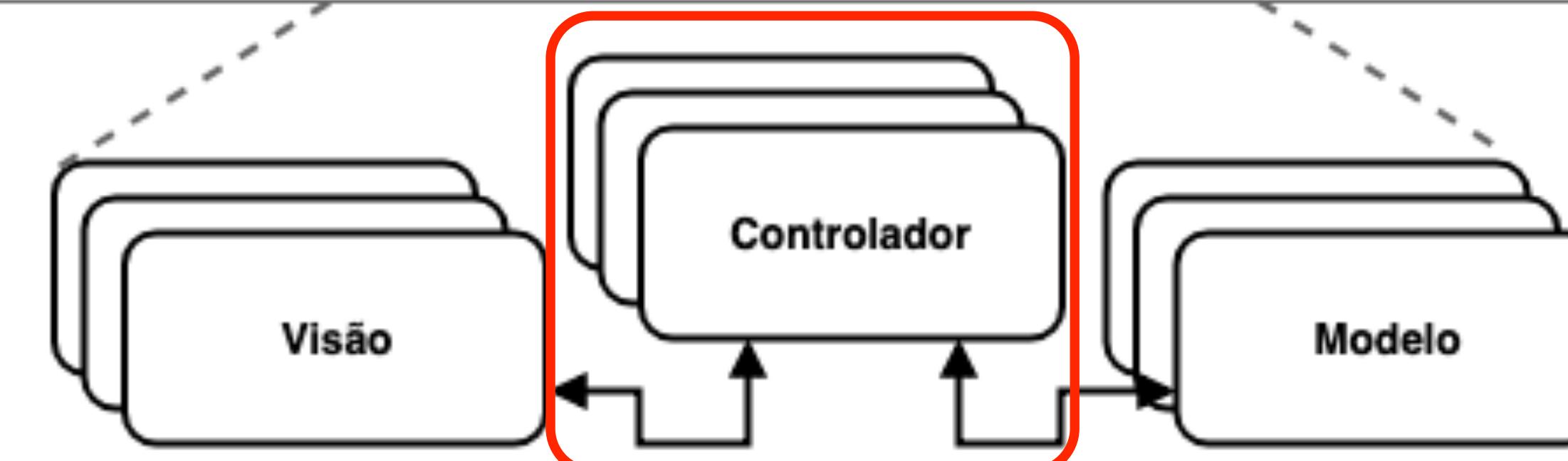
Arquitetura 3 camadas

Escalabilidade



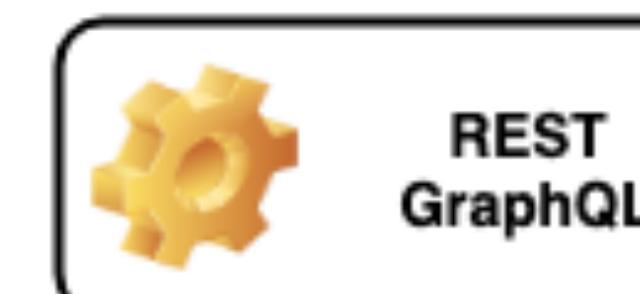
Padrões Arquiteturais

Model-View- Controller

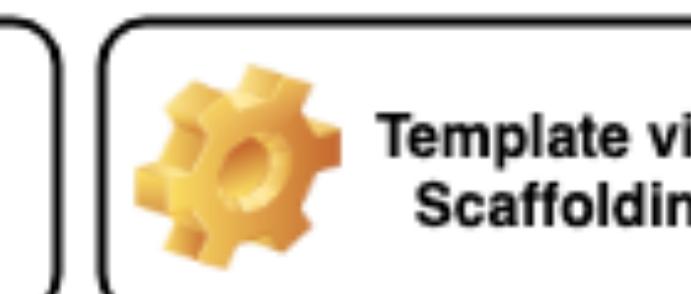


Padrões de Projeto

Idiomas



REST
GraphQL



Template view
Scaffolding

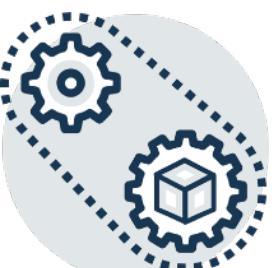


ORM
Active Record
Data Mapper



REST (Representational State Transfer) – R. Fielding, 2000

- Ideia: **Requisições independentes** especificam em qual recurso operar e o que fazer com ele
 - Tese de PhD de Roy Fielding
 - Wikipedia: “a post hoc description of the features that made the Web successful”
- Um **serviço** (no contexto de SOA) cujo as **operações** são serviços RESTFul
- Idealmente, RESTFul URI nomeia as operações



Rotas

- No MVC, cada interação que o usuário pode fazer é tratada por uma ação de um controlador (controller action)
 - Os frameworks possuem métodos que tratam esta interação
- Uma rota mapeia <HTTP method, URI> para uma ação de um controlador

Rota	Ação
GET /property/3	Apresenta as informações do imóvel cujo ID é 3
POST /property	Cria um novo imóvel com as informação incorporadas na requisição
PUT /property/7	Atualiza imóvel cujo ID é 7 com as informação incorporadas na requisição
DELETE /property/5	Remove o imóvel cujo ID é 5



Breve revisão ao subsistema de rotas

- dispara <method,URI> para a ação correta do controlador
- provê métodos auxiliares que geram um par <method,URI> dada uma ação do controlador
- analisa os parâmetros da consulta de ambas URI e cria a submissão em um hash
- atalhos embutidos para geração de todas as rotas de CRUDI (embora a maioria dos apps também tenham outras rotas)

I	GET /property	{PropertyController.index}
C	POST /property	{PropertyController.create}
	GET /property/new	{PropertyController.new}
	GET /property/:id/edit	{PropertyController.edit}
R	GET /property/:id	{PropertyController.show}
U	PUT /property/:id	{PropertyController.update}
D	DELETE /property:id	{PropertyController.destroy}

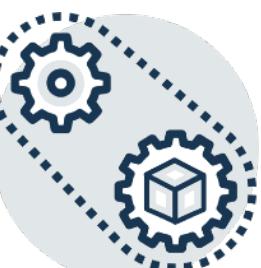


GET /movies/3/edit HTTP/1.0

- Encontre rota:
 - GET /property/:id/edit {PropertyController.edit}
- Analise parâmetros coringas (wildcard): params[:id] = "3"
- Envie para o método `edit` em `movies_controller.rb`
- Para incluir a URI na visão gerada que irá submeter o formulário para a ação de atualização no controlador com `params[:id]=3`, `PUT /movies/3`

I	GET /property
C	POST /property
	GET /property/new
	GET /property/:id/edit
R	GET /property/:id
U	PUT /property/:id
D	DELETE /property:id

	{PropertyController.index}
	{PropertyController.create}
	{PropertyController.new}
	{PropertyController.edit}
	{PropertyController.show}
	{PropertyController.update}
	{PropertyController.destroy}

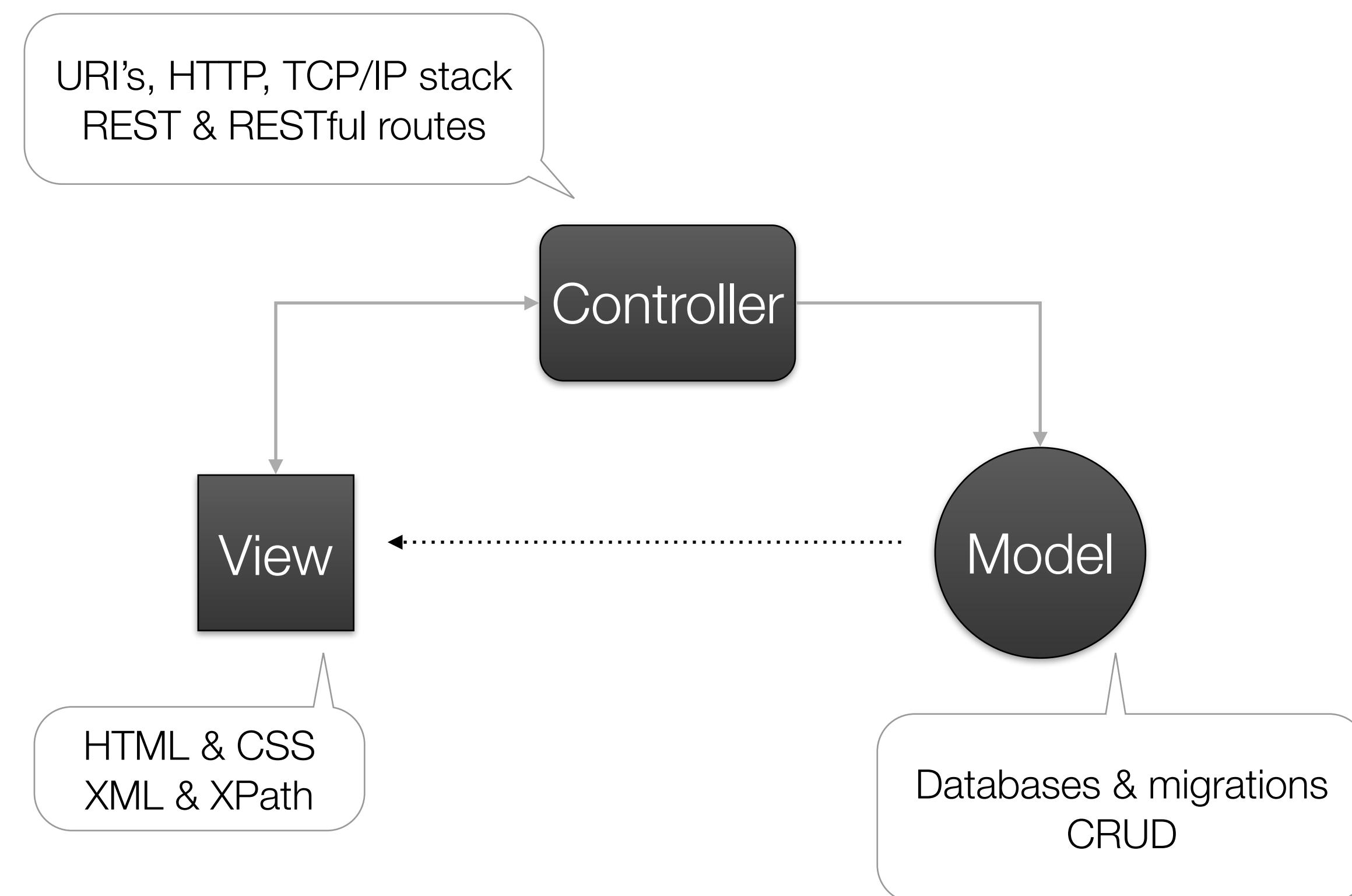




Conclusão e Reflexões: Arquitetura SaaS



The Big Picture



A aplicação não escala

- Escalabilidade é uma preocupação arquitetural – não está confinada a uma linguagem ou framework
- As camadas stateless da arquitetura 3-tier escalam
 - Com computação em nuvem, só precisa se preocupar com as constantes
- RDBMS tradicionais não escalam
- Várias soluções combinam armazenamento relacional e não-relacional (NoSQL) escalam melhor
 - Data Mapper trabalha bem com elas
- Uso inteligente de caching podem melhorar consideravelmente os fatores das constantes



Frameworks, Apps, Design patterns

- Muitos padrões de projeto até o momento, muito mais ainda estão por vir
- Em 1995, era o “velho oeste”: os grandes Web sites eram minicomputadores !!!
 - Nada de 3-tier ou cloud
- Boas práticas (padrões) foram “extraídas” da experiência e capturadas em frameworks
- Porém APIs transcendem isso: 1969 protocolos + 1960s linguagem de marcação + 1990 browser + 1992 Web server -> funciona hoje em dia!



Arquitetura é sobre alternativas

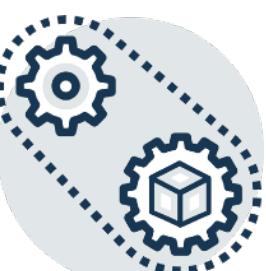
Pattern we're using	Alternatives
Client-Server	Peer-to-Peer
Shared-nothing (cloud computing)	Symmetric multiprocessor, shared global address space
Model-View-Controller	Page controller, Front controller, Template view
Active Record	Data Mapper
RESTful URIs (all state affecting request is explicit)	Same URI does different things depending on internal state

Como você irá trabalhar em outros aplicativos SaaS para além deste curso, você pode se encontrar considerando diferentes opções de arquitetura e questionar as escolhas que estão sendo feitas.

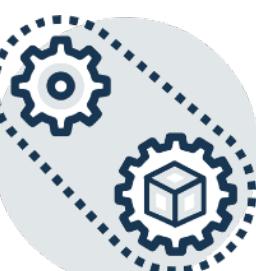


Conclusões: Arquitetura & Node*

- Model-View-Controller é um **padrão arquitetural** bem conhecido para estruturar apps, podemos utilizar ele nos projetos
- AdonisJs **codifica** a estrutura de apps SaaS como MVC
 - Zend (PHP), Django (Python), Grails (Groovy), Play! (java)...
- Express **permite codificar** a estrutura de apps SaaS como MVC
 - Views são interfaces com código embarcado, transformado em HTML quando enviado ao browser (vamos ver isso mais à frente)
 - Models são armazenados em tabelas de um RDBMS, acessados utilizando **Active Record** (ou **Data Mapper**)
 - Controllers **unem views (ou APIs) e models** através de rotas e código nos métodos controladores

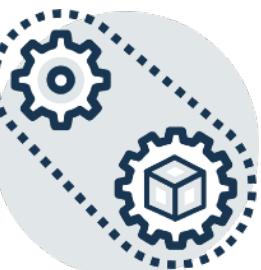


B of A check, c.1950 O que está faltando?



ERMA (Electronic Recording Machine, Accounting)

- 1950: B of A pays SRI International to research problem, propose design
- 1/28/1952: demo system contract signed
- 1956: General Electric contract for production
- 1959: first production system (built by GE) deployed, handles 33K accounts/hour; 31 more subsequently built
- Feb 23, 1962: all accounts converted to ERMA
- 1967: ERMA replaced by IBM 360



ERMA legacies

- MICR
 - Goal: machine-readable, robust to obfuscation, human-readable as backup
 - Printed with iron-oxide-impregnated ink, read by cassette-recorder-like heads
 - As of 2016, still required on checks that are automatically processed
- Credit cards tied to your bank account
 - BankAmericard
- Two surviving ERMA installations...

1 2 3 4 5 6 7 8 9 0



To learn more

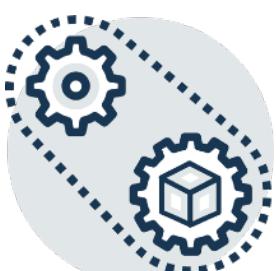
More technical details →
[http://techcrunch.com/
2009/03/01/information-
technology-50-years-ago/](http://techcrunch.com/2009/03/01/information-technology-50-years-ago/)



← Seeing ERMA in Concord:
[http://about.bankofamerica.com/
en-us/our-story/heritage-center-
locations.html](http://about.bankofamerica.com/en-us/our-story/heritage-center-locations.html)



Introdução a Behavior-Driven Design & User Stories



Revisão: O Agile é bem-sucedido quando...

- Trabalhar **perto, continuamente** com as partes interessadas **durante** o desenvolvimento
 - Usuários, clientes, desenvolvedores, programadores de manutenção, operadores, gerentes de projeto,...
- Mantenha o "**protótipo**" em funcionamento ao implantar **novos recursos** a cada iteração
 - Normalmente a cada 1 ou 2 semanas
 - Em vez de 5 fases principais, a cada mês
- Verifique com as partes interessadas o que vem a seguir, para **validar** a criação da coisa certa (vs. **verificar**)

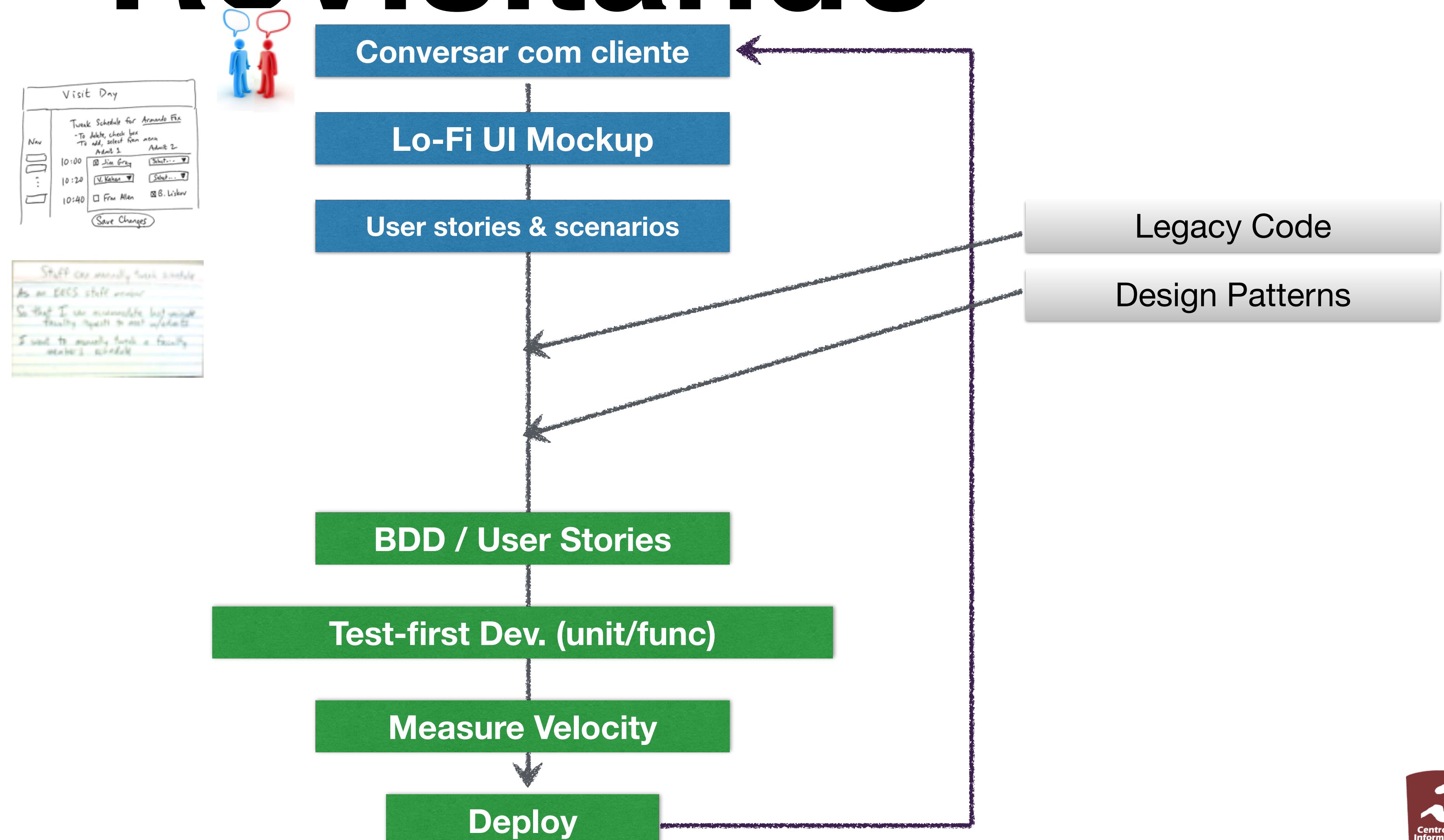


Behavior-Driven Design (BDD)

- BDD faz perguntas sobre o comportamento da app **antes e durante o desenvolvimento**, para reduzir falhas de comunicação
 - Validação vs. Verificação
- Requisitos escritos como **histórias de usuário**
 - Leves descrições de como a app é utilizada
- BDD concentra-se no **comportamento** de execução da aplicação ao invés da sua **implementação**
- Test-Driven Design ou TDD (em breve) testa a implementação

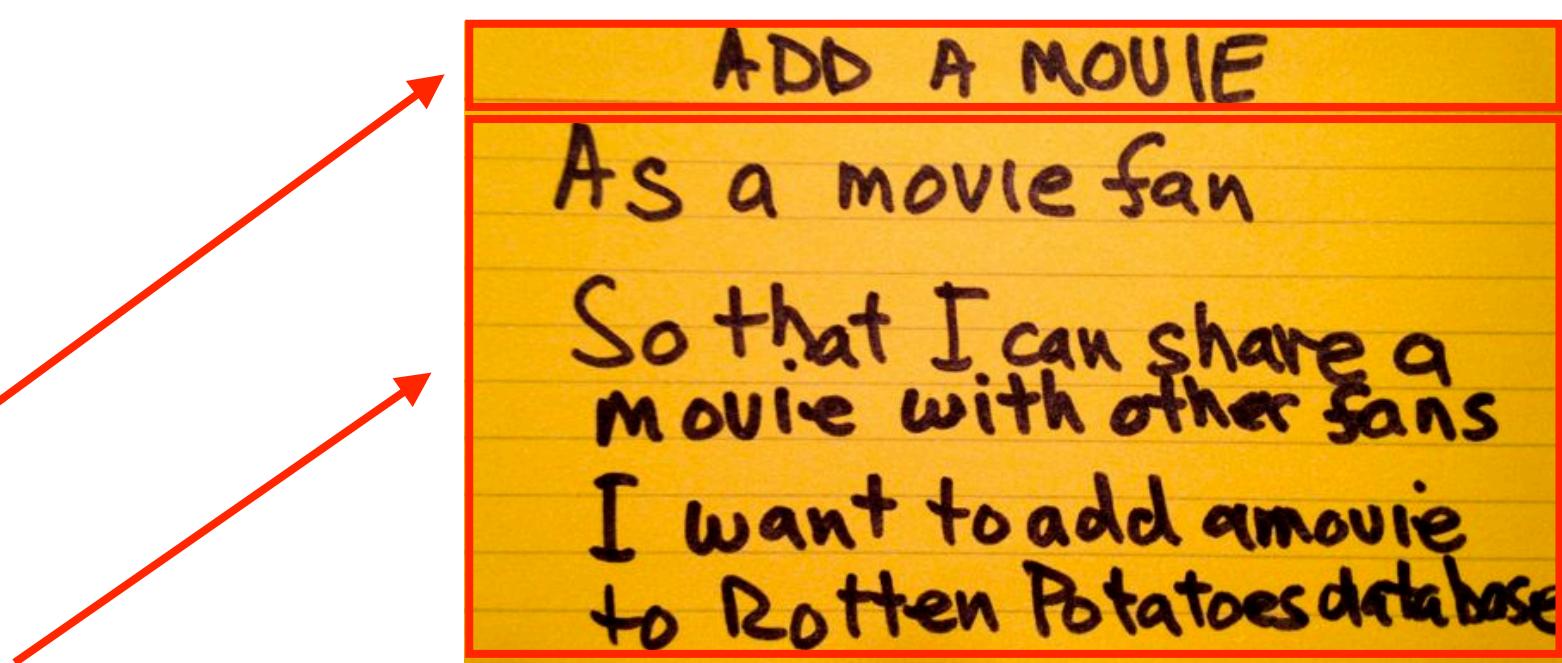


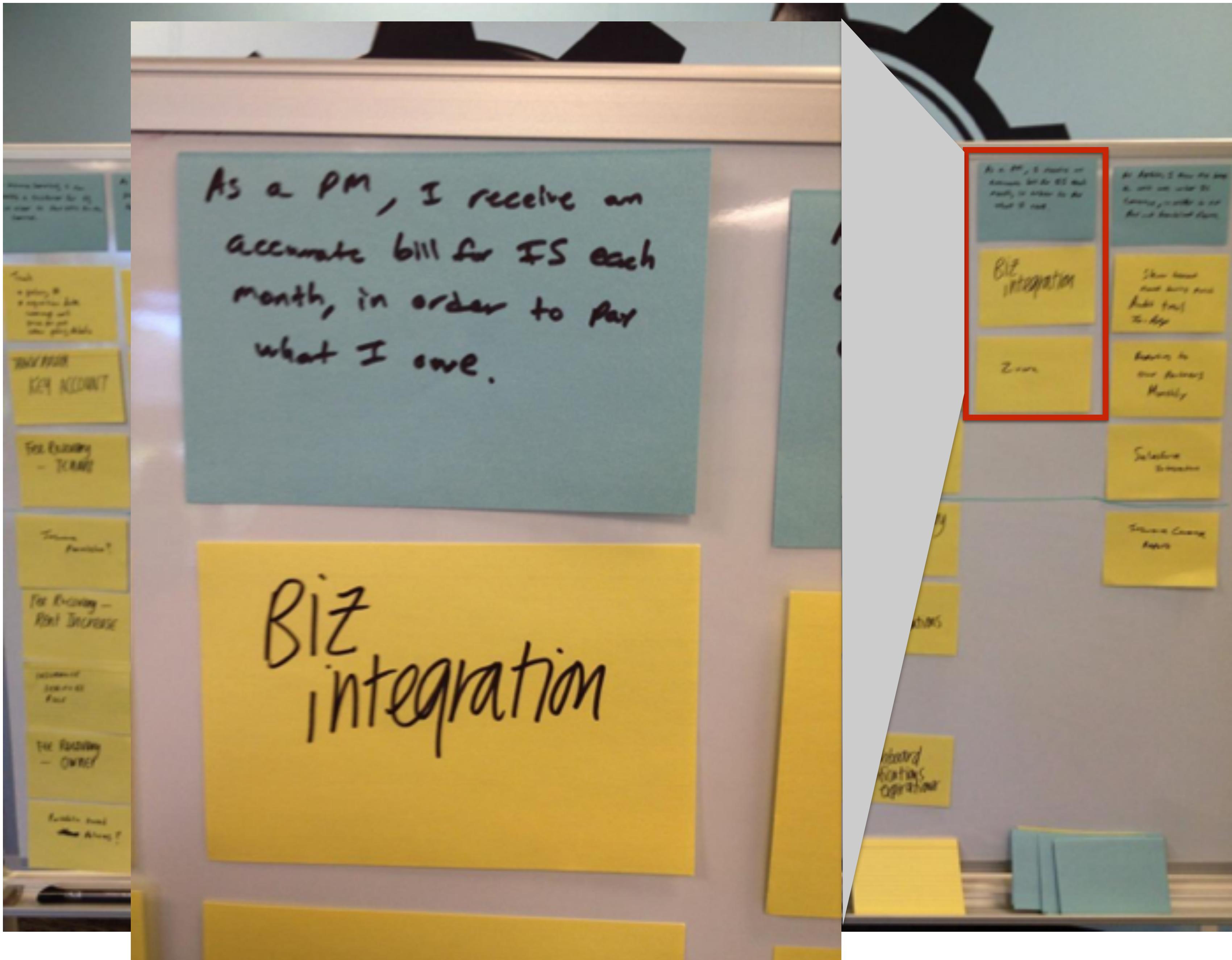
Revisitando



User Stories

- 1-3 sentenças em linguagem natural (cotidiana)
 - Se encaixa em um cartão 3"x5"
 - Escrito por/com cliente
- Formato "connextra":
 - nome do recurso/característica (feature)
 - **Como um** [tipo de stakeholder],
 - **Para que** [eu possa alcançar algum objetivo],
 - **Eu quero** [fazer alguma tarefa]
- As 3 frases devem estar lá, podem estar em qualquer ordem
- Ideia: histórias de usuário podem ser formuladas como testes de aceitação antes que o código seja escrito





Thanks: Klaus Schauser, CEO, Appfolio Inc

Por que cartão 3x5?

- (a partir da comunidade de User Interface)
- Nonthreatening => todos os stakeholders participam do brainstorming
- Fácil de reorganizar => todos os stakeholders participam na definição de prioridades
- Como histórias devem ser curtas, fica fácil de mudar durante o desenvolvimento
 - Muitas vezes temos novos **insights** durante o desenvolvimento



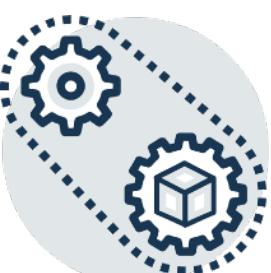
Trabalhando com User Stories

- O que faz uma boa história de usuário?
- Como desenvolvê-las em conjunto com o cliente?
- Como estimar a dificuldade de terminar uma?
- Como priorizar?
- Como rastrear quem está trabalhando em quais?
- Como verificar se o código entregue se comporta de acordo com as histórias acordadas?



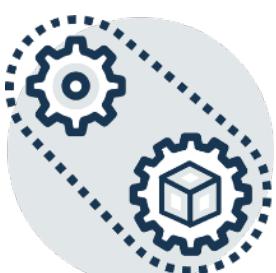
Diferentes stakeholders podem descrever o mesmo comportamento de formas diferentes

- Veja quais das propriedades próximas a mim estão vagos
 - Como interessado em reservar uma propriedade
 - Para que eu possa reservar uma propriedade disponível
 - Eu quero ver qual das propriedades próximas a mim estão disponíveis
 - Mostrar oportunidade de reserva
 - Como agente de turismo
 - Para que eu possa induzir um usuário para alugar uma propriedade
 - Eu quero mostrar-lhe quais propriedades próximas a ele estão disponíveis
- A. Isso deve ser consolidado em uma única história de usuário do ponto de vista do usuário?
- B. Isso deve ser consolidado em uma única história de usuário do ponto de vista do agente?
- C. Isso deve ser deixado em duas histórias, porque a funcionalidade e a experiência do usuário podem ser diferentes e ambas podem ser importantes?



Diferentes stakeholders podem descrever o mesmo comportamento de formas diferentes

- Veja quais das propriedades próximas a mim estão vagos
 - Como interessado em reservar uma propriedade
 - Para que eu possa reservar uma propriedade disponível
 - Eu quero ver qual das propriedades próximas a mim estão disponíveis
 - Mostrar oportunidade de reserva
 - Como agente de turismo
 - Para que eu possa induzir um usuário para alugar uma propriedade
 - Eu quero mostrar-lhe quais propriedades próximas a ele estão disponíveis
- A. Isso deve ser consolidado em uma única história de usuário do ponto de vista do usuário?
- B. Isso deve ser consolidado em uma única história de usuário do ponto de vista do agente?
- C. Isso deve ser deixado em duas histórias, porque a funcionalidade e a experiência do usuário podem ser diferentes e ambas podem ser importantes?



Product Backlog

- Sistemas reais tem 100s de histórias de usuário
- Backlog: histórias de usuário ainda não concluídas
- Priorizar os itens mais valiosos
- Organizar para que eles correspondam ao SW liberado ao longo do tempo

