

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-
Compartilhual 3.0 Não Adaptada



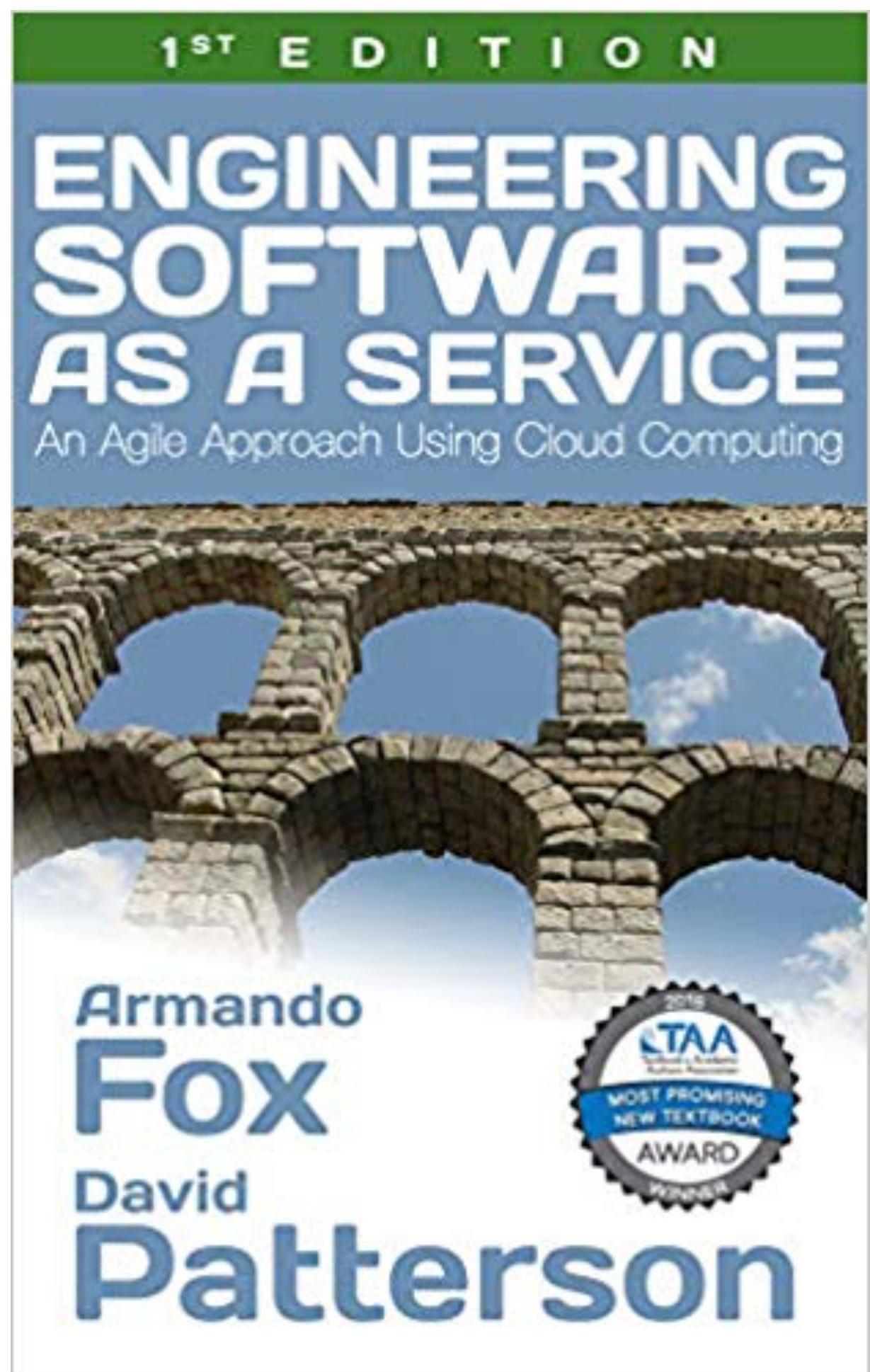
Mais informações visite

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



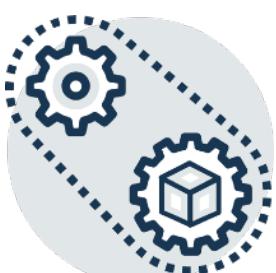
Referências

- 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/>



Linguagens - Plano de 8 passos

1. Tipos e digitação
2. Tipos/operações primitivas, fluxo de controle, strings...
3. Métodos (funções, procedimentos)
4. Abstração e encapsulamento
5. Expressões idiomáticas
6. Gerenciamento de bibliotecas e dependências
7. Depuração (idioma e estrutura)
8. Ferramentas de teste



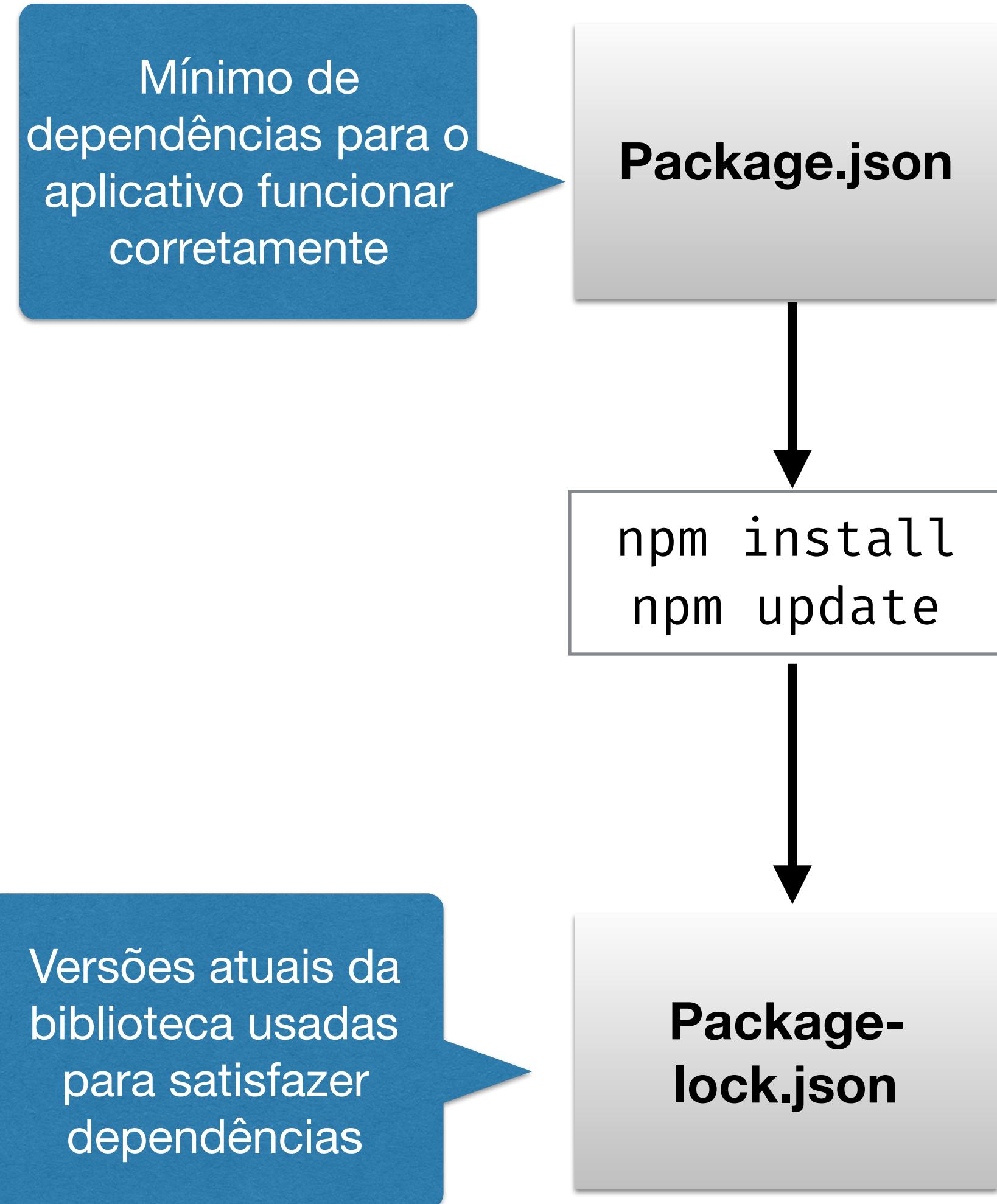
O dilema de sucesso da reutilização de software...

- Os aplicativos modernos dependem **muito** de bibliotecas, e as bibliotecas dependem umas das outras
 - por exemplo, GitLab → 836 bibliotecas
- A biblioteca pode depender da versão do idioma
- Como garantir que o aplicativo seja sempre implantado com bibliotecas apropriadas?



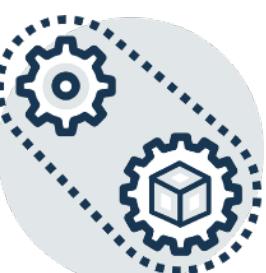
Sistema de Gerenciamento de Pacotes

- e.g. pip (Python), bundle (Ruby), npm (JavaScript)...
- Registra dependências de bibliotecas da aplicação, incluindo as versões da biblioteca
- Resolve as dependências quando a aplicação, as bibliotecas, etc. mudarem



Versão semântica (semver.org) e Deprecation

- Emergente padrão *de facto*; existem outros
- Número da versão semântica: **x.y.z**
 - Alteração no z: correção de bug/correção menor
 - Alteração na versão y: secundária (minor), adiciona funcionalidade, mas preserva a compatibilidade com versões anteriores
 - Mudança em x: versão principal, quebrando alterações
- **Deprecation warnings** sinalizam comportamento(s) que será(ão) removido(s) ou alterado(s) de forma incompatível
 - Para garantir que você os veja, atualize para a última versão secundária (minor) antes de alterar para a versão principal



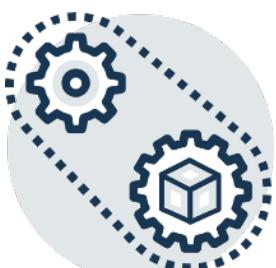
Pergunta

- Quais arquivos de gerenciamento de bibliotecas em um aplicação Javascript (Node) devem ser versionados?
 - A. Somente o package
 - B. Somente o package-lock
 - C. Tanto o package quanto package-lock
 - D. Nenhum deles



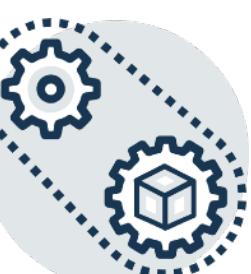
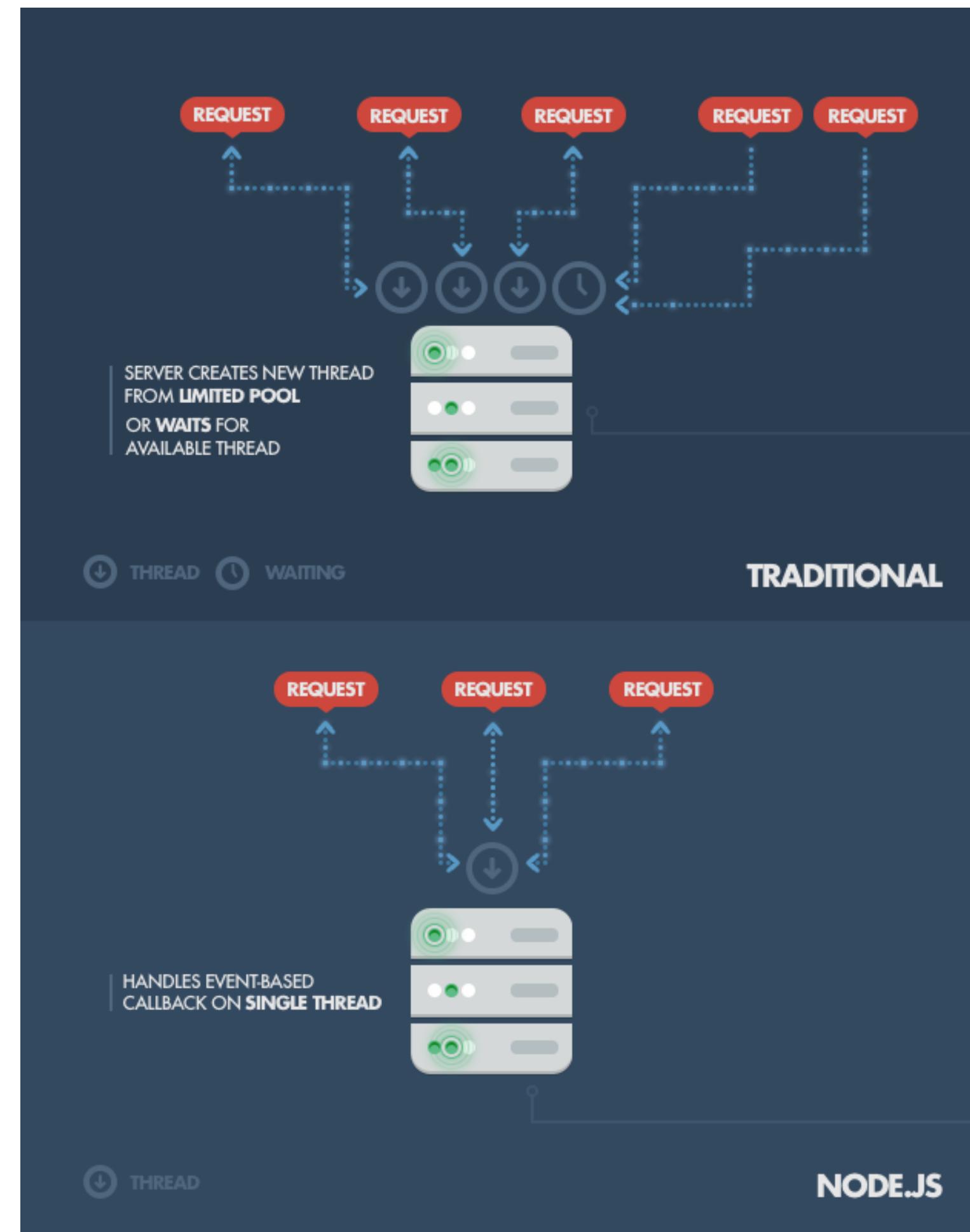
Pergunta

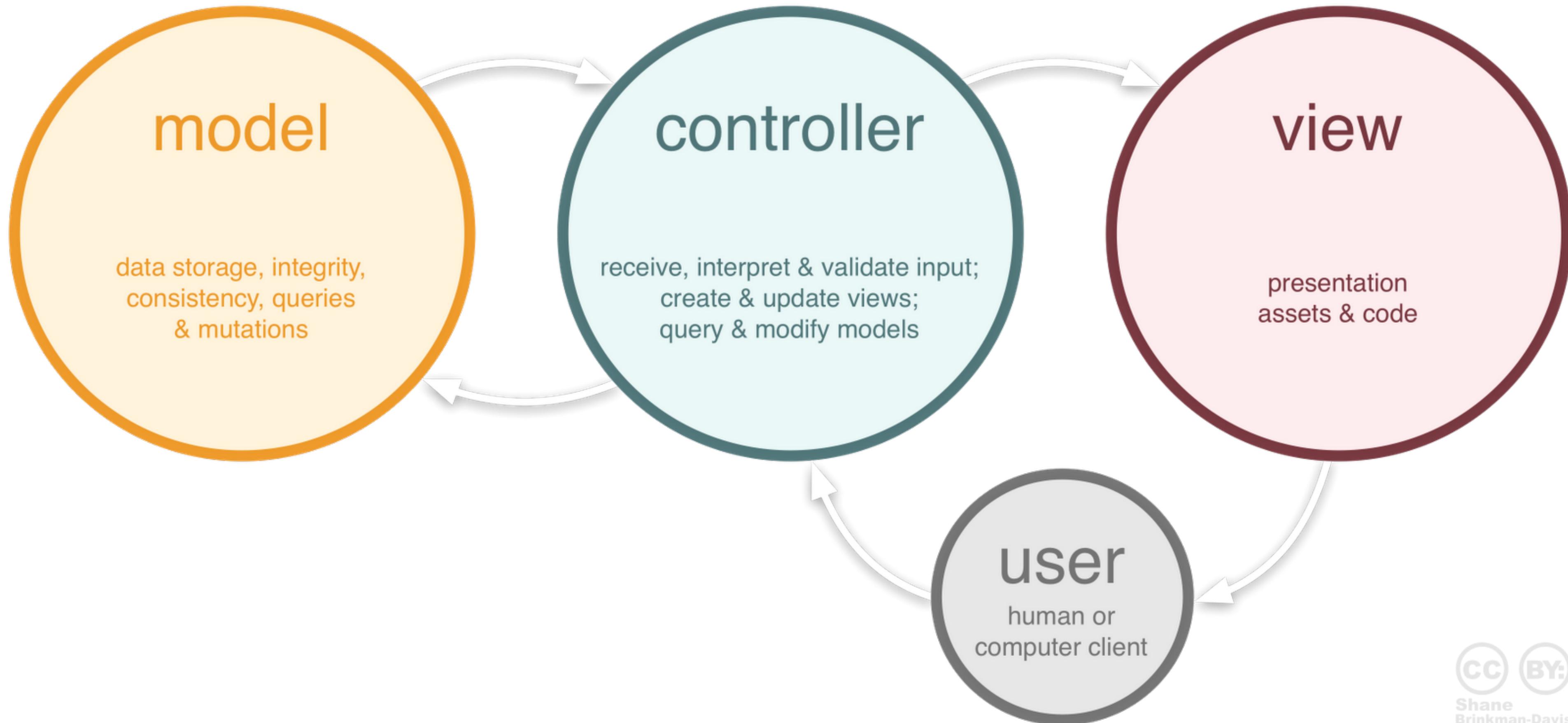
- Quais arquivos de gerenciamento de bibliotecas em um aplicação Javascript (Node) devem ser versionados?
 - A. Somente o package
 - B. Somente o package-lock
 - C. Tanto o package quanto package-lock
 - D. Nenhum deles



Revisão Node.js

- Node.js não é uma nova **bala de prata** para o desenvolvimento web
- É uma plataforma que atende uma **necessidade específica**.
- Node.js **não** é adequado para **operações intensivas em CPU**; de fato, usá-lo para computação pesada **anulará** quase todas as suas vantagens.
- Node.js se destaca é na criação de aplicativos de rede **escaláveis** e **rápidos**, pois é capaz de lidar com um **grande número de conexões simultâneas** com **alto rendimento**, o que equivale a **alta escalabilidade**.

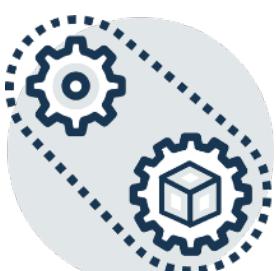




CC BY:
Shane
Brinkman-Davis

Model-View-Controller

O popular MVC

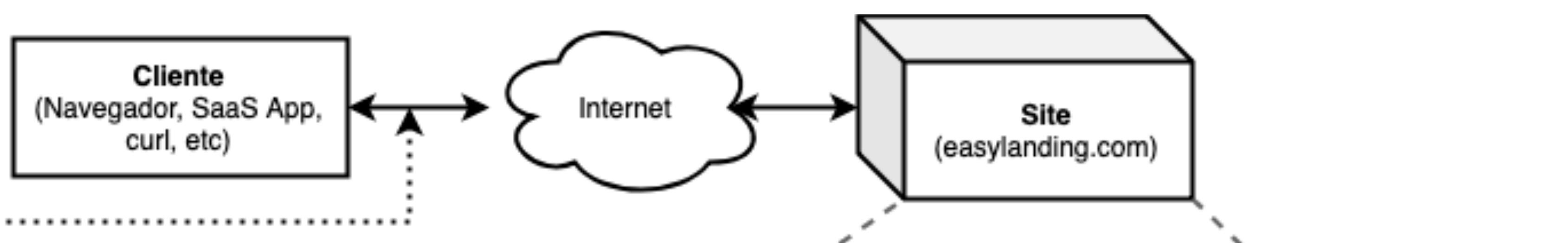


Para onde vão os frameworks?

- Existe **estrutura de aplicação comum** ...
- em aplicações interativas **voltadas para o usuário**...
- ... que poderiam **simplificar** o **desenvolvimento** de aplicativos, se os capturássemos em um **framework**?



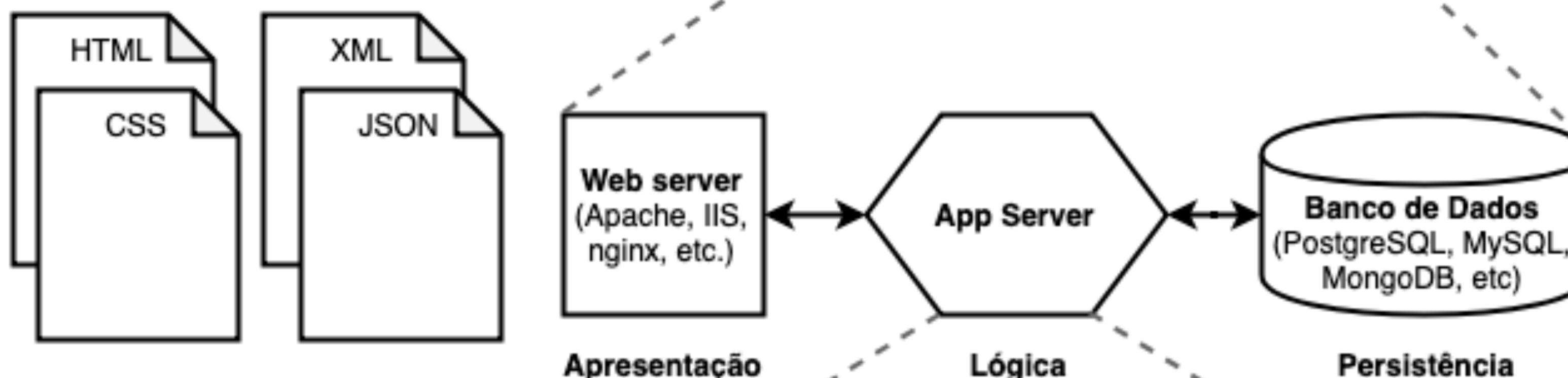
Cliente-Servidor vs P2P



HTTP & URIs

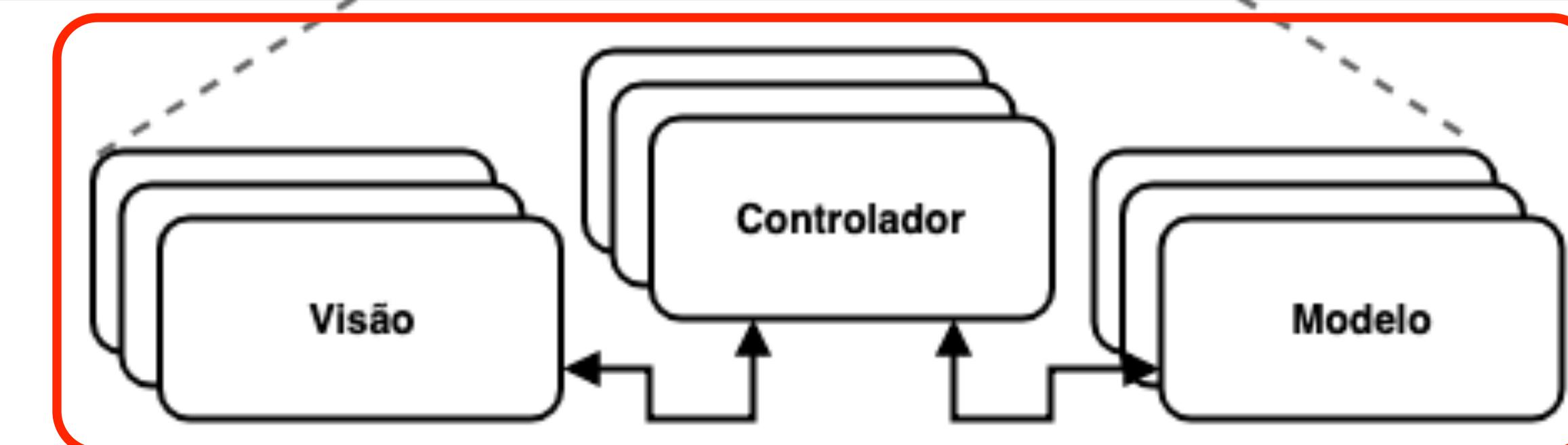
Arquitetura 3 camadas

Escalabilidade



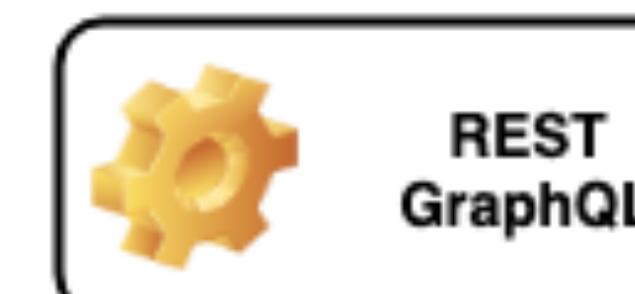
Padrões Arquiteturais

Model-View- Controller

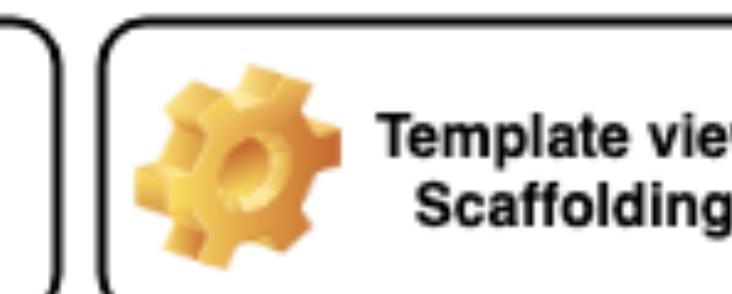


Padrões de Projeto

Idiomas



REST
GraphQL



Template view
Scaffolding



ORM
Active Record
Data Mapper



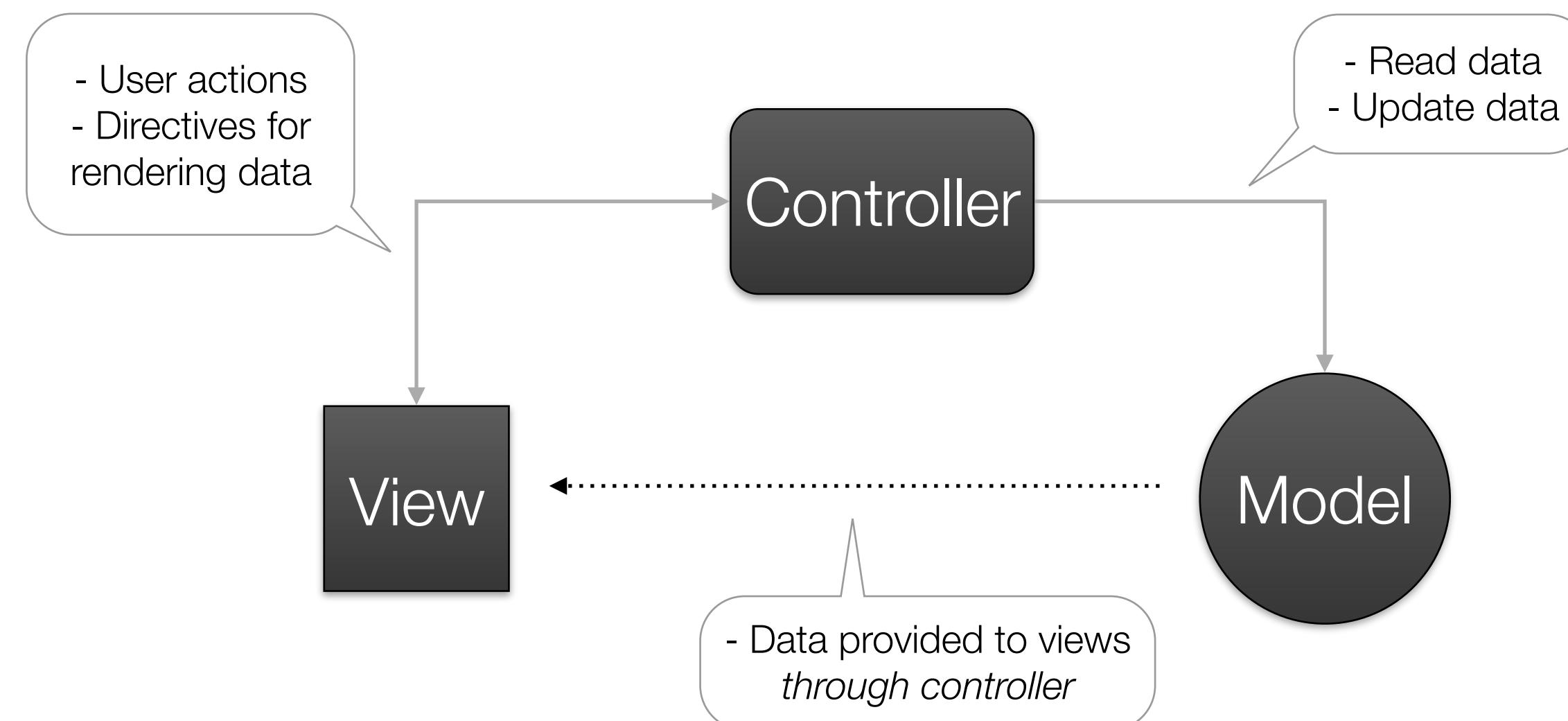
Padrão arquitetural Model-View-Controller

- Padrão arquitetural utilizado para ajudar na separação da camada de interface com o usuário das demais partes da aplicação
 - O model contém as classes cujas instâncias devem ser visualizadas e manipuladas
 - O view contém os objetos utilizados para renderizar a aparências dos dados do modelo na interface do usuário
 - O controller contém os objetos que controlam e tratam as interações do usuário com o view e o model.
 - O padrão de projeto Observable é normalmente utilizado para separar o model do view.

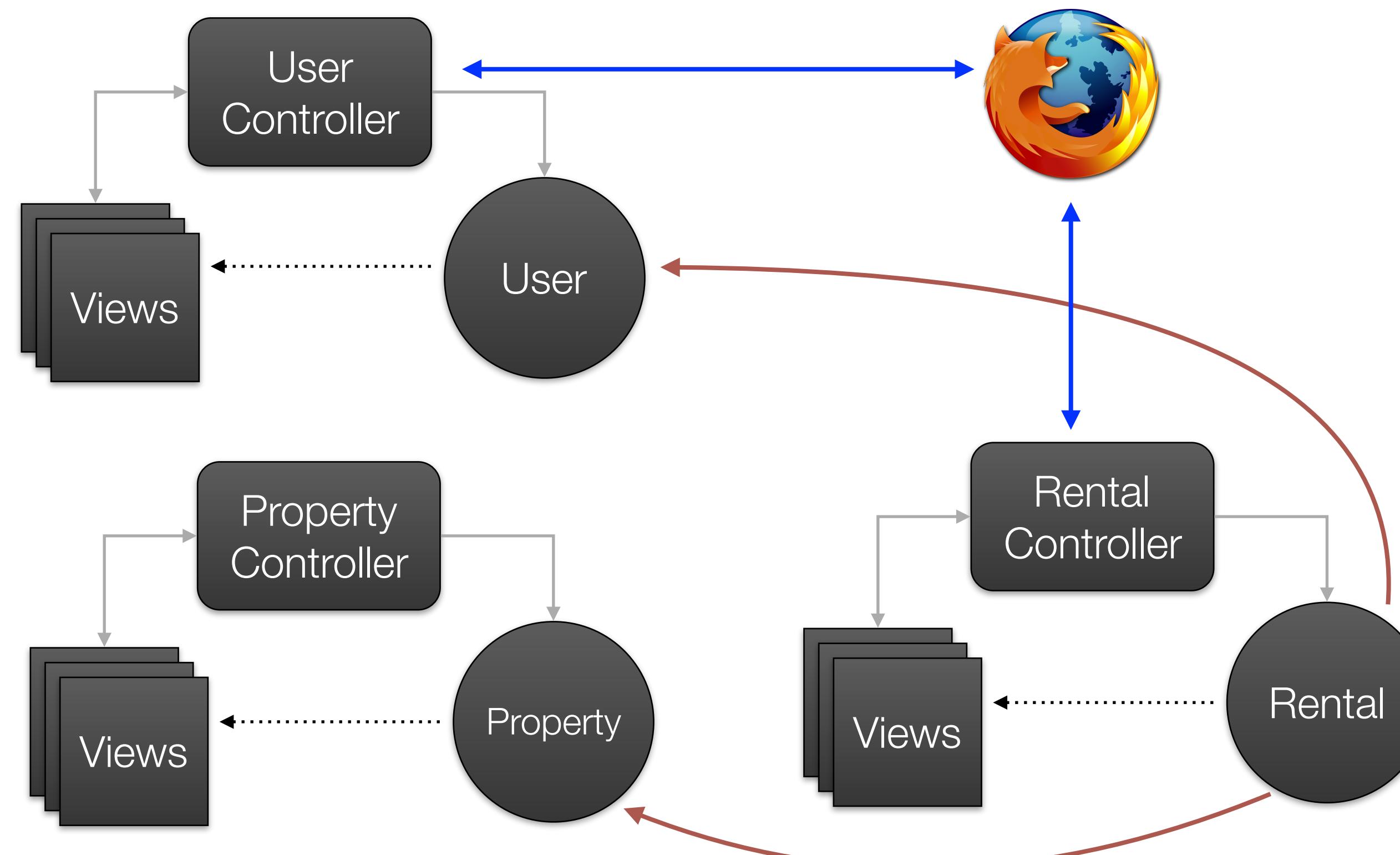


Padrão arquitetural Model-View-Controller

- **Objetivo:** separar a organização dos recursos (modelo) da interface do usuário e apresentação (view), introduzindo o controlador
 - medeia ações do usuário que solicitam acesso aos dados
 - apresenta dados para renderização pela view

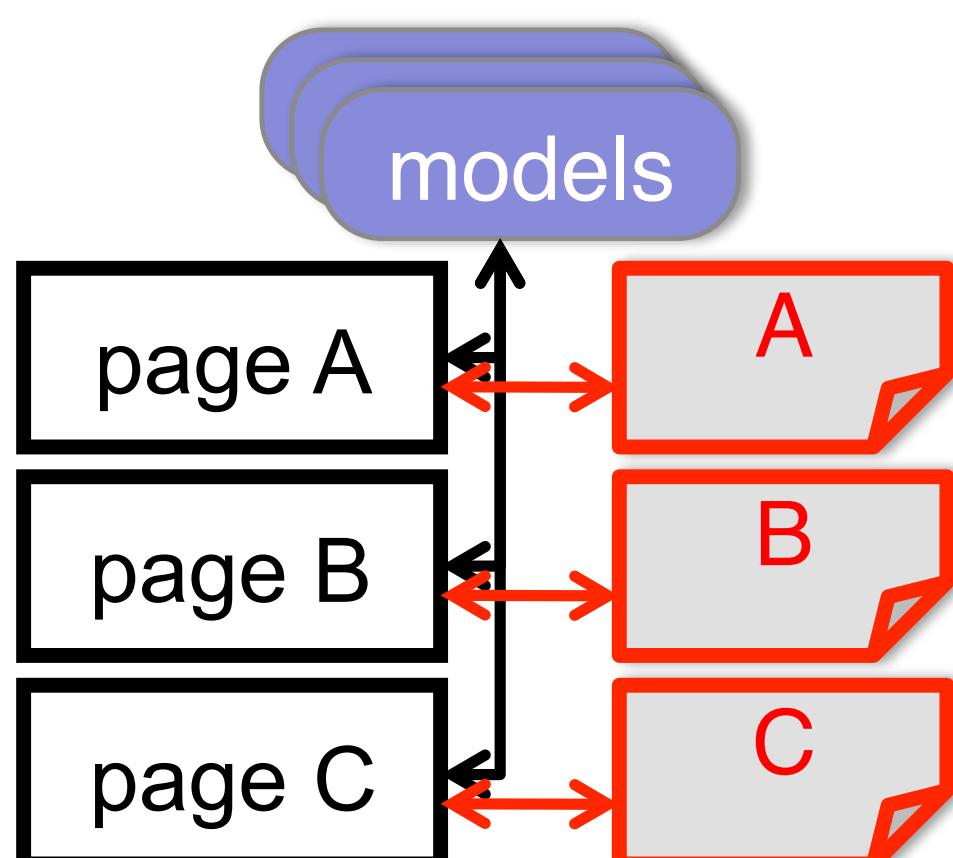


Cada entidade tem um model, controller e um conjunto de views

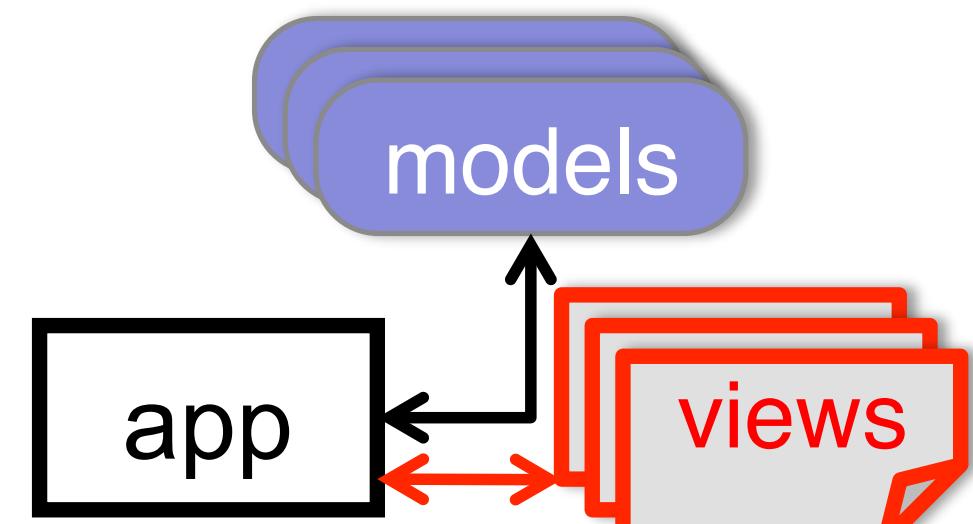


Alternativas ao MVC

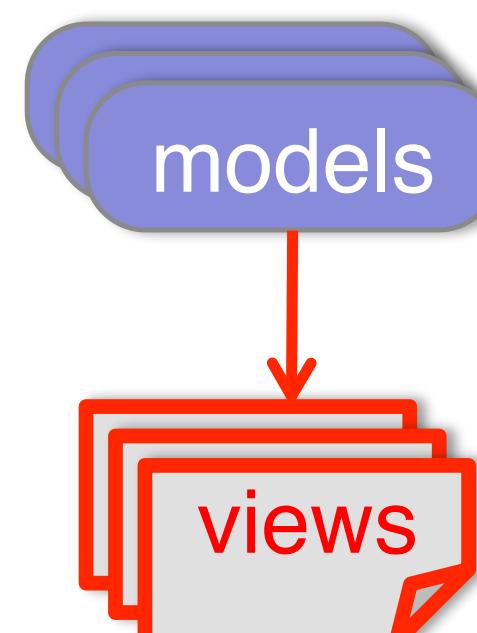
Page Controller
(Ruby Sinatra)



Front Controller
(J2EE servlet)



Template View
(PHP)



Node dá suporte a apps SaaS estruturadas como MVC, mas outras arquiteturas podem ser uma melhor opção para outras aplicações



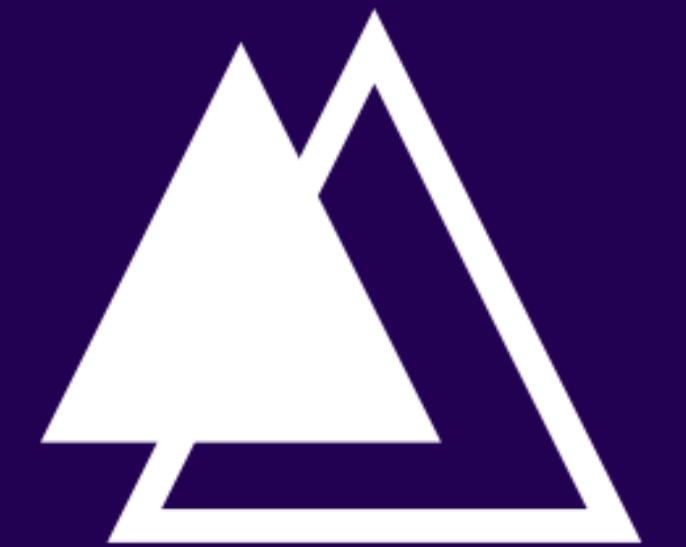
Pergunta

- Qual alternativa **NÃO** é verdade sobre o padrão arquitetural Model-View-Controller (MVC)
 - A. Nas apps SaaS na Web, controller actions e conteúdos da view são transmitidos usando HTTP
 - B. Todas apps MVC possuem tanto uma parte “cliente” (ex. Browser Web) e a parte da “nuvem” (ex. App Node.js na nuvem)
 - C. MVC é somente uma das diversas possibilidades de se estruturar uma app SaaS
 - D. Apps Peer-to-Peer podem ser estruturadas como MVC



Pergunta

- Qual alternativa **NÃO** é verdade sobre o padrão arquitetural Model-View-Controller (MVC)
 - A. Nas apps SaaS na Web, controller actions e conteúdos da view são transmitidos usando HTTP
 - B. Todas apps MVC possuem tanto uma parte “cliente” (ex. Browser Web) e a parte da “nuvem” (ex. App Node.js na nuvem)
 - C. MVC é somente uma das diversas possibilidades de se estruturar uma app SaaS
 - D. Apps Peer-to-Peer podem ser estruturadas como MVC



ADONIS

Olá AdonisJs!

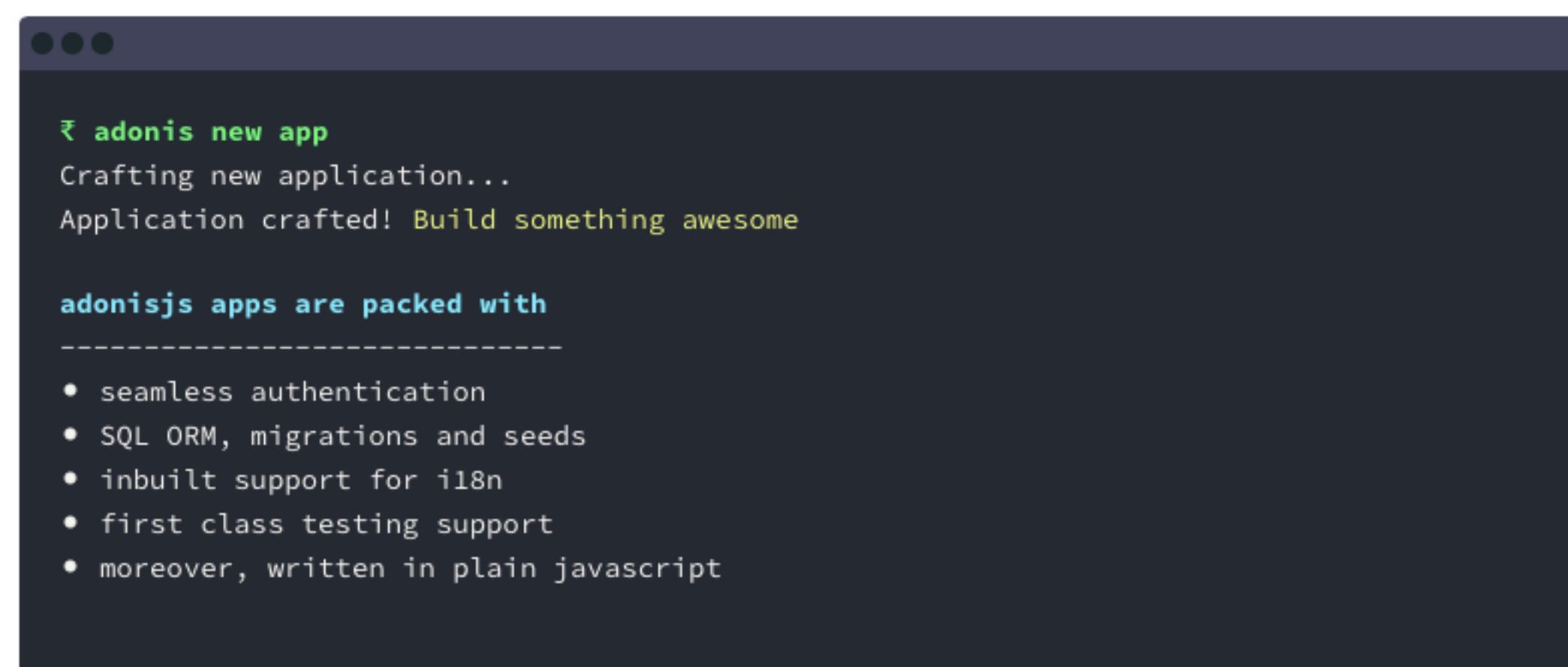
Do Zero ao CRUDI



AdonisJs, quem é ele na fila do pão!?

- É um framework Node.js que permite ao desenvolvedor focar apenas na regras de negócio deixando as decisões de projeto por conta dele
- Arquitetura MVC, RESTful, padrões de projeto...

<https://adonisjs.com>



```
...  
$ adonis new app  
Crafting new application...  
Application crafted! Build something awesome  
  
adonisjs apps are packed with  
-----  
• seamless authentication  
• SQL ORM, migrations and seeds  
• inbuilt support for i18n  
• first class testing support  
• moreover, written in plain javascript
```

Adonis Blog Demo

O que faremos?

- Criar a API de backend com AdonisJs
- Aplicação clone do AirBnB com recursos como:
 - Autenticação via JWT;
 - Cadastro de imóveis para aluguel;
 - Upload de fotos dos imóveis...
- Projeto disponível no Github
 - [https://github.com/vinicius3w/easylanding-server/
tree/v0.1-alpha](https://github.com/vinicius3w/easylanding-server/tree/v0.1-alpha)



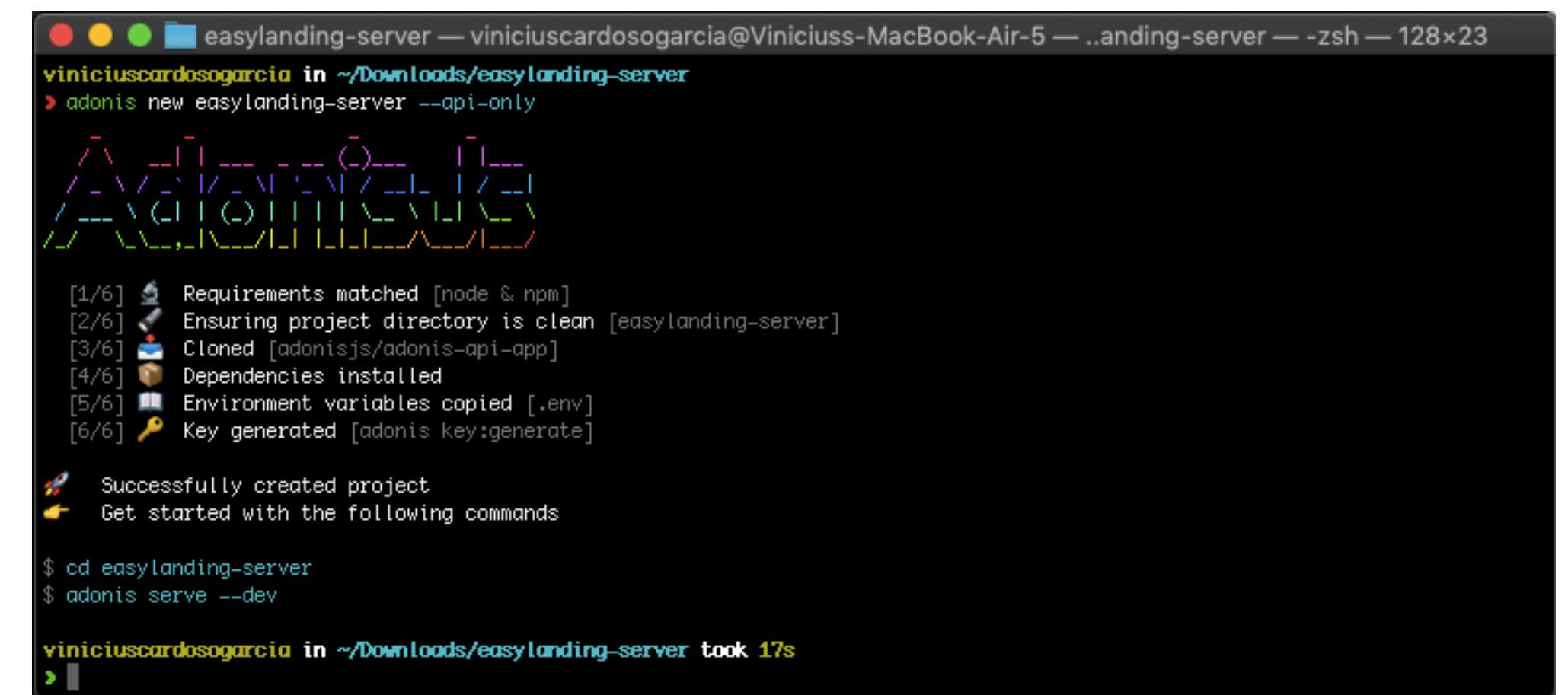
Iniciando com o Adonis

- Para começarmos o desenvolvimento da nossa aplicação, vamos instalar o **CLI (Command Line Interface)** do Adonis que vai nos ajudar muito durante o desenvolvimento:

```
npm i -g @adonisjs/cli
```

- Em uma pasta de sua escolha execute agora o seguinte comando que irá criar um novo projeto com a estrutura do Adonis:

```
adonis new easylanding-server --api-only
```



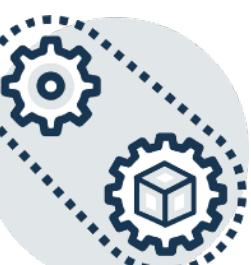
```
easylanding-server — viniciuscardosogarcia@Viniciuss-MacBook-Air-5 — ..anding-server — zsh — 128x23
viniciuscardosogarcia in ~/Downloads/easylanding-server
> adonis new easylanding-server --api-only

[1/6] Requirements matched [node & npm]
[2/6] Ensuring project directory is clean [easylanding-server]
[3/6] Cloned [adonisjs/adonis-api-app]
[4/6] Dependencies installed
[5/6] Environment variables copied [.env]
[6/6] Key generated [adonis key:generate]

Successfully created project
Get started with the following commands

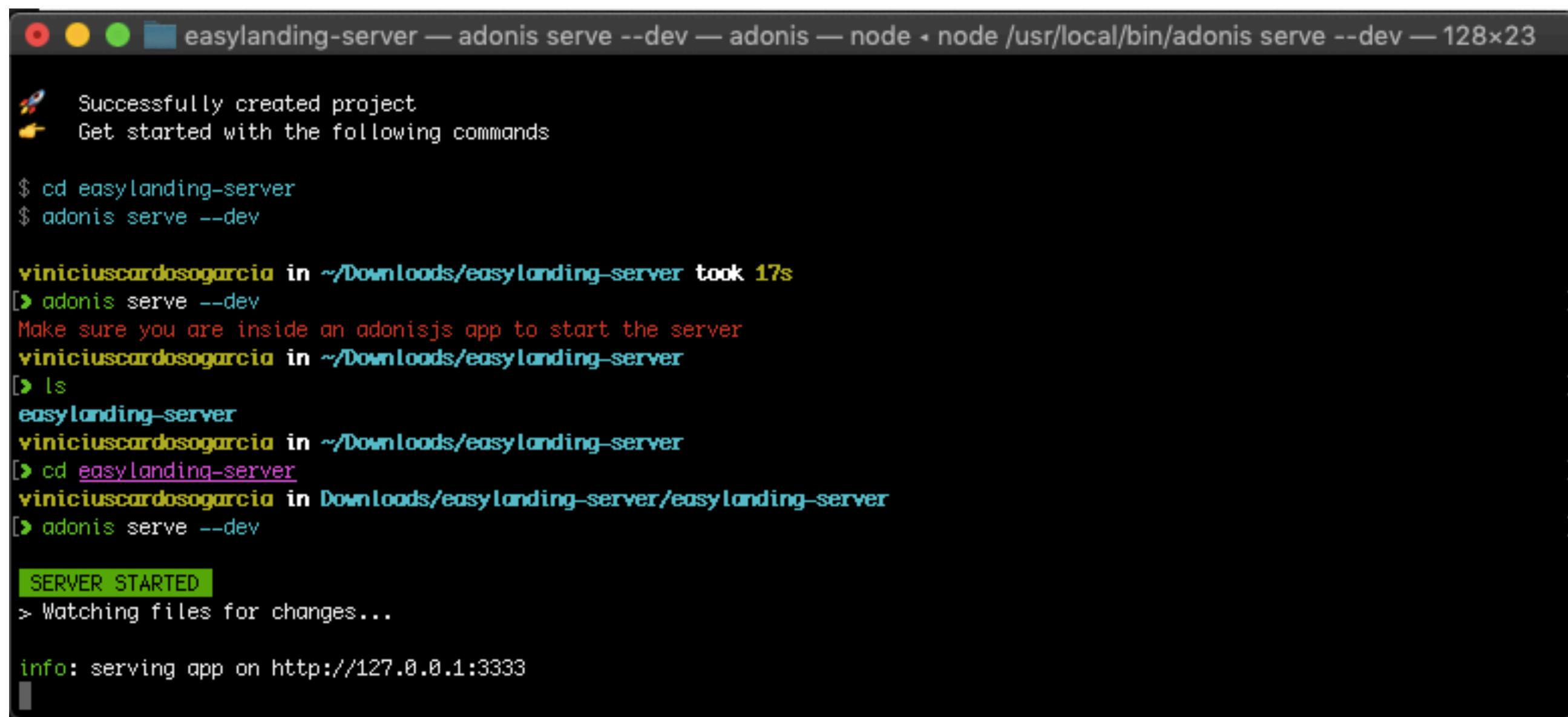
$ cd easylanding-server
$ adonis serve --dev

viniciuscardosogarcia in ~/Downloads/easylanding-server took 17s
>
```



Colocando no ar

- Agora abra a pasta do projeto recém-criado e rode o comando `adonis serve --dev`
- Esse comando basicamente iniciará o servidor de desenvolvimento que agora pode ser acessado no endereço <http://localhost:3333>
- Uma parte bem legal do Adonis é que se você alterar qualquer arquivo esse servidor **se auto reinicia** com as novas alterações (graças ao **Nodemon**).



```
Successfully created project
Get started with the following commands

$ cd easylanding-server
$ adonis serve --dev

viniciuscardosogarcia in ~/Downloads/easylanding-server took 17s
[> adonis serve --dev
Make sure you are inside an adonisjs app to start the server
viniciuscardosogarcia in ~/Downloads/easylanding-server
[> ls
easylanding-server
viniciuscardosogarcia in ~/Downloads/easylanding-server
[> cd easylanding-server
viniciuscardosogarcia in Downloads/easylanding-server/easylanding-server
[> adonis serve --dev

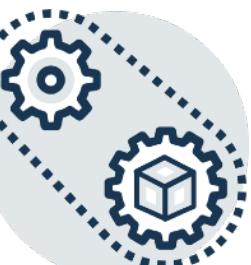
SERVER STARTED
> Watching files for changes...

info: serving app on http://127.0.0.1:3333
```



Criando API REST com AdonisJS

- O AdonisJS utiliza o padrão MVC como arquitetura
- Toda camada de banco de dados é abstraída pelo **Lucid ORM** (<https://adonisjs.com/docs/4.1/lucid>)
 - isso quer dizer que jamais escreveremos uma consulta no banco de dados como `SELECT * FROM users WHERE id = 1`
 - Ao invés disso utilizaremos métodos disponíveis nos nossos models, como: `const user = User.find(1);`



Criando API REST com AdonisJS

- Para iniciar nosso app precisamos de um banco de dados SQL disponível
 - Eu vou utilizar o PostgreSQL mas fique à vontade para utilizar MySQL ou até SQLite.
- Rode o comando abaixo dependendo do banco que escolher:
 - MySQL: `npm install mysql`
 - PostgreSQL: `npm install pg`
 - SQLite: `npm install sqlite3`



Configurando acesso ao Banco

- Vamos configurar os acessos ao banco de dados através do arquivo `.env` que define nossas variáveis ambiente
 - Ou seja, dados que podem variar de acordo com o ambiente da nossa aplicação: **desenvolvimento**, **produção**, **staging**, **testes**, etc...
- Dentro do arquivo `.env` altere as seguintes linhas para os dados de acesso à sua base dados
 - Lembre de alterar o `DB_CONNECTION` para o tipo de base que você está usando

```
DB_CONNECTION=pg
DB_HOST=127.0.0.1
DB_PORT=5432
DB_USER=easylanding
DB_PASSWORD=
DB_DATABASE=easylandingdb
```



Migrations

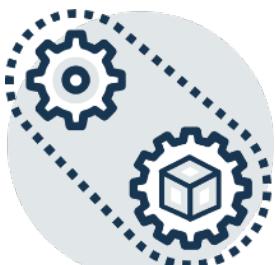
- Se tudo estiver correto você agora poderá executar o comando adonis migration:run na raiz do seu projeto e obterá um resultado semelhante à

```
migrate: 1503250034279_user.js  
migrate: 1503250034280_token.js  
Database migrated successfully in 262 ms
```



O que aconteceu?

- Executamos todas **migrations** do nosso projeto em nossa nova base
 - Mas o que são migrations?
 - Pense nelas como um **controle de versão da nossa base**
 - Com elas podemos facilmente criar novas tabelas, alterar colunas e todos os outros desenvolvedores do projeto sempre terão a base de dados atualizada com o código



Hogwarts

- Se você abrir o arquivo app/Models/User.js não verá nenhum campo da base de dados referenciado na classe
- Isso porque tudo isso é abstraído pelo Lucid e ele identifica os campos para a gente **automagicamente**

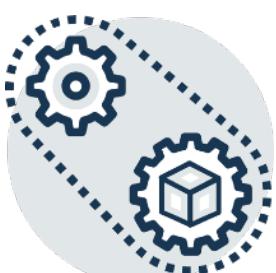


Autenticação com JWT

- Para iniciar o processo de autenticação vamos criar nosso controller de usuários, para isso, em seu terminal execute o comando:

```
adonis make:controller User --type http
```

- Nesse momento um novo arquivo app/
Controllers/Http/UserController.js deve
ter sido criado e dentro dele faremos nossas
primeiras ações para cadastro e login de usuários



UserController

- Basicamente o que estamos fazendo aqui em cada linha é:
 - Importamos o model de usuário;
 - Definimos um novo método assíncrono `create` que recebe um parâmetro `request` que possui todos dados da requisição como parâmetros, corpo, headers, etc
 - Estamos utilizando `{}` por volta dos parâmetros, isso porque no AdonisJS utilizamos a desestruturação do **ES6** no `controller`
 - Buscamos os campos de `username`, `email` e `password` do corpo da nossa `requisição` e os armazenamos em um objeto chamado `data`;
 - Criamos um **novo usuário** repassando os parâmetros vindos da `requisição` e salvamos esse novo usuário em uma variável `user`;
 - Retornamos o novo usuário como resultado da requisição, como selecionamos, no nosso caso o retorno será um JSON.

```
"use strict"

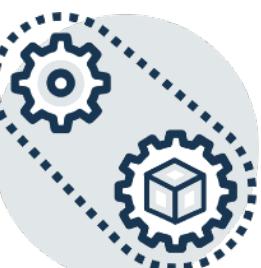
const User = use("App/Models/User")

class UserController {
    async create ({ request }) {
        const data = request.only(["username", "email", "password"])

        const user = await User.create(data)

        return user
    }
}

module.exports = UserController
```



Rota

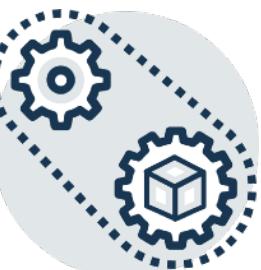
- Com o **controller** criado, vamos criar nossa **rota** que dará acesso a esse método, para isso no arquivo `start/routes.js` troque seu conteúdo para

```
'use strict'

const Route = use('Route')

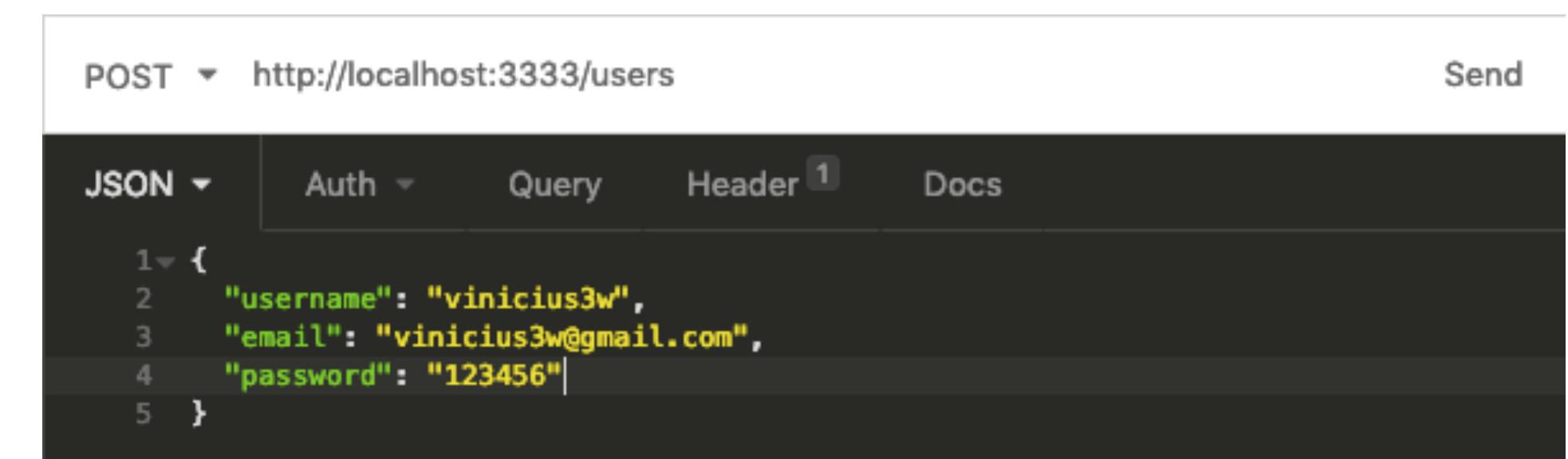
Route.post('/users', 'UserController.create')
```

- Pronto, acabamos de criar uma rota que aceita o método POST no endereço `/users` e chama o método `create` no controller `UserController`



Como testar!? Insomnia!

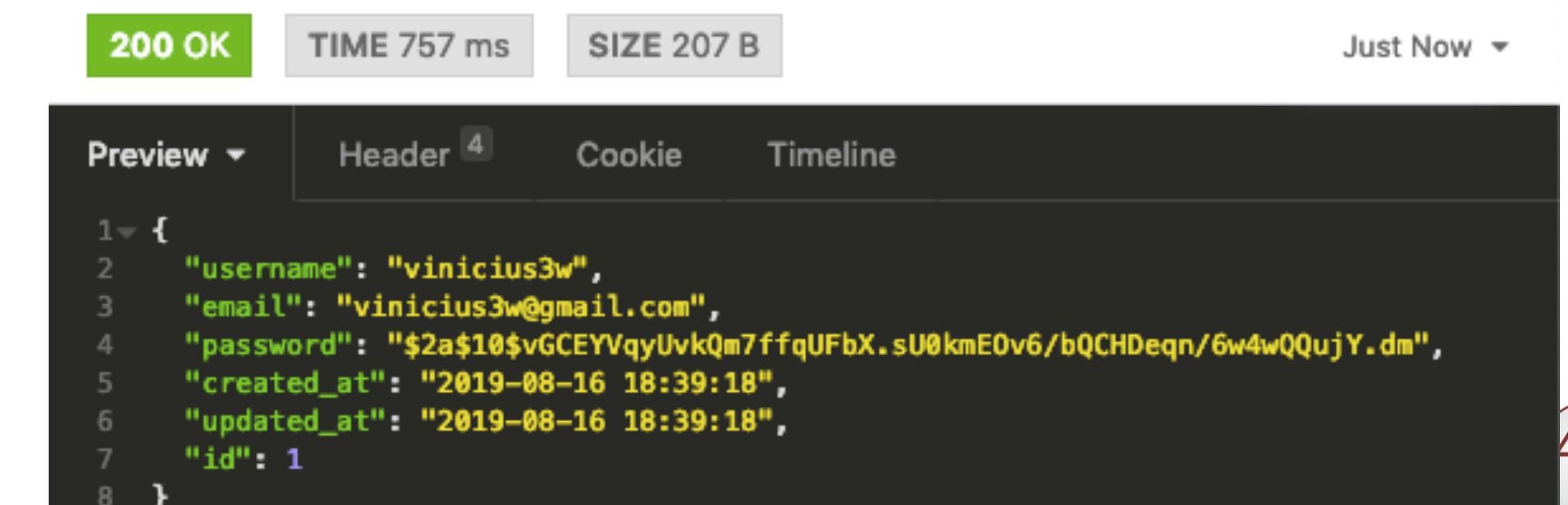
- Dentro do Insomnia clique na ação “**New request**” para criar uma nova requisição, nomeie a mesma como “**Cadastro**” e com ação POST e no tipo de corpo da requisição selecione JSON
- No topo preencha com o endereço para a rota que acabamos de criar e adicione alguns dados ao corpo em formato JSON para criarmos o primeiro usuário
- Agora, clique no botão “Send” para realizar a requisição e você deve obter um resultado similar a esse



POST `http://localhost:3333/users` Send

JSON Auth Query Header 1 Docs

```
1 {  
2   "username": "vinicius3w",  
3   "email": "vinicius3w@gmail.com",  
4   "password": "123456"  
5 }
```



200 OK TIME 757 ms SIZE 207 B Just Now

Preview Header 4 Cookie Timeline

```
1 {  
2   "username": "vinicius3w",  
3   "email": "vinicius3w@gmail.com",  
4   "password": "$2a$10$vGCEYVqyUvkQm7ffqUFbX.sU0kmE0v6/bQCHDeqn/6w4wQQujY.dm",  
5   "created_at": "2019-08-16 18:39:18",  
6   "updated_at": "2019-08-16 18:39:18",  
7   "id": 1  
8 }
```



Autenticação

- Agora que temos nosso usuário cadastrado vamos fazer a lógica de autenticação
- Como uma boa prática vamos criar um **novo controller** para lidar apenas com a **autenticação (sessão)**, para isso, execute o comando adonis make:controller Session --type http
- Além de buscar a `request` buscamos também uma variável `auth` dos parâmetros que é o meio que o Adonis nos dá para realizarmos a autenticação de forma simples
- Resgatamos os campos de **e-mail** e **senha** da requisição e executamos o método `auth.attempt` para fazer **login** e retornar nosso **token JWT**

```
'use strict'

class SessionController {
  async create ({ request, auth }) {
    const { email, password } = request.all()

    const token = await auth.attempt(email, password)

    return token
  }
}

module.exports = SessionController
```



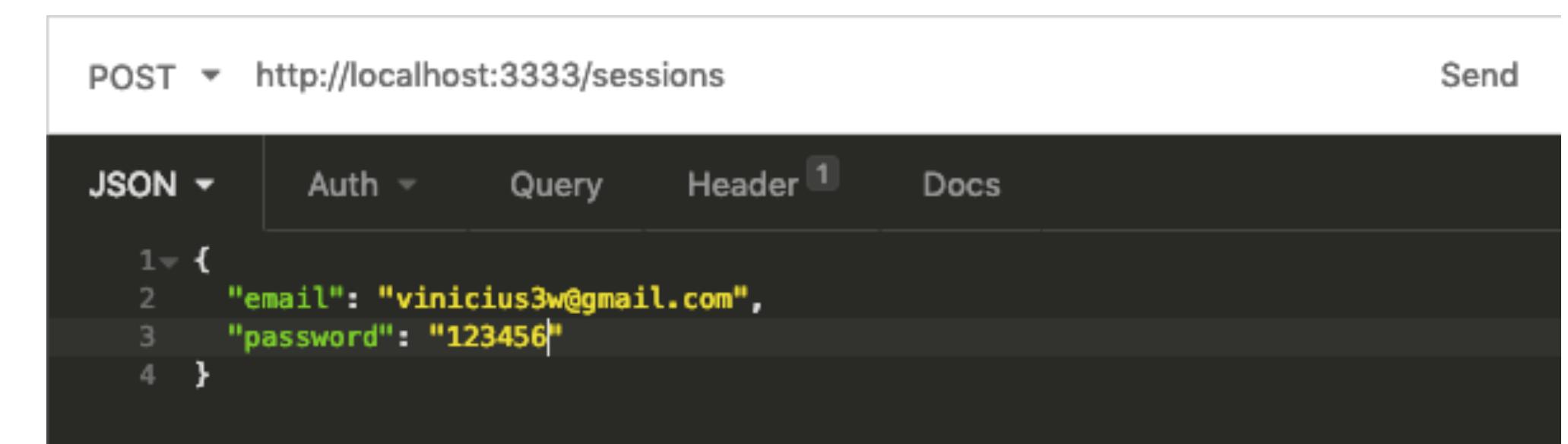
Criando a rota de autenticação

- Vamos agora criar nossa rota de autenticação sobrescrevendo o conteúdo do arquivo `start/routes.js`
- Agora no **Insomnia** crie uma **nova requisição** com a **nova rota** passando o **e-mail** e **senha** do usuário recém-criado para testar tudo isso
- Enviando essa requisição agora teremos acesso ao **token JWT** que servirá para **validarmos** se o usuário está autenticado ou não em nosso app

```
'use strict'

const Route = use('Route')

Route.post('/users', 'UserController.create')
Route.post('/sessions', 'SessionController.create')
```



POST <http://localhost:3333/sessions> Send

JSON Auth Query Header 1 Docs

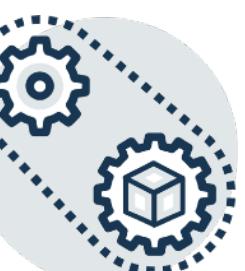
```
1 {  
2   "email": "vinicius3w@gmail.com",  
3   "password": "123456"  
4 }
```



200 OK TIME 114 ms SIZE 164 B Just Now

Preview Header 4 Cookie Timeline

```
1 {  
2   "type": "bearer",  
3   "token":  
4     "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlaWQiOjEsImlhCI6MTU2NTk5MjE4Mn0.AEx4N8HlMx7kG8f1fxkc5xmjzf0TyjxSurab8npXm4",  
5   "refreshToken": null  
6 }
```



Sumário

- A API já está mais do que **pronta** para partirmos para **os próximos passos** como **criação de imóveis**, **upload de imagens** e **retorno de imóveis próximos baseados na latitude e longitude do usuário**
- Com esse **token** que recebemos no **login** vamos conseguir **validar nas requisições que precisam de autenticação**, como um cadastro de imóvel, se o usuário é válido e está autenticado em nossa aplicação
- Criamos toda nossa API com AdonisJS do total zero realizando **cadastro e autenticação** via JWT e salvando nossos dados no banco.
- Aprendemos também alguns conceitos importantes como o **ORM**, **models**, **controllers**, **migrations**, **JWT**, etc...
- Código dessa parte
 - <https://github.com/vinicius3w/easylanding-server/tree/v0.1-alpha>

