



Boas práticas ✓

Material Complementar

Introdução

O que vamos ver?

O que eles tem em comum?

BEM (Block Element Modifier)

Para que serve?

BEM-BLOCK__ELEMENT--MODIFIER

Bloco:

Elemento 

Modificador 

Uso

Benefícios

Atomic Design

Documentação

O que é?

Divisão

Design System

Disclaimer

O que é?

Características

Material Design

Exemplo do Design System da Gama

Material Complementar

Storybook

O que é?

Características

Storybook

Documentação

Micro Frontends

O que são?

Características

Quais são benefícios

Exemplo

MVC (Model, View e Controller)

Diagrama

[Modelo \(Model\)](#)

[Visão \(View\)](#)

[Controlador \(Controller\)](#)

[MVVM Model-View-viewmodel](#)

[Documentos](#)

[O que é?](#)

[Diagrama](#)

[Modelo](#)

[Visão](#)

[ViewModel](#)

[Design Patterns](#)

[O que são?](#)

[Objetivos](#)

[SINGLETON → Esse é o tipo](#)

[STRATEGY](#)

[ADAPTER](#)

[Material Complementar](#)

Material Complementar

- [Podcast FalaDev](#)
- [Slides](#)

Introdução

O que vamos ver?

- ☐ BEM
- ☐ Design System
- ☐ Storybook
- ☐ Micro Frontends
- ☐ MVC (Model View Controller)
- ☐ MVVM (Model View ViewModel)
- ☐ Design Patterns: Singleton, Strategy e Solid

O que eles tem em comum?

- Padronizar algo para que todos saibam

- Fique mais fácil de escalar, entendimento e manutenção.
- Evita o retrabalho
- Essas metodologias evitam problemas como a falta de padrão e de organização.



Cada padrão resolve um tipo de problema específico, nenhum é bala de prata.

BEM (Block Element Modifier)

Para que serve?

- Melhora a organização no código CSS.

BEM-BLOCK__ELEMENT--MODIFIER

- Metodologia para ajudar o desenvolvedor a criar componentes reutilizáveis e facilmente compartilháveis dentro do Front-End
- Simples de ser implementado, pois basta seguir a convenção de nomes.
- Possível criar blocos modularizados dentro do seu código
- Pode ser utilizado com outros padrões e metodologias de organização
- Exemplos no getbem.com

Bloco:

- Elemento único representado por ele mesmo.
- ex: header, container, menu, checkbox, input

Elemento

- Trecho de um bloco que não tem um significado de forma isolada, mas sim associada ao bloco.
- ex: menu item, list item, checkbox caption, header title

Modificador

- Flags de mudança de comportamento ou aparência.

- ex: disabled, highlighted, checked, fixed, size big, color yellow

Se formos colocar mais classes no elemento HTML, **separe** essa classe CSS em outro bloco.

```

<!-- Block Element Modifier - Bloco_Elemento--Modificador -->

<!-- Padronização para
classes de elementos utilizamos 2x '__' (underline)
Ex: .list__item, .list__title

Para classes de modificadores utilizamos 2x '--' (traços)
Ex: .list__item--highlight, .list__author--active

O que é um bloco?
Basicamente um bloco é um container cujos conteúdos são elementos da marcação HTML.
Ex: -->

```

Uso

HTML	CSS
<pre> <button class="button"> Normal button </button> <button class="button button--state-success"> Success button </button> <button class="button button--state-danger"> Danger button </button> </pre>	<pre> .button { display: inline-block; border-radius: 3px; padding: 7px 12px; border: 1px solid #050505; background-image: linear-gradient(#EEE, #DDD); font: 700 13px/18px Helvetica, arial; } .button--state-success { color: #FFF; background: #569E3D linear-gradient(#790858, #569E3D) repeat-x; border-color: #4A993E; } .button--state-danger { color: #900; } </pre>

Benefícios

Modular

- Estilizando em escopo de bloco nunca dependem de outros elementos em uma página assim, você não terá problemas com herança ou especificidade dos elementos.

Reutilizável

- O comportamento de cada bloco deve ser independente e possibilita "N maneiras" de reutilizar em seu projeto, reduz a quantidade de CSS para dar manutenção.

- Com um conjunto de diretrizes de estilo, você pode construir uma biblioteca de blocos, tornando o CSS mais eficiente.

Estruturado

- O BEM deixa o seu CSS com uma estrutura mais limpa e de fácil entendimento.

Atomic Design

Documentação

- [Atomic Design](#)

O que é?

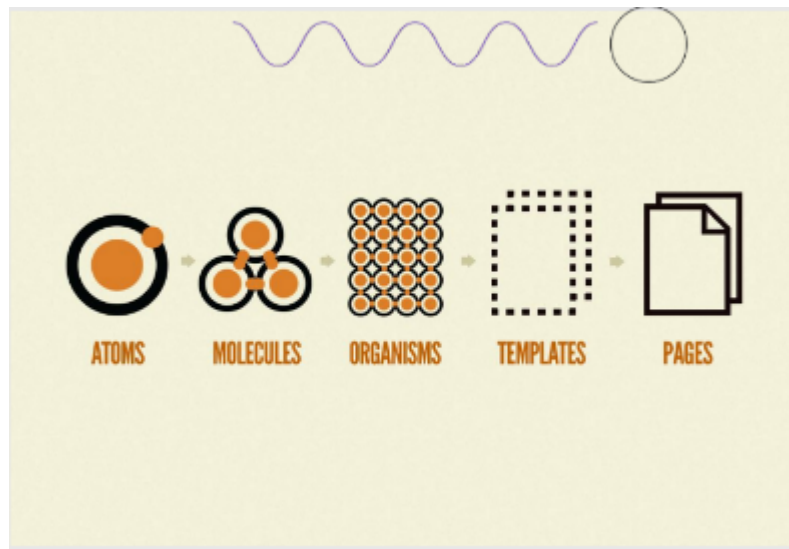
- Atomic Design foi criado em meados de 2013, ele tem como forte referência química para elaborar uma metodologia eficiente para utilização em design e interfaces
- E a essência do Atomic design é as referências de átomos e moléculas em suas divisões e composição criando elementos uniformes e que são muito utilizados em padrões de Design System.

Divisão

Ele é dividido em 5 partes ou componentes que juntos compõem interfaces ordenadas por hierarquias

- **Átomos**
 - Elementos ou blocos que formam a interface
 - Exemplo: elementos isolados, botões ou até mesmo inputs
- **Moléculas**
 - Grupo de elementos que formam de interface e funcionam de maneira agrupada.
 - Exemplo: um formulário composto por inputs e botões.
- **Organismos**
 - Conjunto de moléculas que tratam um contexto de interface.

- **Templates**
 - São elementos do nível de páginas onde no formato de componentes formando a estrutura de páginas.
- **Páginas**
 - E o conjunto dos elementos mencionados formando o resultado final.



Design System

Disclaimer

- Antes de qualquer coisa, o Design System é um grande aliado quando o assunto é estruturar produtos digitais, atuar com gestão e padronização de elementos para escalar e atualizar.
- O uso do Design System não se limita apenas a elementos web, mas se estende a elementos Mobile podendo ser agnóstico de frameworks ou até mesmo linguagens de programação.

O que é?

- Ele é um conjunto de **padrões de design** ou componentes preestabelecidos e consensuais.

- Com vários elementos formamos componentes.
- A ideia do Design System é fazer o reuso de tudo que já está pronto
- Imaginamos como pecinhas de lego que padroniza a identidade do seu produto
- É algo antigo e foi originado no Component Based Development em 1970
- Trata-se de uma documentação composta por elementos e informações associadas a marca onde o mesmo deve ser implementado, ou seja tokens com medidas de espaçamento, cores e tom de voz da marca.
- Onde se aplica até mesmo a bibliotecas e padronização de componentes.

Características

- Documento vivo de design com todos os componentes de um software
- Tem por objetivo aumentar a eficiência dos designers no momento de especificação
- Trás consistência para os usuários, inclusive desenvolvedores com uma linguagem clara e unificada
 - Já tá pronto, só importar e fazer o uso
 - Ajuda o dev na hora de criar e implementar os componentes, pois já estão todos especificados no **Design System**
- Deve ter um objetivo claro no momento de sua criação

Material Design

Material Design. Criado e mantido pelo Google o mesmo é utilizado em todos os seus produtos e atualmente adotado por muitas outras empresas.

O que é?

- Uma biblioteca Disponível para os frameworks mais utilizados do mercado que possibilita a utilização de elementos padronizados.
- **Ex:** Botões, Inputs, Cards, Tooltips e muitos outros elementos.
- Todos os estes elementos estão devidamente encapsulados nesta biblioteca facilitando o uso e escala em produtividade.
- Link para o [Material Design](#)

Exemplo do Design System da Gama

Documento do Design System

🔍 Type to search...


Components

Getting Started

Consistency

Readme

Getting Started



Design systems enable teams to build better products faster by making design reusable—reusability is the primary value of design systems. A design system is a collection of reusable components, guided by c together to build any number of applications.

Regardless of the technologies and tools behind them, a successful design system follows these guidi

- **It's consistent.** The way components are built and managed follows a predictable pattern.
- **It's self-contained.** Your design system is treated as a standalone dependency.
- **It's reusable.** You've built components so they can be reused in many contexts.
- **It's accessible.** Applications built with your design system are usable by as many people as possible
- **It's robust.** No matter the product or platform to which your design system is applied, it should perf

Lista do Design System da Gama

QUEROOO!!

<Button text="Quero!!" onClick={() => alert('alo')} />

text	String
smallText	String
onClick	Func
colorProfile	'primary' 'secondary' 'neutral' "primary"
disabled	Bool

Components

BulletIcons

Buttons

Card

Carousel

Contents

CourseCatalog

Forms

Headers and Footers

Links

Lists

Loading

Menus

Navigation

Sections

Slidebar

Texts

Titles

Video

Material Complementar

- Design System na prática pro Designers e Devs

Storybook

O que é?

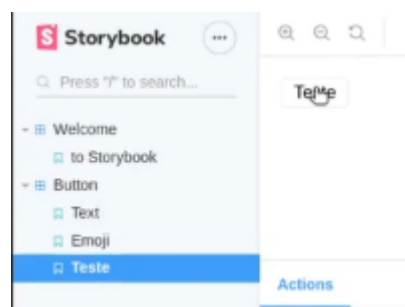
- Contar uma história do seu **Front-End**
- Dentro Storybook podemos ter o Design-System, como foi o caso da Gama Academy.

Características

- Ferramenta para preaparar o ambiente de desenvolvimento de **componentes de UI**
- Permite desenvolver de forma isolada componentes de interfaces
- Integrado os principais frameworks front-end utilizados (VueJS, Angular, React)
- Possibilidade de trabalhar com addons, adicionando mais funcionalidades
- Exibe na interface erros de sintaxe

Storybook

- Conseguimos criar nossos componentes e nosso próprio Design System com essa interface do Storybooks.
- Passamos uma função com vários tipos de componentes para a documentação



Documentação

- Storybook

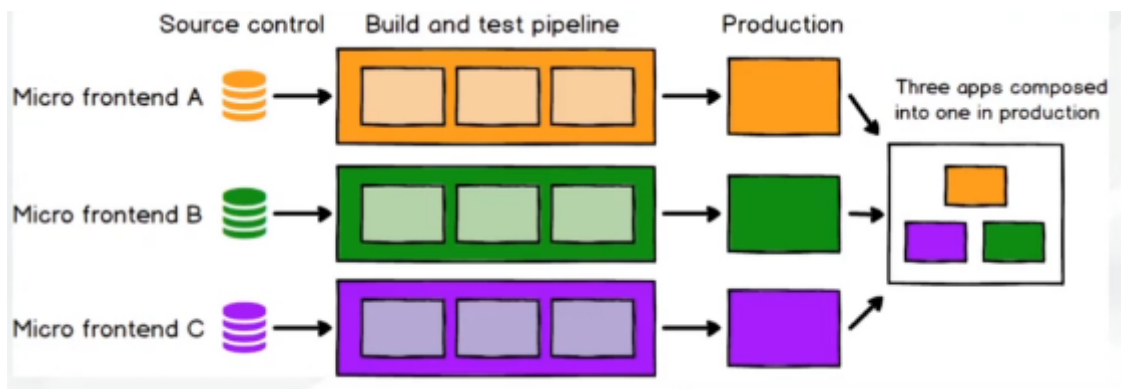
Micro Frontends

O que são?

- Começou a ganhar notoriedade em 2016, na TofWorks.
- Assim como temos microsserviços, que é separar nossos serviços em partes pequenos e independentes e disponibilizamos eles como serviços, temos no Front-end o Micro-Frontend
- Ele é um método onde é possível mesclar frameworks e tecnologias em uma mesma aplicação, possibilitando e auxiliando trabalho de times em features ou até mesmo funcionalidades de produtos de maneira isolada e contida, vamos entender mais exemplos a seguir.

Características

- Evolução natural no desenho de arquitetura de software
- Aprofundamento em martinfowler.com/artciles/micro-frontend.html
- Criar provas de conceito em single-spa.js.org
- Aceita várias tecnologias, React, Vue etc. simultaneamente.
- Não precisa estar vinculado a uma linguagem.



Quais são benefícios

- Desacoplados de tecnologia
- Merge reduzido, e conflito de código também
- Github não precisa ser dividido.
- Independência e agilidade no desenvolvimento do Software
- Se houver problema, é um problema desacoplado, o site não irá quebrar por conta dele.

Exemplo

- Temos nossos componentes feitos com vários Frameworks
- Ao lado temos uma imagem ilustrando a composição de uma página utilizando libs e frameworks distintos na mesma página.
- Exatamente esta é a proposta do Micro Frontend oferecer maior autonomia para o desenvolvimento e escala por squads (formação de equipes para atuar com desenvolvimento).
- São infinitas as possibilidades para uso deste formato, tanto na entrega de melhorias quanto na evolução de cada feature, imagine que no ato de publicar cada atualização os módulos são completamente isolados, ou seja não afetam o funcionamento dos demais.

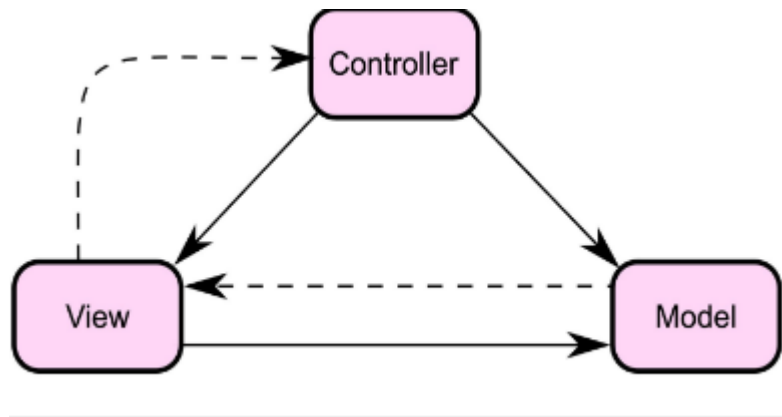


MVC (Model, View e Controller)

- MVC é uma arquitetura de solução que tem por conceito padrão a divisão do serviço em 3 camadas. **MODEL, VIEW e CONTROLLER.**
- Ele foi formulado por volta da década de 70 e seus conceitos são utilizados até hoje, por ser um grande facilitador na divisão de responsabilidades de uma aplicação.
- O seu uso tem como premissa, dividir níveis interconectados como interação com banco de dados e respostas para os usuários.
- Enquanto numa aplicação tem o Front-End e Back-end, a nível de sistemas temos a divisão de **responsabilidade/camadas** entre Model, View e Controller.

Diagrama

- Este diagrama tem como objetivo expor de maneira simples as relações entre as camadas, trazendo um pouco mais de clareza quanto ao conceito como um todo.



Modelo (Model)

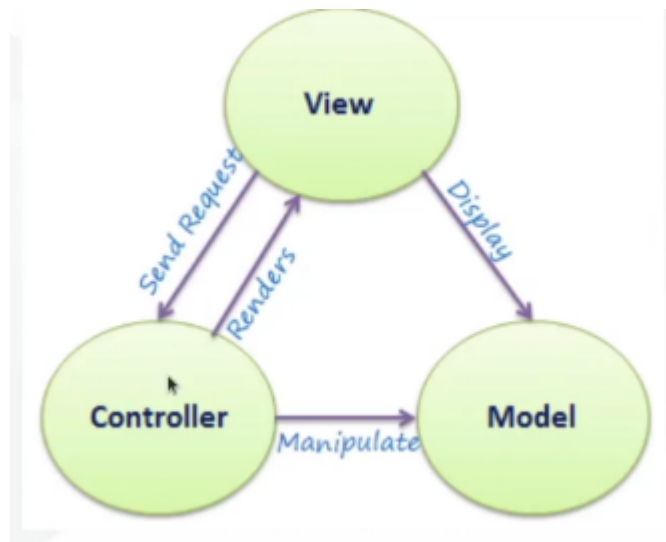
- Gerencia as Entidades do nosso sistema
- Lida com informações da nossa aplicação
- Receber, tratar e validar todos os dados
- Conectar a base de dados

Visão (View)

- Camada de interação do usuário com o sistema
- Renderiza componentes que fazem parte da experiência da aplicação

Controlador (Controller)

- Recebe requisições e eventos do usuário, tratar e respondê-las
- Requer ao Model os dados necessários
- Encaminha resposta do Model para a View
- Captura dados na View e encaminha para o Model
- Seria onde estaria nossas regras de negócio
- Aqui que também temos a conversão de nossos dados para o JSON ou outra linguagem que tanto nossa View e Model entendam
- Uma **API** seria o nosso Controller.



MVVM Model-View-viewmodel

Documentos

- [Microsoft MVVM](#)

O que é?

- É um padrão arquitetural de desenvolvimento que auxilia na separação do desenvolvimento da interface gráfica do usuário sendo por meio de linguagens de marcação ou até mesmo por código de toda a camada de lógica.
- Desenvolvido pela Microsoft por volta de 2005 trata-se de uma variação do padrão de projeto Presentation Model e arrisco em dizer que este padrão **"é o primo mais novo do MVC"**.
- Este padrão é utilizado por diversos frameworks, tanto no desenvolvimento mobile, quanto no desenvolvimento WEB:

Mobile

- Java, Swiftkl, Flutter, Xamarin Forms... Entre outros.

WEB

- Angular, VueJS, Ember... Entre outros.

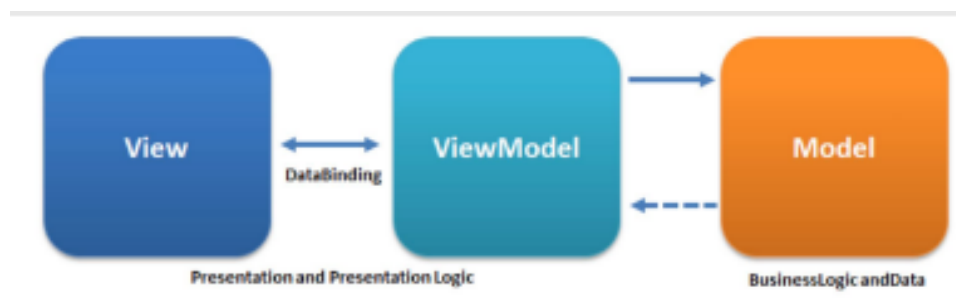
Diagrama

- Conforme o diagrama ao lado, podemos enxergar o conceito de maneira simples.

- Principal conceito por de trás desta solução é o **desacoplamento de código** trazendo muitos benefícios para o desenvolvimento.

Componentes do padrão MVVM:

1. Modelo
2. Visualizador
3. Ver modelo
4. Encadeador



Modelo

- Regras de negócio
- Encapsula dados
- Prover notificações através da interface (INotifyPropertyChanged)

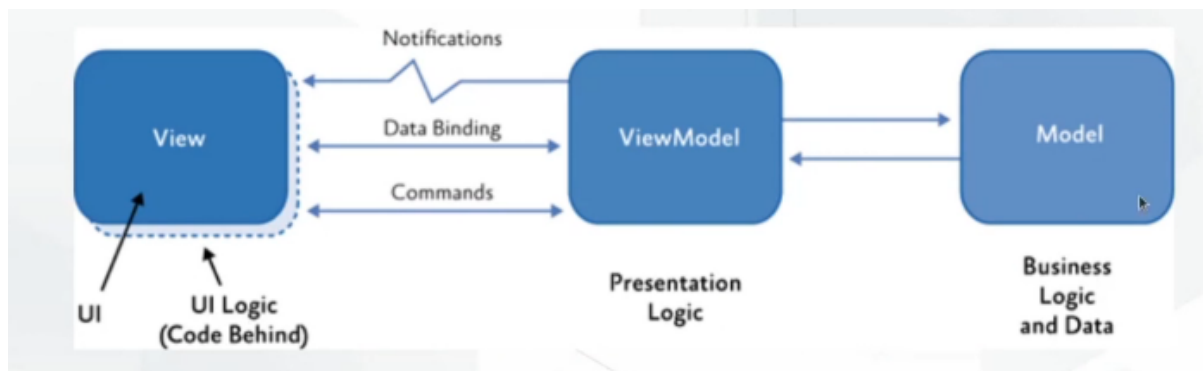
Visão

- Camada de interação do usuário com a aplicação
- Renderiza componentes que fazem parte da experiência da aplicação
- Define aparência e estrutura do que o usuário visualiza na tela
- Normalmente contém um Code-Behind sobre a lógica dessa interface
- Deve possuir um Binding Context indicando qual ViewModel está referenciada

ViewModel

- Trata da Lógica de controles
- Não conhece a View

- Lança Notificações de estado ou de alterações de estado (OnNotifyPropertyChanged)



Design Patterns

O que são?

- Design patterns são padrões de desenho ou padrões de projeto.
- É um **padrão de design** para projetos.
- Muito a ver com Engenharia de Software como um todo e POO.
- Para ser mais claro, **um padrão de projeto encapsula alguns parâmetros importantes**, sendo assim é composto por 4 elementos:
 1. **Nome do padrão;**
 2. **Problema a ser resolvido;**
 3. **Solução fornecida pelo padrão;**
 4. **Consequência;**

Objetivos

Por tanto os padrões de projetos (Design Patterns) tem como objetivo:

1. **Facilitar a reutilização de soluções (códigos ou afins);**
2. **Estabelecer um vocabulário entre o desenho e solução;**



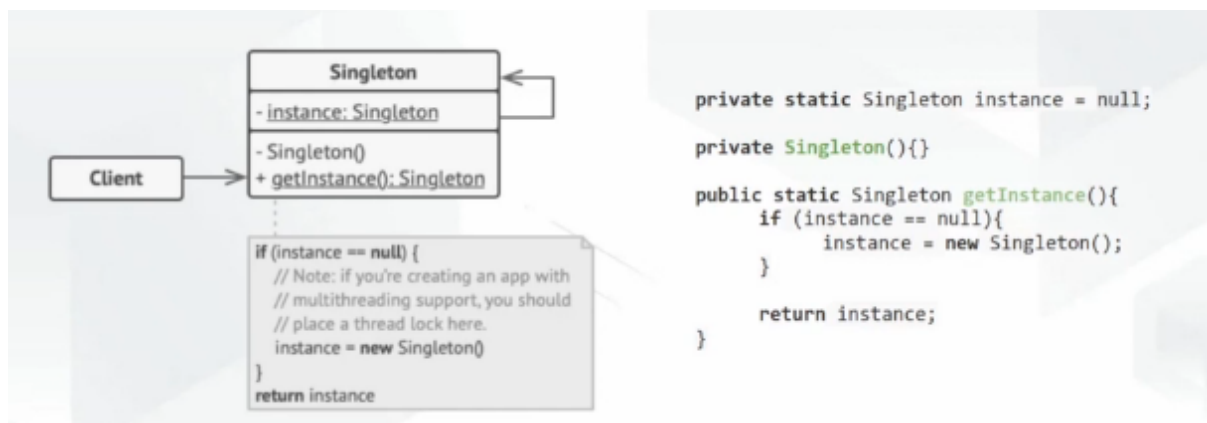
Dentro de cada **tipo** tem diversos outros **padrões**.

SINGLETON → Esse é o tipo

- Padrões de projeto do tipo **CREATIONAL**
- Garante que existe apenas uma instância de uma classe
- Provê acesso global por toda a aplicação em um único ponto
- Exemplo prático é quando criamos um Controller no Back-End e exportamos para toda a aplicação, inclusive na parte das **rotas**.
 - Importamos no arquivo o Controller e chamamos a função dentro dele
 - Não precisamos instanciar sempre quando formos utilizar.

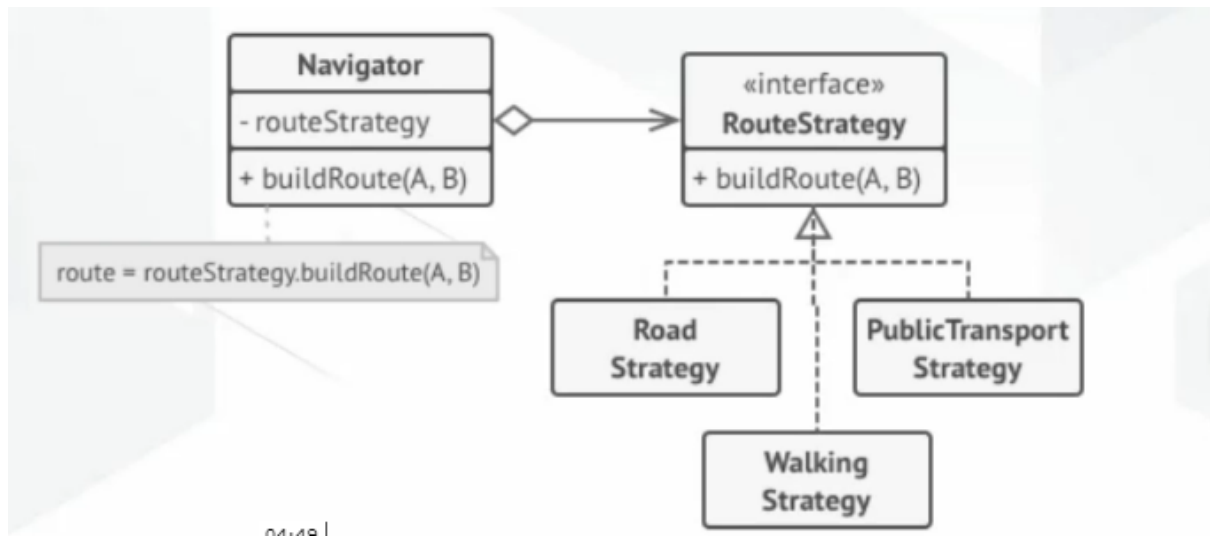
```
export default new MeetupController();
```

```
routes.post('/meetups', MeetupController.getInstance());
```



STRATEGY

- Padrão do tipo **BEHAVIORAL (COMPORTAMENTAL)**
- Pode definir uma série de algoritmos diferentes, cada um separado em suas devidas classes, mas que pode ser intercambiáveis
- Vamos ao exemplo em
 - Todos os formatos usam o Route.
 - Quebramos nossa aplicação em pedaços menores, visando ajudar a legibilidade e reutilização da função em vários lugares
 - Separamos a responsabilidade de cada coisa.



ADAPTER

- Padrão de projeto tipo **STRUCTURAL (ESTRUTURAL)**
- Serve para adaptar nossa implementação já existente a um novo cenário diferente daquele originalmente desenhado
- Ajuda muito quando precisamos utilizar integrações com serviços de terceiros (WebService do governo por exemplo)

Material Complementar

- [Código Fonte](#)