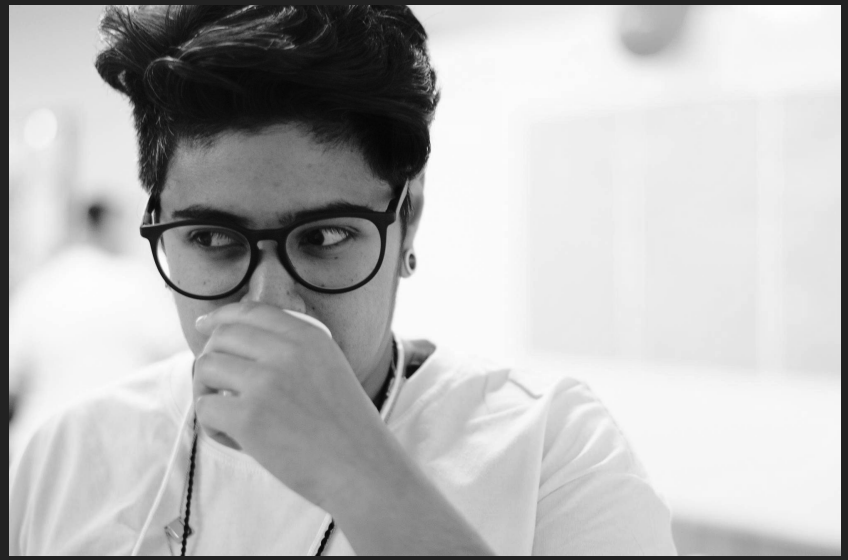
DESIGN PATTERNS

Como **evitar** problemas no futuro

Zalba Monteiro

Desenvolvedora Full Stack



Desenvolvedora de software há **7 anos**, atualmente trabalhando na CWI Software.

Amante de tecnologias que envolvem o universo **JavaScript**, entusiasta de iniciativas de **empoderamento** de pessoas negras e protagonismo feminino. Potterhead sim!! Feminista sim!!!

E entre um código e outro, ainda dá tempo de tomar um bom café, fazer uma tatuagem ou cozinhar para os amigos.

Design Pattern é uma solução reutilizável geral para um problema que ocorre com frequência dentro de um determinado contexto no projeto de software

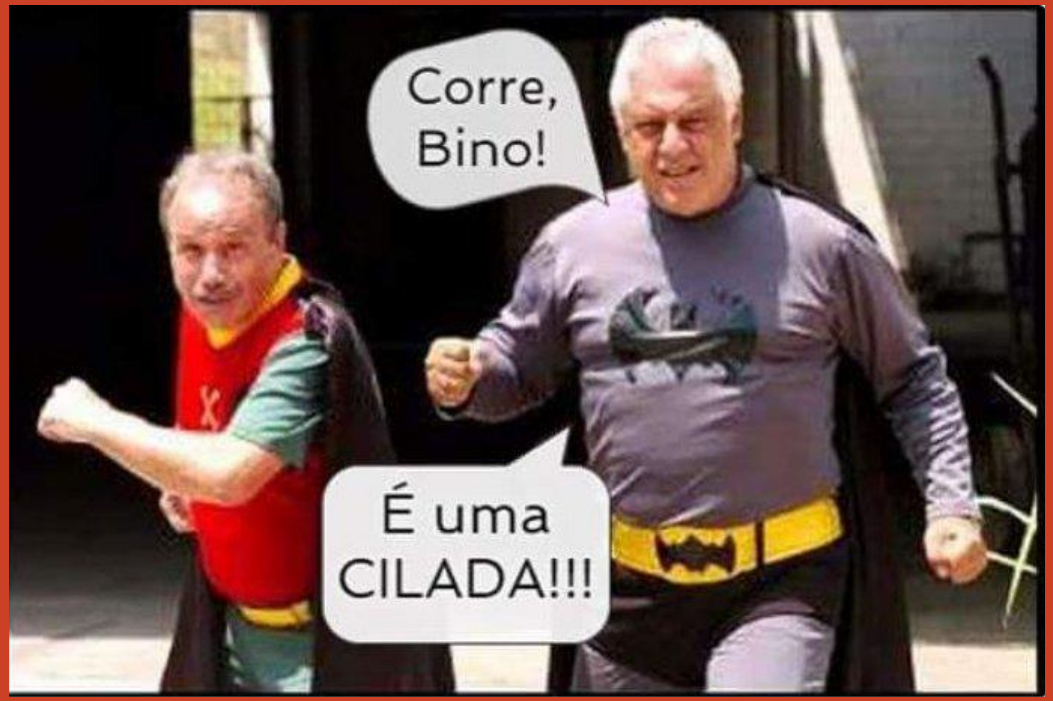
NodeJS

vs Outros Frameworks

Node.js é extremamente flexível e adaptável à necessidade da aplicação e sua ideia de estrutura de projetos.

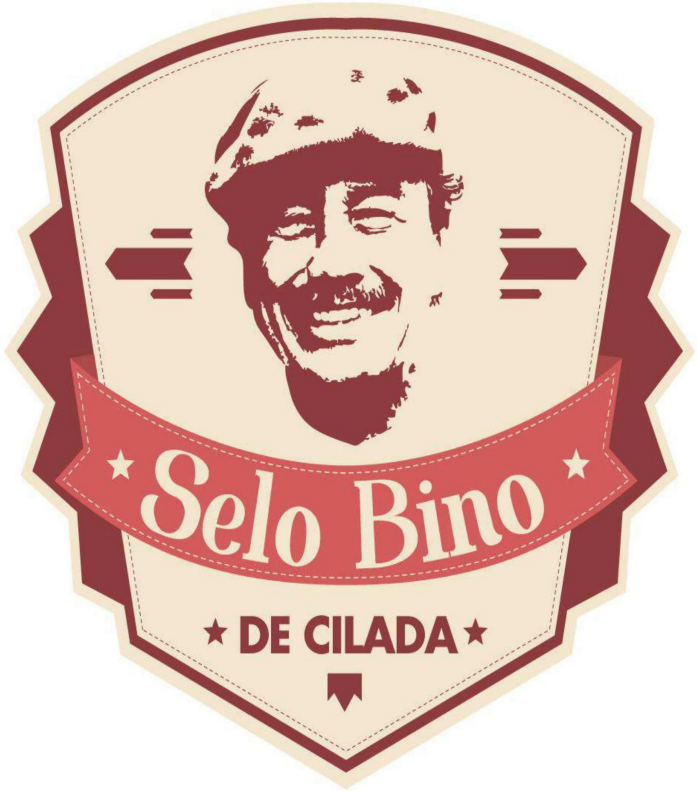
Anti-patterns

mais comuns em Node.js


Corre, Bino!
É uma CILADA!!!

1ª Cilada

Node.js é **JavaScript**...



Portanto, é **assíncrono**...

O que é algo muito legal, mas, nem sempre, principalmente no desenvolvimento **server-side**.

2ª Cilada

Promises e Callback Hell

```
1   api.getItem(1)
2     .then(item => {
3       item.amount++;
4       api.updateItem(1, item)
5         .then(update => {
6           api.deleteItem(1)
7             .then(deletion => {
8               console.log('done!');
9             })
10        })
11    })
```

Atenção!

Não é porque você vai usar **promisses** que não vai precisar ficar de olho no **callback hell.**

3ª Cilada

Tudo que vai, **volta**...


```
1   api.getItem(1)
2     .then(item => {
3       item.amount++;
4       api.updateItem(1, item);
5     })
6     .then(update => {
7       return api.deleteItem(1);
8     })
9     .then(deletion => {
10      console.log('done!');
11    })
```

Esquecer o return...

Design Patterns

Alguns tipos mais **populares**

Singleton especifica que apenas uma instância da classe pode existir, e esta será utilizada por toda a aplicação.

Dessa forma, temos apenas **um ponto de acesso** central a esta instância da classe.

```
//area.js
var PI = Math.PI;

function circle (radius) {
  return radius * radius * PI;
}

module.exports.circle = circle;
```

Quando usar?

Para **controlar** a concorrência de acesso a recursos compartilhados;

Para uma **classe** utilizada com frequência por várias partes distintas do sistema, e essa classe não gerencia nenhum estado da aplicação.

Observer **é um padrão de projeto de software que define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente.**

Observer é também chamado de Publisher-Subscriber, Event Generator e Dependents.

```
// MyFancyObservable.js
var util = require('util');
var EventEmitter = require('events').EventEmitter;

function MyFancyObservable() {
  EventEmitter.call(this);
}

util.inherits(MyFancyObservable, EventEmitter);
```

Quando usar?

Quando existe a necessidade de fazer com que um conjunto de objetos seja notificado e atualizado **automaticamente** após um determinado evento no sistema;

Cuidado: a desvantagem é que usar este padrão de forma indiscriminada pode causar sério impacto na **performance** do sistema.

Factory é um objeto responsável por criar e entregar outros objetos baseados nos parâmetros de entrada.

3 variações do padrão Factory

Simples

Permite interfaces para criar objetos sem expor a criação lógica para o cliente.

Método

Permite interfaces para criar objetos, mas permite subclasses para determinar qual classe instanciar.

Abstrato

Já uma fábrica abstrata é uma interface para criar objetos relacionados sem especificar/expor as suas classes.

```
function MyClass (options) {
  this.options = options;
}

function create(options) {
  // modify the options here if you want
  return new MyClass(options);
}

module.exports.create = create;
```

Vantagem

Factory (fábrica) permite-nos ter uma **localização centralizada** onde todos os nossos objetos são criados.

Injeção de dependência é quando uma ou mais dependências (ou serviços) são injetadas, ou passadas por referência, num objeto dependente.

```
function userModel (options) {
  var db;

  if (!options.db) {
    throw new Error('Options.db is required');
  }

  db = options.db;

  return {
    create: function (done) {
      db.query('INSERT ...', done);
    }
  }
}

module.exports = userModel;
```

Vantagens

Deixar seu código **desacoplado;**

Facilitar nos **testes** e **isolamento** dos componentes;

Aplicável em cenários onde podemos trocar o comportamento do sistema, utilizando classes que realizam as mesmas funcionalidades, porém de fontes diferentes.

```
function userModel (options) {
  var db;

  if (!options.db) {
    throw new Error('Options.db is required');
  }

  db = options.db;

  return {
    create: function (done) {
      db.query('INSERT ...', done);
    }
  }
}

module.exports = userModel;
```

Quando usar?

Quando existe a possibilidade de **trocar um comportamento** no sistema em determinado componente, ou quando você deseja **isolar** esses componentes para **testes**.

Lembre-se: um padrão é aplicado para um problema decorrente em determinados cenários. Se você não tiver essa necessidade, não tem o porquê aplicar esse padrão.

Um padrão de código ruim é **melhor** que nenhum

ESLint veio para te ajudar!

Lembre-se: se você não quer configurar o ESLint, utilize a fórmula já pronta de algumas empresas de sucesso.

Você pode adotar um padrão de projetos através de um framework confiável

ADONIS

Search docs

Quick Start Guides Recipes Sponsors Community 2

Adonis.js

It all boils down to confidence

Writing micro-services or you are a fan of TDD, it all boils down to confidence. AdonisJs simplicity will make you feel confident about your code.

Get started → Screencasts

Node.js web framework

AdonisJs is a Node.js web framework with a breath of fresh air and drizzle of elegant syntax on top of it. We prefer **developer joy** and **stability** over anything else.

sails v1.0

iniciar Apoio, suporte Documentaçã

Sails.js

Velas 1.0 chegou.

Confira o guia de atualização , ou criar um novo aplicativo .

iniciar

Confira os últimos desenvolvimentos no repositório do GitHub .

Star 19,968 Curtir 6,9 mi Tweet Follow @sailsjs

O Sails facilita a criação de aplicativos Node.js personalizados e de nível corporativo.

Crie aplicativos Node.js práticos e prontos para produção em questão de semanas, não meses.
Sails é a estrutura MVC mais popular para o Node.js, projetada para emular o padrão familiar de frameworks MVC como Ruby on Rails, mas com suporte para os requisitos de aplicativos modernos: APIs orientadas a dados com uma arquitetura escalável e orientada a serviços.

Boilerplate da comunidade

Obrigada!

Zalba Monteiro

zalbasmonteiro@gmail.com

LinkedIn /zalbamonteiro
GitHub /zalbamonteiro