

Crie seu Primeiro App Mesmo sem saber nada

{odifiq}

V2.0

O guia definitivo
Para iniciantes criarem suas
aplicações do ZERO e entrar
de uma vez na área de software



ÍTALO PAIVA

O que saber antes de ler este e-book

Antes de mais nada eu queria te avisar para **NÃO SE ASSUSTAR** com o tamanho do e-book.

Ele ficou um pouco grande porque eu fiz questão de te dar realmente um **passo a passo para criar o seu app** e explicar cada passo do **jeito mais completo e simples possível** e com **bastantes exemplos**.

Por isso eu coloquei vários prints da minha tela ao longo do texto para exemplificar visualmente o que você deveria encontrar e facilitar o seu entendimento.

Vá em frente, dê uma “folheada” aí até o final do e-book e veja que tem bastantes imagens! **As imagens são as reais culpadas pelo tamanho do e-book.** Mas isso é uma coisa boa, aproveite!

Isso nos leva ao segundo ponto: **o e-book foi pensado para iniciantes.** Esse e-book é uma grande introdução ao mundo do desenvolvimento de software, por isso eu quis deixar tudo bem explicado para que você não se sinta perdido(a).

No final desse e-book você vai sair com um app que vai estar rodando na internet e uma boa bagagem de conhecimento.

Apesar de ser um app simples, você vai ter praticado vários conceitos importantes na programação e ainda vai conhecer ferramentas profissionais de desenvolvimento.

Terceiro ponto: esse **NÃO** é um e-book só de leitura. Se você está realmente comprometido a entrar nessa área e aprender de fato alguma coisa, **você vai ter que praticar** o que está descrito aqui.

Vou descrever brevemente os capítulos do e-book para você ter uma ideia do que vai encontrar.

As primeiras páginas de conteúdo eu dedico para te contextualizar no mundo do desenvolvimento web e mobile na atualidade, tanto do ponto de vista de necessidades do desenvolvimento quanto do ponto de vista ferramental.

No capítulo “Parte 1” eu te explico o que é Ambiente de Desenvolvimento, quais ferramentas você vai precisar para criar o seu app e como configurar o ambiente de desenvolvimento na sua máquina, que é uma das primeiras coisas que você precisa fazer para começar a programar.

No capítulo “Parte 2” eu te mostro como instalar o Quasar, que vai ser o framework que nós vamos usar para nos ajudar a criar o app. O Quasar é uma ferramenta muito poderosa que utiliza o Vue.js por trás e que traz muitas facilidades no desenvolvimento. É ótimo para quem está começando!

No capítulo “Parte 3” eu te mostro como usar a linha de comando do Quasar para criar a estrutura do projeto.

No capítulo “Parte 4” eu faço uma breve explicação sobre as 3 linguagens principais para se desenvolver para a web (que nós vamos utilizar ao longo do e-book), o HTML, o CSS e o JavaScript. Além disso eu coloquei links para uma série de vídeos mais densa

sobre cada uma dessas linguagens lá para quem deseja se aprofundar e entender melhor.

No capítulo “Parte 5” nós vamos explorar o projeto que foi criado pelo Quasar, para você começar a entender. O Vue.js (que é usado pelo Quasar) possui uma arquitetura específica de componentes e por isso eu gastei esse capítulo para explicar como isso funciona antes de partir para a criação de fato do app.

No capítulo “Parte 6” eu te mostro como usar o Quasar, principalmente como usar a documentação dele ao seu favor.

Como eu falei, esse e-book foi pensado para iniciantes, por isso que até agora nós só estávamos nos preparando para começar a criação do nosso app. **Não adianta nada você querer começar a programar rápido se não vai entender nada** que estiver fazendo, não é mesmo?

No capítulo “Parte 7” nós vamos criar um app simples de lista de tarefas para colocar em prática todos os conceitos vistos.

No capítulo “Parte 8” nós vamos pegar o app construído e subir ele em um servidor na nuvem para podermos acessar a partir de um celular. Nesse capítulo eu te apresento também o conceito de PWA (Progressive Web Apps), que é uma tendência no mundo web e mobile. Você vai ver que com uma PWA a experiência de uso do app é igual a de um nativo!

Centenas de pessoas já baixaram esse e-book e eu espero que ele te ajude da mesma forma como ajudou essas outras pessoas!

Vamos lá, já pode começar!

Índice

Sobre o autor	5
Desenvolvimento mobile na atualidade	6
Ecossistema de ferramentas para aplicativos híbridos	8
Parte 1 - Ambiente de desenvolvimento	10
Parte 2 - Instalando o Quasar	11
Parte 3 - Criando o projeto do Aplicativo	13
Parte 4 - As três bases do desenvolvimento web	21
Parte 5 - Explorando o projeto criado	25
Parte 6 - Como usar o Quasar	36
Parte 7 - Criando o seu app	47
Parte 8 - Rodando o seu app	82
Conclusão	91
Desafios	92
Contato	94

Sobre o autor



Meu nome é Ítalo Paiva e sou um Engenheiro e amante de Software.

Tenho uma paixão imensa por ensinar o que aprendi nesses anos de experiência desenvolvendo softwares por esse mundo afora.

Minha missão agora é transformar vidas através da programação, porém de uma forma simples, fácil e eficiente.

Por isso eu criei a [Codifiq!](#)

Neste e-book eu quero te mostrar que criar um aplicativo nos dias de hoje não é uma tarefa de outro mundo, porque temos muitas ferramentas para nos auxiliar.

Essa já é a terceira versão deste e-book que está ainda mais simples e fácil!

Centenas de pessoas já baixaram esse e-book e eu espero que ele te ajude assim como ajudou essas pessoas.

Qualquer dúvida e para mais conteúdos, você me acha em:



<https://www.instagram.com/codifiq/>



<https://www.facebook.com/codifiq>



<https://codifiq.com.br/>

Desenvolvimento mobile na atualidade

Hoje em dia passamos cada vez mais tempo vidrados na telinha dos nossos celulares, não é verdade?! As crianças praticamente já nascem dominando essa tecnologia.

A popularização das tecnologias móveis e os benefícios e comodidades proporcionados por essas tecnologias criam uma demanda crescente de mão de obra capacitada para a criação de softwares para essas plataformas.

Atualmente, temos três plataformas em destaque na tecnologia mobile: o Android da Google, o iOS da Apple e o Windows Phone da Microsoft.

Cada uma dessas plataformas tem suas particularidades técnicas. Por exemplo, cada uma utiliza uma linguagem de programação diferente: O Android usa Java e/ou Kotlin, o iOS usa Swift e o Windows Phone usa C#.

Se você quer criar um aplicativo, provavelmente vai querer que ele esteja disponível para todas as plataformas, porque aí tem mais alcance, não é?!

Então, para criar aplicativos mobile temos basicamente duas opções:

- criar um aplicativo para cada plataforma, o que implica em no mínimo **três projetos diferentes**, o que gasta MUITO tempo de desenvolvimento e requer um time com pessoas capacitadas em **TODAS** as linguagens;

- ou criar um **aplicativo híbrido**, para que com apenas **UM PROJETO** você consiga atingir os usuários de todas as plataformas.

Para não cair em especificidades de cada plataforma, os **aplicativos híbridos** se baseiam em algo que todas as plataformas têm em comum, algo que todas as plataformas entendem: **a web**.

Todas as plataformas conseguem interagir com a web, então construímos aplicativos com as mesmas tecnologias utilizadas na web: **HTML, CSS e Javascript**.

Nesse e-book vamos te ensinar a criar o código de um aplicativo híbrido utilizando apenas tecnologias web.

Vamos lá!

Ecossistema de ferramentas para aplicativos híbridos

Existem várias tecnologias capazes de construir aplicativos híbridos, como React Native, Native Script, Ionic, Meteor, etc.

Quero deixar uma coisa bem clara: **não** estou aqui para levantar nenhuma bandeira e defender tecnologia X ou Y, nem vou entrar no mérito de dizer qual é a melhor ou pior.

Vou apenas te mostrar uma das milhares opções que existem, que é uma opção que eu acho bem bacana para quem está começando e que também queira evoluir na área.

Nós vamos utilizar o [Quasar](#), que é uma tecnologia que tem uma proposta um tanto quanto ambiciosa: “Construir websites responsivos, PWAs, aplicativos híbridos (que parecem nativos!) e aplicações desktop, tudo simultaneamente utilizando a mesma base de código, utilizando Vue.js”.

Por que o Quasar?! Primeiro porque ele utiliza o Vue.js, que é uma tecnologia bastante poderosa, versátil e simples que usa a linguagem de programação Javascript.

Em segundo, o Quasar possui uma paleta de componentes prontos para uso que segue o guia de estilos da Google, o [Material Design](#).

Com esses componentes prontos para uso, fica mais fácil e rápido criar aplicativos utilizando esse framework.

Por isso, para quem está iniciando, ele é uma boa pedida! E para quem não está começando, você tem todo o poder do Vue.js para sustentar e evoluir sua aplicação.

Parte 1 - Ambiente de desenvolvimento

Para desenvolver um software uma das primeiras coisas que temos que fazer é preparar o **ambiente**.

O ambiente no desenvolvimento de software significa toda a infraestrutura necessária para o seu software rodar perfeitamente.

O Quasar é um framework construído em JavaScript, por isso para usar ele nós precisamos instalar o [Node.js](#).

O Node.js é uma plataforma que nos permite executar código JavaScript no lado do servidor.

O comum era rodar código JavaScript somente no navegador, ou seja, do lado do cliente. E com o Node.js o JavaScript começou a ser entendido no servidor e não só no navegador.

Em outras palavras, o Node.js é como se fosse um grande interpretador de JavaScript. E ele foi construído em cima do mesmo interpretador que roda o JavaScript no Google Chrome, a engine V8.

Na [página inicial do Node.js](#) tem instruções de instalação para todos os sistemas operacionais (Linux, Mac e Windows)

Como a maioria das pessoas acabam utilizando o Windows no dia a dia, para facilitar eu fiz um [vídeo mostrando como instalar o Node.js, o Quasar e o editor de texto Sublime Text no Windows](#).

Pronto! Se você seguir essas instruções, o seu ambiente de desenvolvimento vai estar no jeito!

Parte 2 - Instalando o Quasar

Quando vamos instalar alguma ferramenta, o primeiro passo é olhar na documentação da ferramenta qual o processo e os pré-requisitos necessários para a instalação. Assim como um produto qualquer que a gente compra, para saber como usá-lo nós lemos o manual de uso. Em software acontece da mesma forma.

Então, vamos olhar a documentação do Quasar, para ver o que é preciso para instalá-lo:

[Documentação de instalação do Quasar](#)

Quasar CLI Installation

Make sure you have Node >=8 and NPM >=5 installed on your machine.

```
# Node.js >= 8.9.0 is required.  
  
$ yarn global add @quasar/cli  
# or  
$ npm install -g @quasar/cli
```

Documentação da Instalação do Quasar

E aqui nessa página de instalação você vai ver que tem um comando para rodar e instalar o Quasar.

Ítalo, aonde eu executo esses comandos?!

E eu te respondo: **no terminal!**

O terminal é a famosa telinha preta dos filmes, onde os hackers supostamente estão executando seus ataques.

Mas falando sério, o terminal é uma **ferramenta fundamental no desenvolvimento de software**, pois conseguimos fazer qualquer coisa que um computador faz através de um terminal (por isso que os hackers amam a telinha preta!).

Na verdade, as telinhas bonitinhas que vocês veem são só “atalhos” para chamadas de comandos que podem ser executados em um terminal.

Se você estiver usando algum Linux ou um Mac é só procurar pela aplicação chamada “Terminal”.

Se você estiver usando o Windows e instalou o Node.js nele, você vai rodar esses comandos no **Prompt de Comando** (pesquise por “cmd” no menu do Windows).

Agora vamos instalar a CLI do Quasar para poder criar o nosso projeto. Rode esse comando no terminal:

```
npm install -g @quasar/cli
```

Ao rodar esse comando, o gerenciador de pacotes do Node.js (o NPM (Node Package Manager)) vai baixar a linha de comando do Quasar e todas as dependências necessárias.

Por isso, talvez demore um pouco para terminar esse comando, mas é só esperar!

Nosso ambiente está pronto! Agora vamos criar o nosso projeto!

Parte 3 -

Criando o projeto do Aplicativo

Agora nós vamos utilizar a linha de comando do Quasar que instalamos no passo anterior para criar a estrutura do nosso projeto:

```
quasar create INFORME_O_NOME_DO_SEU_PROJETO
```

Você precisa dar um nome para o seu projeto. **Troque “INFORME_O_NOME_DO_SEU_PROJETO” no comando acima por um nome que você queira dar para o seu projeto.**

No exemplo abaixo nós chamamos de “meu_app”. Ah, não use espaços nem caracteres especiais (como acentos) no nome do seu projeto. Ao invés de espaços, utilize underlines.

```
quasar create meu_app
```

Criando um projeto chamado “meu_app” com o Quasar

Ao rodar esse comando, o Quasar vai te perguntar algumas informações sobre o seu projeto.

As primeiras perguntas são mais descritivas sobre o projeto, como nome do projeto, descrição, autor, etc. Responda como quiser essa primeira parte.

```
* quasar create meu_app
projects quasar create meu_app
-----
| - \ - - - / - / - | ' - |
| - | | - | ( - ) \ ( - | |
\ - \ - / - / - | - |

Primeiras perguntas. Responda como quiser.

? Project name (internal usage for dev) meu_app
? Project product name (must start with letter if building mobile apps) Meu primeiro app
? Project description Meu primeiro app
? Author Italo Paiva <italopaiva@codifiq.com>
? Pick your favorite CSS preprocessor: (can be changed later) (Use arrow keys)
  ↘ Sass with indented syntax (recommended)
    Sass with SCSS syntax (recommended)
    Stylus
    None (the others will still be available)

Pergunta sobre Pré-processadores de CSS
Escolha a primeira opção, não vamos usar isso agora.
```

Primeiras perguntas do Quasar para criação do Projeto

A próxima pergunta será sobre a estratégia de importação dos componentes do Quasar. O jeito mais simples para desenvolver é falar para o Quasar fazer a importação automática. Dessa forma nós só usamos os componentes e o Quasar importa eles automaticamente pra gente. Por isso escolha a opção “Auto-import”.

```
x
projects quasar create meu_app
-----
? Project name (internal usage for dev) meu_app
? Project product name (must start with letter if building mobile apps) Meu primeiro app
? Project description Meu primeiro app
? Author Italo Paiva <italopaiva@codifiq.com>
? Pick your favorite CSS preprocessor: (can be changed later) Sass
? Pick a Quasar components & directives import strategy: (can be changed later) (Use arrow keys)
  * Auto-import in-use Quasar components & directives
    - slightly higher compile time; next to minimum bundle size; most convenient
  * Manually specify what to import
    - fastest compile time; minimum bundle size; most tedious
  * Import everything from Quasar
    - not treeshaking Quasar; biggest bundle size; convenient
      Escolha a opção "Auto-Import"
```

Pergunta sobre a estratégia de importação dos componentes

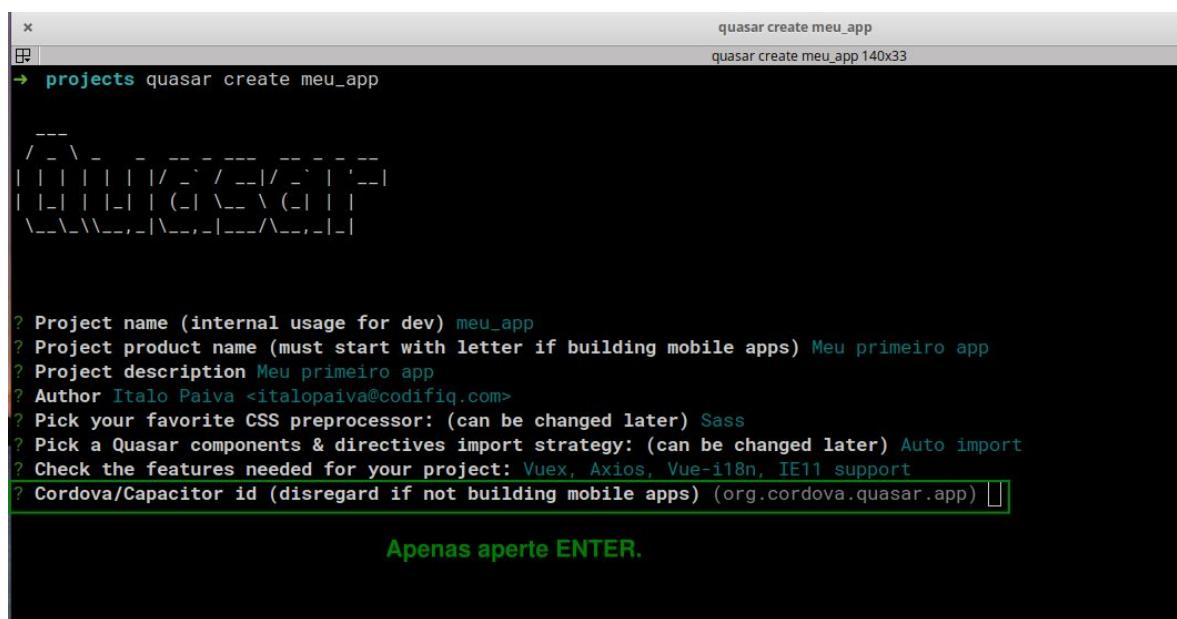
As próximas perguntas serão mais técnicas. O Quasar vai te perguntar quais ferramentas de suporte ao desenvolvimento você vai querer usar. Para começarmos, faça como está na imagem abaixo.

```
x
projects quasar create meu_app
-----
? Project name (internal usage for dev) meu_app
? Project product name (must start with letter if building mobile apps) Meu primeiro app
? Project description Meu primeiro app
? Author Italo Paiva <italopaiva@codifiq.com>
? Pick your favorite CSS preprocessor: (can be changed later) Sass
? Pick a Quasar components & directives import strategy: (can be changed later) Auto import
? Check the features needed for your project:
  ○ ESLint
  ● Vuex
  ● Axios
  ● Vue-i18n
  ● IE11 support
      Perguntas sobre as ferramentas necessárias no projeto
      Desmarque o "ESLint" e marque todas as outras opções.
      Para marcar e desmarcar use a tecla "espaço".
```

O “ESLint” é uma excelente ferramenta para nos ajudar a seguir boas práticas de código e escrever um código padronizado. Mas como você está começando, acho que ele vai mais te atrapalhar do que te ajudar. Por isso a gente **não** vai instalar ele.

As outras ferramentas como “Vuex”, “Axios” e “Vue-i18n” nós não vamos usar nesse e-book, mas eu falei para você instalar caso queira evoluir o projeto no futuro (vai que tem uma segunda versão desse e-book, né?!).

A próxima pergunta será sobre o identificador do app para o Cordova. Não se assuste! Não vamos entrar nesse quesito por agora, portanto apenas aperte ENTER e prossiga a criação do projeto.



```
quasar create meu_app
quasar create meu_app 140x33
→ projects quasar create meu_app

? Project name (internal usage for dev) meu_app
? Project product name (must start with letter if building mobile apps) Meu primeiro app
? Project description Meu primeiro app
? Author Italo Paiva <italopaiva@codifiq.com>
? Pick your favorite CSS preprocessor: (can be changed later) Sass
? Pick a Quasar components & directives import strategy: (can be changed later) Auto import
? Check the features needed for your project: Vuex, Axios, Vue-i18n, IE11 support
? Cordova/Capacitor id (disregard if not building mobile apps) (org.cordova.quasar.app) []
```

Apenas aperte ENTER.

Pergunta sobre configuração do Cordova

A próxima pergunta é sobre qual gerenciador de dependências o Quasar vai utilizar. Escolha o NPM.

```

x
          quasar create meu_app
          quasar create meu_app 140x33
projects quasar create meu_app

```
? Project name (internal usage for dev) meu_app
? Project product name (must start with letter if building mobile apps) Meu primeiro app
? Project description Meu primeiro app
? Author Italo Paiva <italopaiva@codifiq.com>
? Pick your favorite CSS preprocessor: (can be changed later) Sass
? Pick a Quasar components & directives import strategy: (can be changed later) Auto import
? Check the features needed for your project: Vuex, Axios, Vue-i18n, IE11 support
? Cordova/Capacitor id (disregard if not building mobile apps) org.cordova.quasar.app
? Should we run `npm install` for you after the project has been created? (recommended) (Use arrow keys)
❯ Yes, use Yarn (recommended)
 Yes, use NPM
No, I will handle that myself
```

```

Escolha o NPM como gerenciador de dependências

Escolha o NPM como gerenciador de dependências

Se tudo deu certo, você verá uma tela parecida com a da imagem abaixo.

```

x
italopaiva@italopaivapc:~/projects
Testing binary
Binary is fine
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN sass-loader@8.0.2 requires a peer of sass@1.3.0 but none is installed. You must install peer dependencies yourself.
npm WARN sass-loader@8.0.2 requires a peer of fibers@>= 3.1.0 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules/webpack-dev-server/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: "x64")
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules/watchpack/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: "x64")
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: "64")
added 1482 packages from 710 contributors and audited 14914 packages in 40.044s
found 0 vulnerabilities

[*] Quasar Project initialization finished! Se tudo deu certo, vai aparecer algo assim! :)
To get started:
  cd meu_app
  quasar dev

```

O próximo passo é rodar esses comandos para começar!

Documentation can be found at: <https://quasar.dev>

Quasar is relying on donations to evolve. We'd be very grateful if you can read our manifest on "Why donations are important": <https://quasar.dev/why-donate>

Donation campaign: <https://donate.quasar.dev>

Any amount is very welcomed.

If invoices are required, please first contact razvan@quasar.dev

Please give us a star on Github if you appreciate our work:
<https://github.com/quasarframework/quasar>

Enjoy! - Quasar Team

Tela de sucesso na criação do projeto

O próximo passo é rodar os comandos informados pelo Quasar, ilustrados na imagem anterior. Você vai rodar os seguintes comandos:

```
ls  
cd meu_app
```

O comando “ls” lista todos os arquivos e pastas que existem no diretório atual. Só rodamos esse comando para ver se a pasta do nosso app foi criada.

No Windows o comando para listar os arquivos do diretório atual é o “dir”.

O comando “cd” serve para entrar em alguma pasta. Nós usamos ele para entrar na pasta do nosso app. Tanto no Linux quanto no Windows, esse comando é o mesmo!

Pronto, nosso projeto está criado! Agora vamos vê-lo em ação:

```
quasar dev
```

Se você estiver no Windows e ao rodar o comando acima der algum erro de permissão, tente subir o servidor em outra porta usando o comando abaixo:

```
quasar dev -p 80
```

Esse comando vai subir um servidor com a nossa aplicação para podermos acessá-la pelo navegador.

Você **tem que ter entrado dentro da pasta** criada com o nome que você deu para o seu projeto para rodar esse comando.

The screenshot shows a terminal window with the following content:

```

x
https://github.com/quasarframework/quasar
Enjoy! - Quasar Team
Entre na pasta do projeto que foi criada com o comando "cd"
→ projects cd meu_app
→ meu_app quasar dev Rode o comando "quasar dev" para iniciar a aplicação

Dev mode..... spa
Pkg quasar..... v1.9.10
Pkg @quasar/app... v1.6.0
Debugging..... enabled

app:quasar-conf Reading quasar.conf.js +0ms
app:dev Checking listening address availability (0.0.0.0:8080)... +3ms
app:webpack Extending SPA Webpack config +478ms
app:mode 'store' feature flag was missing and has been regenerated +8ms
app:generator Generating Webpack entry point +4ms
app:dev-server Booting up... +1ms

SPA [100%] in ~5s

[DONE] Compiled successfully in 5033ms

[N] App dir..... /home/italopaiva/projects/meu_app
App URL..... http://localhost:8080
Dev mode..... spa
Pkg quasar..... v1.9.10
Pkg @quasar/app... v1.6.0

[『wds』: Project is running at http://0.0.0.0:8080/
『wds』: webpack output is served from
『wds』: 404s will fallback to /index.html
app:browser Opening default browser at http://localhost:8080 +5s]

```

A green box highlights the command `→ projects cd meu_app`. Another green box highlights the command `→ meu_app quasar dev`. A red box highlights the text "Rode o comando 'quasar dev' para iniciar a aplicação". A green box highlights the message "[DONE] Compiled successfully in 5033ms". A green box highlights the message "[『wds』: Project is running at http://0.0.0.0:8080/". A green box highlights the message "[app:browser Opening default browser at http://localhost:8080 +5s]".

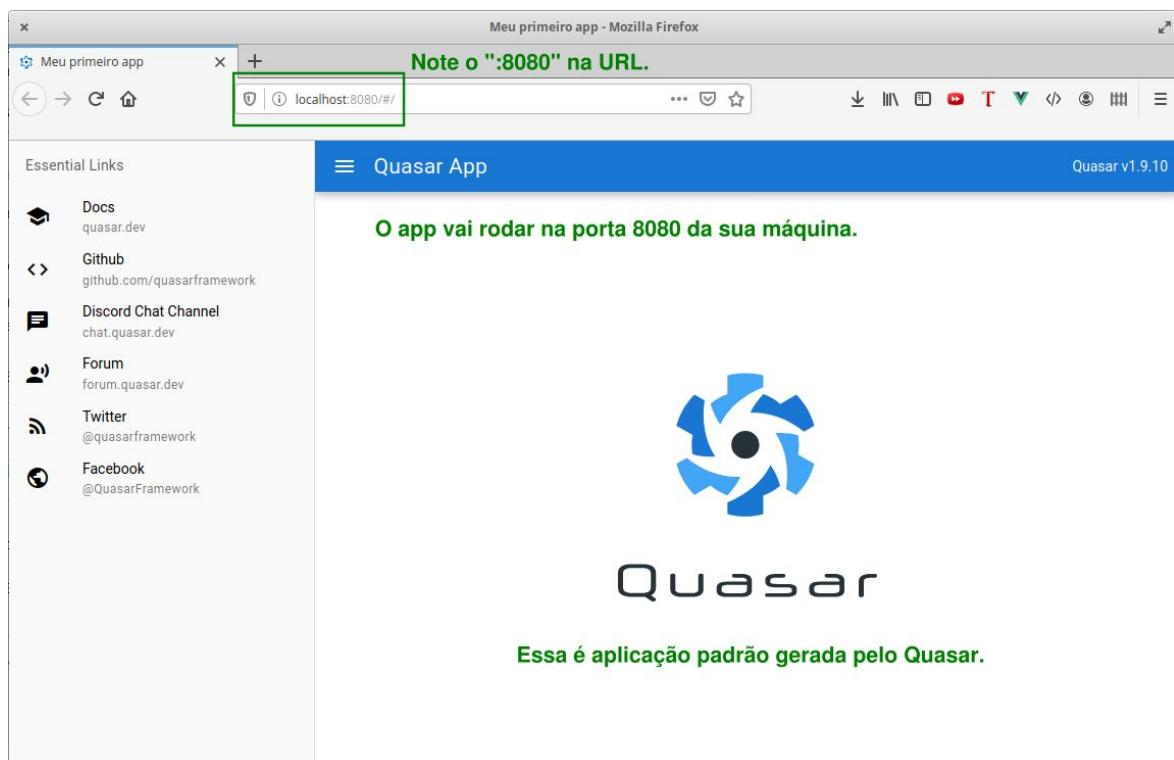
**Se tudo deu certo,
vai aparecer algo assim.**

Rodando a aplicação criada

Se tudo deu certo (é pra ter dado! haha), ao final do comando “**quasar dev**” o próprio Quasar vai abrir o seu navegador com a aplicação padrão criada pelos comandos que nós rodamos até agora.

Como você está rodando a aplicação na sua própria máquina, o endereço da sua máquina vai ser “**localhost**”.

O Quasar sobe a aplicação na porta 8080 (se você não sabe o que é uma porta no computador, não tem problema!), por isso para acessar a aplicação precisamos colocar “:8080” no final da URL, como mostra a imagem abaixo.



OBS 1.: Se você tiver rodado o comando “`quasar dev -p 80`”, para acessar o app você tem que usar a URL `localhost:80` ou somente `localhost`.

OBS 2.: Você precisa manter o servidor rodando lá no terminal enquanto estiver fazendo as práticas do restante do e-book. Toda vez que você mudar o código o Quasar automaticamente vai atualizar o servidor e a página para te mostrar as suas alterações.

Caso você tenha algum problema, é só matar o servidor no terminal (apertando `CTRL + C`) e subir de novo (rodando o “`quasar dev`”).

Pronto! A estrutura para a nossa aplicação foi criada com sucesso!

Parte 4 - As três bases do desenvolvimento web

Atualmente, para construir para web você provavelmente vai precisar de pelo menos **três ingredientes principais**: HTML, CSS e Javascript. Cada um tem o seu papel na criação de uma aplicação.

Se você quiser se aprofundar um pouco mais sobre cada uma dessas linguagens, eu fiz uma **série de vídeos pensada para iniciantes** falando de um jeito simples sobre [HTML](#), [CSS](#) e [JavaScript](#). Isso com certeza vai te ajudar a entender melhor esse e-book!

Mas mesmo assim vou te dar uma breve explicação.

O HTML (*Hypertext Markup Language*) é uma linguagem de marcação que serve para criar a estrutura das páginas, ou seja, o conteúdo da página, o que ela deve apresentar na tela.

O CSS (*Cascading Style Sheets*) é uma linguagem utilizada para definir o estilo das páginas, descrevendo como os componentes HTML devem ser apresentados na tela. Tudo que for relacionado a estilo (cor, tamanho, posicionamento, etc), a gente usa CSS para definir. O CSS é aplicado nos elementos HTML.

O JavaScript é uma linguagem de programação que serve para adicionar interação nas páginas e definir o comportamento das páginas. Por exemplo, se você quer que quando o usuário clicar em um botão alguma coisa aconteça, você usa o JavaScript para programar o que deve acontecer quando o botão for clicado.

Juntando essas três linguagens, temos uma caixa de ferramentas poderosíssima para construir aplicações para a web.

Para entender melhor como essas três linguagens conversam dentro de uma aplicação, considere o código apresentado na imagem a seguir ([o código pode ser baixado nesse link](#)).



```

Applications Ter, Jul 3 17:48
File Edit Selection Find View Goto Tools Project Preferences Help
example.html - /example.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
<html>
<head>
<meta charset="UTF-8">
<title>As 3 bases da Web</title>
</head>
<body>
<h1 class="meu-texto-centralizado">As 3 bases da Web</h1>
<p class="meu-texto-azul meu-texto-centralizado">
    O CSS altera a apresentação de um componente HTML
    através da adição de classes CSS pelo atributo <i>"class"</i>.
</p>
<div class="meu-texto-centralizado">
    <button class="meu-botao-grande-e-green" onclick="alertarQueFoiClicado()">
        Clique aqui
    </button>
</div>
<style>
.meu-texto-azul {
    color: blue;
}
.meu-texto-centralizado {
    text-align: center;
}
.meu-botao-grande-e-green {
    background-color: green;
    font-size: 20px;
}
</style>
<script>
    function alertarQueFoiClicado(){
        alert('O botão foi clicado!');
    }
</script>
</body>
</html>

```

Line 41, Column 8 Spaces: 2 HTML

Exemplo de código com HTML, CSS e JavaScript

```

1 <html>
2   <head>
3     <meta charset="UTF-8">
4     <title>As 3 bases da Web</title>
5   </head>
6   <body>
7     <h1 class="meu-texto-centralizado">As 3 bases da Web</h1>
8
9     <p class="meu-texto-azul meu-texto-centralizado">
10       O CSS altera a apresentação de um componente HTML
11       através da adição de classes CSS pelo atributo <i><code>class</code></i></a>.
12     </p>
13
14     <div class="meu-texto-centralizado">
15       <button class="meu-botao-grande-e-green" onclick="alertarQueFoiClicado()">
16         Clique aqui
17       </button>
18     </div>
19
20 <style>
21   .meu-texto-azul {
22     color: blue;
23   }
24
25   .meu-texto-centralizado {
26     text-align: center;
27   }
28
29   .meu-botao-grande-e-green {
30     background-color: green;
31     font-size: 20px;
32   }
33 </style>
34
35 <script>
36   function alertarQueFoiClicado(){
37     alert('O botão foi clicado!');
38   }
39 </script>
40
41 </body>
42 </html>

```

Line 41, Column 8 Spaces: 2 HTML

HTML

CSS

JavaScript

Se você não domina HTML, CSS e JavaScript, [baixe esse código](#), salve com extensão **.html** e abra no seu navegador. Observe esse código por um momento e tente entender o que está acontecendo.

Quero que você veja como as peças se encaixam.

Com o HTML nós criamos a estrutura e o conteúdo da página (mude alguns componentes de lugar para ver que a estrutura da página muda).

Com o CSS nós modificamos cores, posicionamento e tamanho, ou seja, definimos o layout e estilo da página. Apague a seção de CSS do código e veja o que acontece quando você recarrega a página.

Com o JavaScript nós adicionamos interação na página. No exemplo do código acima quando o botão é clicado um alerta é exibido para o usuário. Esse é um exemplo simples, mas com o JavaScript podemos fazer MUUITO mais que isso!

Mas por que eu estou te falando isso?!

Porque o Quasar utiliza o Vue.js e o Vue.js estrutura os componentes se baseando justamente nesses três pilares da Web: estrutura/conteúdo (HTML), layout/estilo (CSS) e interação/comportamento (JavaScript).

Agora que você tem uma base do desenvolvimento web, vamos ver os componentes gerados pelo Quasar para você ver na prática a estrutura apresentada neste capítulo.

Parte 5 - Explorando o projeto criado

Agora, vamos dar uma olhada no código que gera a tela inicial do Quasar que você viu na seção anterior.

Se você estiver desenvolvendo na própria máquina, para ver e editar o código da aplicação você vai precisar de um editor de texto especializado para códigos. O que eu indico para você que tá começando é o **Sublime Text**, que eu [ensino como instalar e configurar no vídeo de instalação do Node.js no Windows](#).

Esse arquivo é o `src/pages/index.vue`. Cada arquivo `.vue` nós chamamos de componente. Para chegar nesse componente faça como mostra a imagem a seguir.

```

File Edit Selection Find View Goto Tools Project Preferences Help
~/projects/meu_app/src/pages/Index.vue (meu_app) - Sublime Text (UNREGISTERED)

FOLDERS
meu_app
  .quasar
  node_modules
  src
    assets
    boot
    components
    css
    i18n
    layouts
  pages
    Error404.vue
    index.vue
  router
  statics
  store
    App.vue
    index.template.html
  .editorconfig
  .gitignore
  .postcssrc.js
  babel.config.js
  jscconfig.json
  package-lock.json
  package.json
  quasar.conf.js
  README.md

Index.vue
<template>
<q-page class="flex flex-center">

</q-page>
</template>

<script>
export default {
  name: 'PageIndex'
}
</script>

```

HTML

JavaScript

Esse é o código que gera a página inicial do app criado pelo Quasar

Separações das bases da web no Vue.js

Como dito antes, o Quasar utiliza o Vue.js. Por isso a extensão do arquivo é “`.vue`”. Cada arquivo “`.vue`” é um componente.

Cada componente possui o seu próprio HTML, CSS e JavaScript.

Perceba na imagem anterior que o Vue separa uma seção para cada linguagem dos três pilares da web (HTML, CSS e JS) dentro de um componente (nessa imagem nós não temos a seção do CSS, mas ela é identificada pelas tags `<style></style>`).

O HTML

O HTML fica dentro das tags `<template></template>`. No Quasar, geralmente as páginas ficarão dentro das tags `<q-page></q-page>`.

Perceba que na página padrão do Quasar (imagem acima), dentro do template existe uma `<q-page></q-page>` com as classes CSS “**flex**” e “**flex-center**”. Remova essas classes CSS do elemento `<q-page>` para ver o que acontece.

Ficou tudo desalinhado, né?! Pois é, essas são classes de estilo do próprio Quasar (por isso não estão listadas na tag `<style></style>`) para alinhamento da página no centro da tela.

Dentro da tag `<q-page>` temos a imagem da logo do Quasar, que está sendo carregada com a tag ``. A tag `` é nativa do HTML e serve justamente para carregar imagens. Pegue o link de uma imagem qualquer no google e coloque no valor do atributo “src” para você ver que a imagem será carregada.

Coloque um texto depois da imagem e veja que o texto será mostrado justamente onde você colocou: depois da imagem! A imagem abaixo ilustra isso acontecendo. Lembre-se que você tem

que salvar o arquivo para que as mudanças se apliquem na página (e lembre-se de manter o servidor de desenvolvimento rodando no terminal).

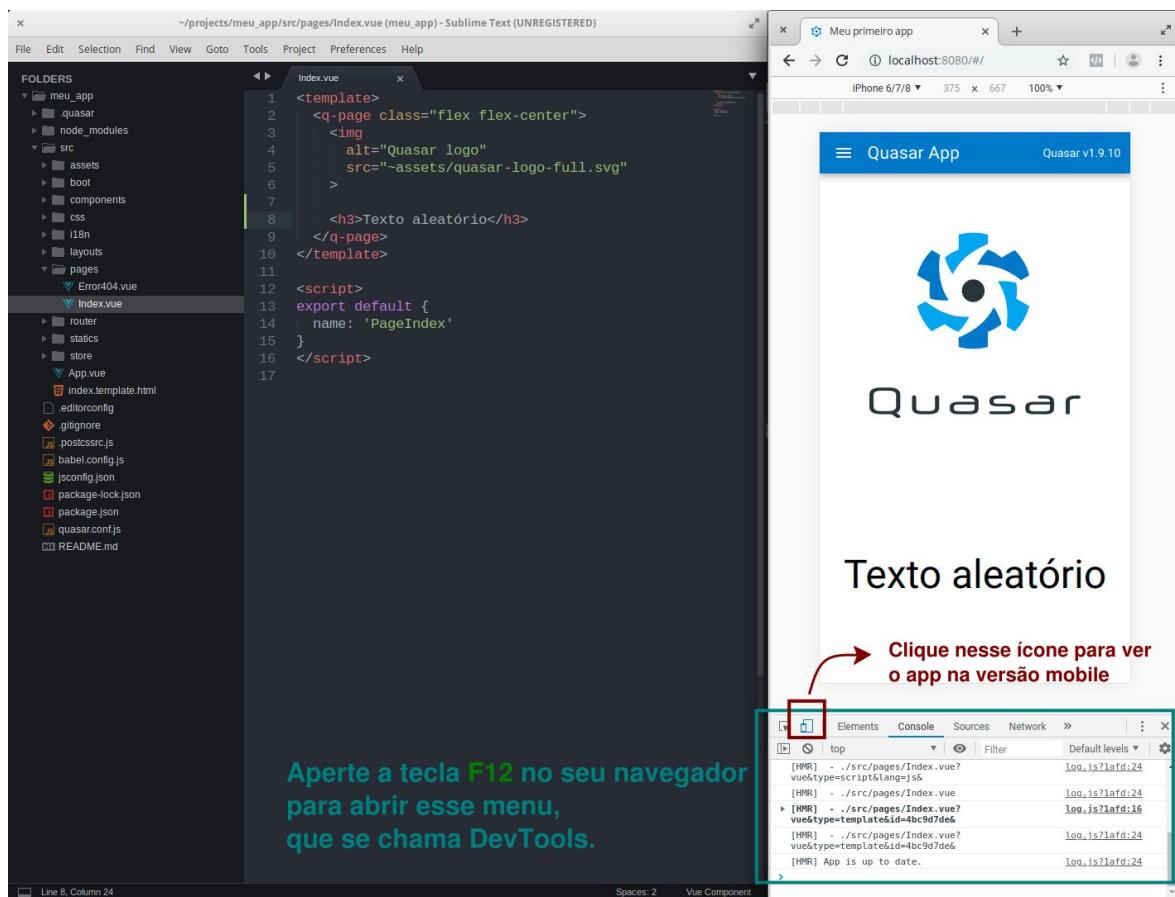
The screenshot shows a development environment for a Quasar application. On the left, a Sublime Text window displays the file `src/pages/Index.vue`. The code includes an `` tag with a source pointing to a Quasar logo SVG file, and an `<h3>Texto aleatório</h3>` tag. On the right, a browser window shows the resulting Quasar application at `localhost:8080/#`. The page features the Quasar logo and the text "Quasar". A green box highlights the text "Texto aleatório" in the browser, which corresponds to the highlighted text in the code editor. Below the browser window, a developer tools console shows log messages related to the component's creation.

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  meu_app
    .quasar
    node_modules
  src
    assets
    boot
    components
    css
    i18n
    layouts
    pages
      Error404.vue
      Index.vue
    router
    statics
    store
    App.vue
      index.template.html
    editorconfig
    .gitignore
    postcssrc.js
    babel.config.js
    jsonconfig.json
    package-lock.json
    package.json
    quasar.conf.js
    README.md
  Line 8, Column 24
  Spaces: 2  Vue Component
  Elements  Console  Sources  Network  >  : 
  [HMR] - ./src/pages/Index.vue?vue&type=script&lang=js&id=100_1s71af:24
  [HMR] - ./src/pages/Index.vue?log.js71af:24
  [HMR] - ./src/pages/Index.vue?log.js71af:16
  [HMR] - ./src/pages/Index.vue?vue&type=template&id=4bc9d7de&log.js71af:24
  [HMR] App is up to date. log.js71af:24
  
```

Colocando texto depois da imagem

Para abrir o site no seu navegador na versão mobile (para ficar igual a imagem anterior) faça como mostra a imagem a seguir.



Abrindo o site na versão mobile no navegador

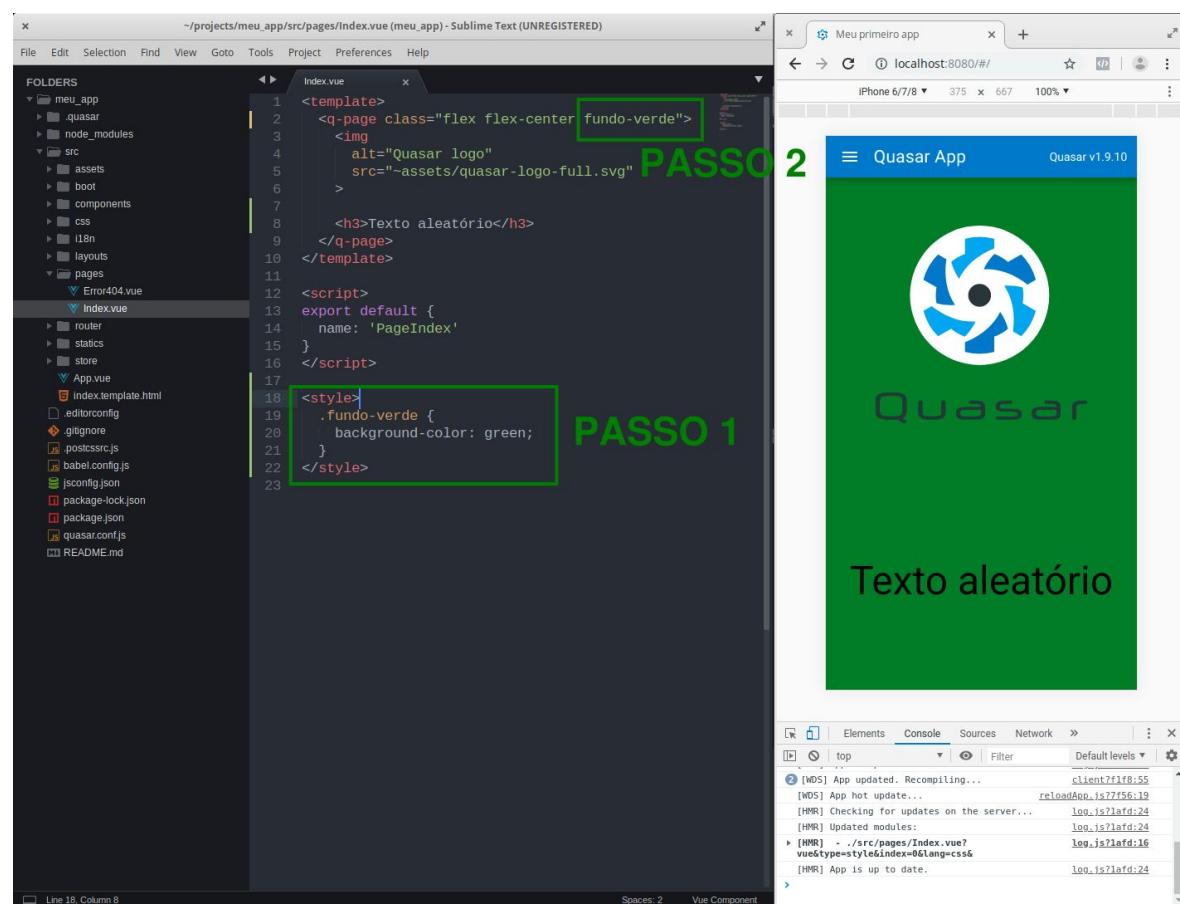
O que você precisa entender do HTML por enquanto é isso: ele define o conteúdo e estrutura da página. Mais conteúdos sobre todas as tags HTML podem ser encontrados [nesse link](#).

Como o Quasar já tem vários componentes prontos, vamos usar basicamente os componentes oferecidos pelo Quasar.

O CSS

O CSS do componente fica dentro das tags `<style></style>`. Dentro dessas tags você pode escrever qualquer código válido de CSS.

Por exemplo, crie uma classe CSS chamada ‘fundo-verde’ e aplique no componente `<q-page>` para ver como fica a aplicação, como ilustra a figura abaixo.



Mais informações sobre as diversas opções do CSS para modificação do layout podem ser encontradas [nesse link](#) e [nessa playlist](#).

O Quasar já tem o estilo dos componentes prontos implementados em CSSs próprios do framework que já são automaticamente carregados. Portanto, não vamos nos prender muito ao CSS neste e-book. Mas você precisa saber que ele existe!

O JavaScript

O JavaScript fica dentro das tags `<script></script>` e segue a [arquitetura proposta pelo Vue.js](#). Dentro dessas tags você pode escrever qualquer código JavaScript válido. Você não precisa se preocupar com toda a estrutura do Vue.js, vou te ensinar o que você precisa saber por agora.

A estrutura básica de um componente Vue é ilustrada na imagem a seguir. Todos os componentes do Quasar utilizam essa estrutura, por isso é importante conhecer pelo menos o básico dela.

Essa estrutura básica é o suficiente para o que vamos ver nesse e-book sobre o Vue.js (se tiver curiosidade [pode pesquisar mais sobre o Vue.js](#)).

```
1 <template>
2 </template>
3
4 <style>
5 </style>
6
7 <script>
8 export default {
9   name: 'NomeDoComponente',
10  data() {
11    return {
12      // Lista de dados que serão utilizados pelo componente
13    };
14  },
15  methods: {
16    // Lista de funções que serão utilizadas no componente
17  }
18}
19</script>
20
```

Estrutura básica de um componente Vue.js

Na chave “name” (linha 9 da imagem), você define um nome para o seu componente.

Na chave “data” (linha 10 da imagem), você define dados que serão utilizados dentro do componente. Vamos utilizar essa chave um pouco mais para frente.

Na chave “methods” (linha 15 da imagem), você define as **funções** que o seu componente vai utilizar. Na orientação a objetos, as funções de um objeto se chama “métodos”, por isso o nome “methods” no Vue. Mas por enquanto, “método = função” para nós.

Para descrever o comportamento da nossa página, ou seja, definir o que ela tem que fazer quando alguém interagir com ela, vamos precisar utilizar **funções**.

Se você não sabe o que é uma função na programação, temos outro e-book que fala bastante sobre isso. Mas vou te dar um spoiler

para podermos continuar: uma função é um tipo de dado que **armazena lógica executável**.

Uma função é uma sequência de passos que tem **um objetivo**. Nós damos um nome para uma função para poder usar depois. Para cumprir esse objetivo, a função **pode precisar de insumos** (entradas). Depois de cumprir o objetivo, a função **pode retornar algum resultado** (saídas). Essa é a estrutura básica de uma função, que é ilustrada na figura abaixo.



Estrutura de uma função.

Como assim?!

Se você quer que o seu software execute alguma ação (como alertar um usuário) você coloca a lógica necessária para isso acontecer dentro de uma função, dá um nome para essa função e usa sua função onde você quiser.

Com funções nós damos vida às nossas aplicações para que elas possam ter o comportamento desejado.

Vamos ver um exemplo!

Na página inicial do Quasar, vamos alertar o usuário toda vez que a logo do Quasar for clicada. Para isso, temos que criar uma função na chave “methods” do nosso componente que dispare um

alerta e chamar essa função quando a imagem for clicada. O código para isso está ilustrado na imagem abaixo.

The screenshot shows a development environment with two main components:

- Sublime Text (Left):** Displays the file `src/pages/Index.vue`. The code defines a template with an image tag and an `@click` event handler. It also defines a `methods` object containing a function `informarQueAImagemFoiClicada()` that alerts "A imagem foi clicada!".
- Browser Preview (Right):** Shows the Quasar App running on `localhost:8080/#/`. The page features the Quasar logo and the text "Quasar App". A blue button labeled "Quasar" is visible at the bottom.
- Console (Bottom):** Shows the browser's developer tools console output, which includes several log messages from the Vue.js framework and the message "App is up to date." at the end.

Adicionando uma função no componente. [Código nesse link.](#)

Primeiro nós criamos a função “`informarQueAImagemFoiClicada()`” dentro da chave “`methods`”. Perceba que demos um nome para a nossa função: “`informarQueAImagemFoiClicada`”.

Para chamar a função quando a imagem for clicada, no Vue.js nós colocamos a diretiva “`@click`” na tag da imagem no HTML do componente. É assim que fazemos o link entre o HTML e o JavaScript! Perceba que chamamos pelo nome a nossa função “`informarQueAImagemFoiClicada`” criada lá embaixo.

Faça o teste, clique na imagem e veja o que acontece!

Uma distinção importante: Criar/declarar uma função e chamar/usar uma função são duas coisas **diferentes**. Você primeiro cria a função para depois usá-la.

```

Applications                               Qua, Jul 4 17:20
untitled* - Sublime Text (UNREGISTERED)
File Edit Selection Find Goto Tools Project Preferences Help
< > untitled
1 <template>
2   <q-page class="flex flex-center">
3     
6   </q-page>
7 </template>
8
9 <style>
10 </style>
11
12 <script>
13   export default {
14     name: 'PageIndex',
15     methods: {
16       informarQueAImagemFoiClicada() {
17         alert('A imagem foi clicada!');
18       }
19     }
20   }
21 </script>
22
23
24

```

Aqui usamos a função que foi declarada logo abaixo

Aqui declaramos a função com o nome "informarQueAImagemFoiClicada"

Distinção entre criar e usar uma função

Quando criamos uma função (no exemplo acima, quando escrevemos ela dentro da chave “methods”) nós apenas estamos **declarando a função**, ou seja, só estamos falando que a aquela função existe.

Quando usamos a função, ela já precisa ter sido **declarada antes**, senão o computador não sabe o que fazer e dá um erro. Quando a função é chamada, o computador procura se tem alguma função com aquele nome declarada e somente se existir uma função com aquele nome, ele executa o código que foi definido dentro da função.

Pronto!

Agora você tem uma base mais sólida para começar a criar os seus apps. Vamos criar uma aplicação bem simples nos próximos capítulos para você ver como é fácil construir apps com o Quasar.

No próximo capítulo vamos explorar um pouco o que o Quasar tem para nos oferecer e aprender como utilizar.

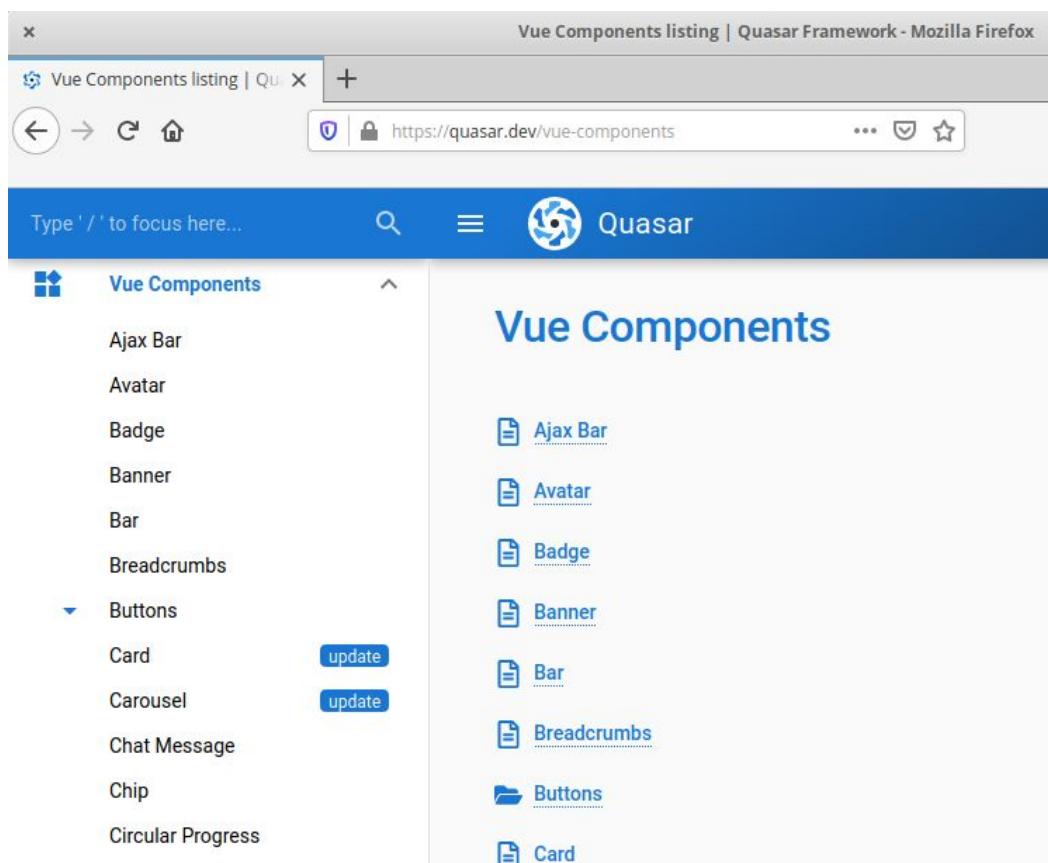
Parte 6 - Como usar o Quasar

Neste capítulo você vai aprender como usar o Quasar para criar um aplicativo.

O Quasar tem uma página onde ele lista todos os componentes que ele possui prontos. Tem tudo que você precisa para criar um aplicativo!

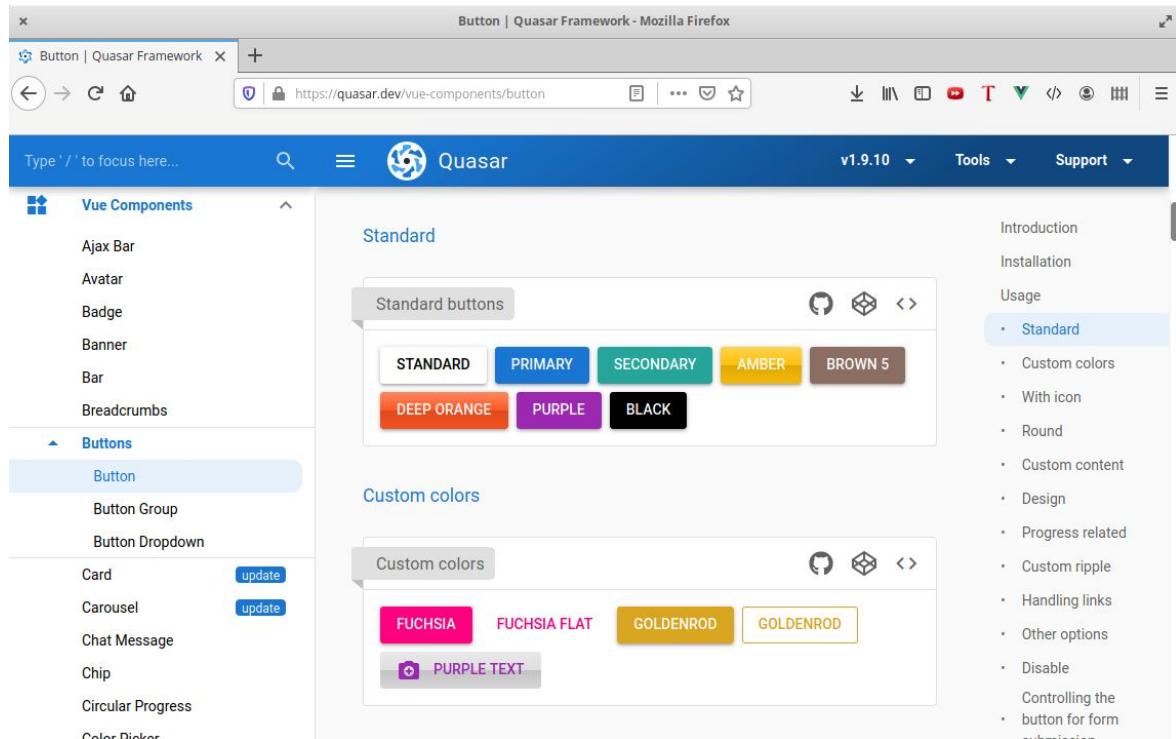
Dica: A documentação está em inglês. Se você não domina o inglês você pode usar um plugin para o seu navegador que traduza a página para você. Ou você pode começar a aprender inglês!

Para conferir, vá na [documentação dos componentes do Quasar](https://quasar.dev/vue-components). O Quasar vai te mostrar todos os componentes que ele tem em categorias.



Explore um pouco essa página e veja o que o Quasar tem para te oferecer.

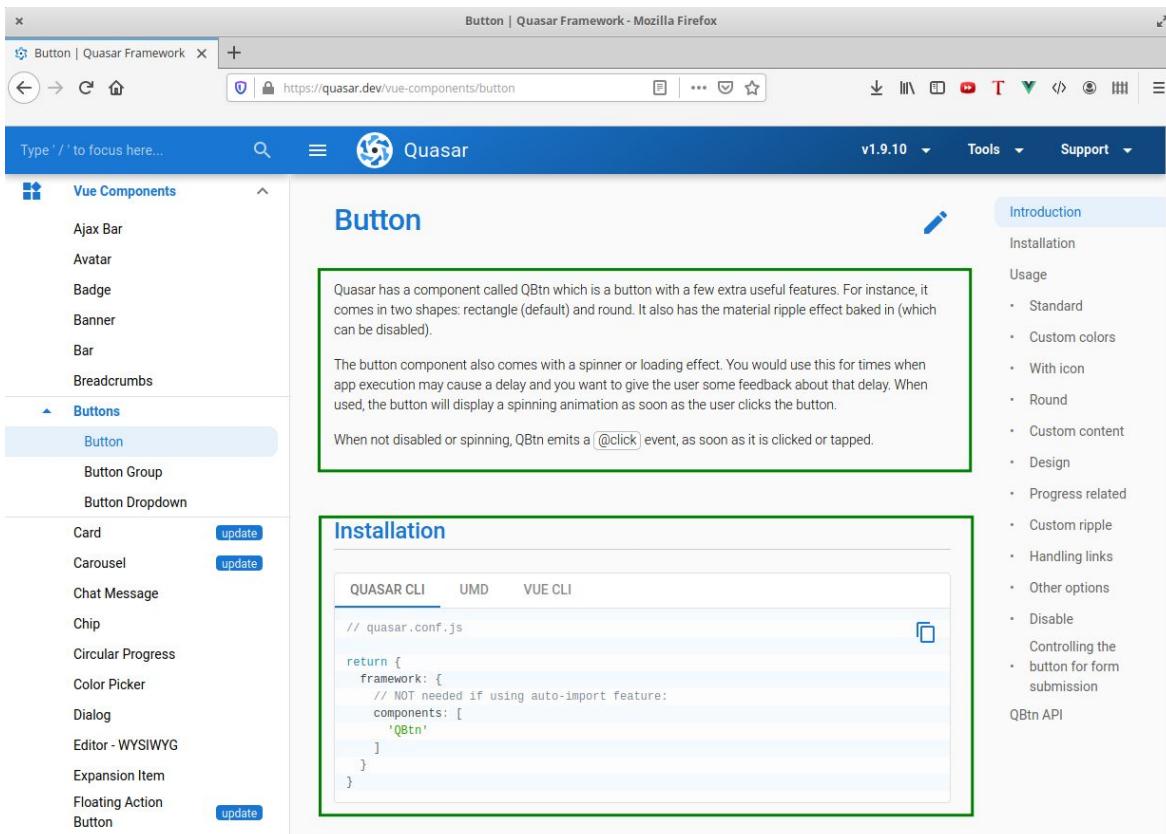
Por exemplo, vamos na [página de botões](#):



Então, se você quer um botão no seu app, você vem aqui pra ver os botões que tem disponíveis. E assim para todos os outros componentes que forem necessários para o seu app.

Para ver como colocar um componente dessa lista do Quasar no seu aplicativo é só seguir a documentação de uso na página de cada componente, que vai ter todas as informações que você precisa para usar o componente.

Por exemplo, na [documentação dos componentes de botão](#) nós temos uma seção descrevendo o componente e uma com instruções de instalação. Logo abaixo temos [alguns exemplos de uso.](#)



A equipe do Quasar fez um trabalho de documentação incrível! Tudo o que você precisa para usar o componente de botão, por exemplo, é explicado nessa tela. E ainda tem várias demonstrações na própria página. Explore e leia um pouco essa página para você entender melhor.

Agora, basicamente é só seguir essa documentação para colocar o botão na sua página. Vamos lá!

O primeiro passo é a instalação do componente.

A seção de instalação dos componentes você pode pular, porque quando nós criamos o projeto no Capítulo 3 nós escolhemos a opção “**Auto-import**”. Isso significa que o Quasar vai carregar automaticamente os componentes pra gente.

Continuando na documentação de uso do botão, depois da instalação do componente, logo em seguida o Quasar te mostra exemplos básicos de uso do componente.

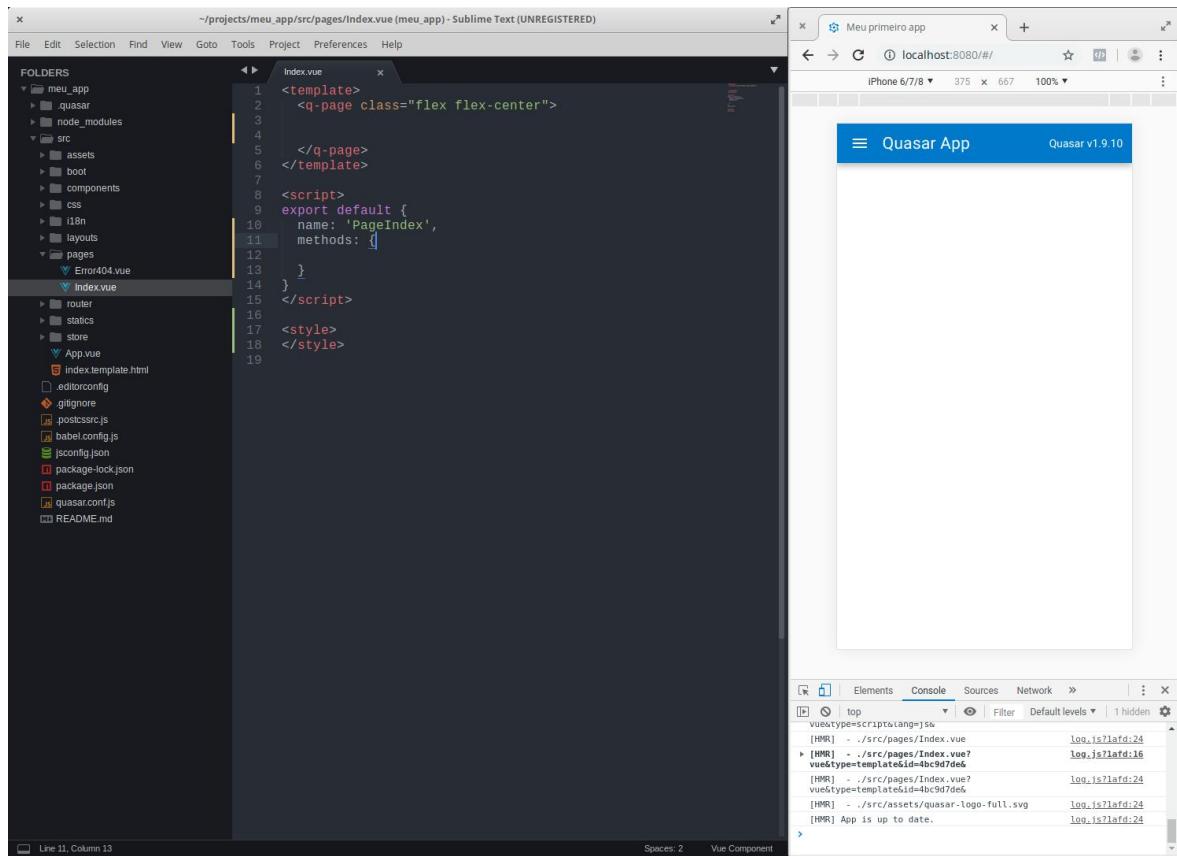
The screenshot shows the Quasar Framework documentation for the Button component. The main content area displays examples of standard buttons with different colors: Primary, Secondary, Amber, Brown 5, Deep Orange, Purple, and Black. Below these examples is a 'View Source' button, which is highlighted with a red box. To the right of the examples, there is a sidebar with navigation links for the Quasar framework, including 'Introduction', 'Installation', 'Usage', and a detailed list under 'Standard' such as 'Custom colors', 'With icon', 'Round', etc. The sidebar also includes a link to the 'QBn API'.

Uso básico do componente de botão

Então, se você quer colocar um botão na sua tela, basta copiar o código do botão apresentado e colar no arquivo da sua página e depois customizar o seu botão para o contexto desejado.

Por exemplo, vamos criar um botão com o código destacado na imagem acima. Só copiar esse código e colocar no nosso arquivo “src/pages/index.vue”.

Mas primeiro, vamos no arquivo “src/pages/index.vue” e apagar a imagem da logo do Quasar e o que fizemos até agora. O arquivo deve ficar como na imagem abaixo.



```

File Edi Selection Find View Goto Tools Project Preferences Help
FOLDERS
~/projects/meu_app
  - .quasar
  - node_modules
  - src
    - assets
    - boot
    - components
    - css
    - i18n
    - layouts
    - pages
      - Error404.vue
      - Index.vue
    - router
    - statics
    - store
    - App.vue
  - index.template.html
  - editorconfig
  - ignore
  - postcssrc.js
  - babel.config.js
  - sconfig.json
  - package-lock.json
  - package.json
  - quasar.conf.js
  - README.md

Index.vue
<template>
<q-page class="flex flex-center">
</q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: {}
}
</script>
<style>
</style>

```

O arquivo deve ficar sem as últimas alterações que nós fizemos

Agora copie e cole o código do botão dentro do componente, salve o arquivo e veja que o botão aparece na tela.

The screenshot shows a development setup with two main windows. On the left is a dark-themed Sublime Text editor displaying the file `src/pages/Index.vue`. The code defines a primary button:

```

<template>
  <q-page class="flex flex-center">
    <q-btn color="primary" label="Primary" />
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}
</script>
<style>
</style>

```

A green box highlights the line `<q-btn color="primary" label="Primary" />`. On the right is a browser window titled "Meu primeiro app" showing the application running at `localhost:8080/#`. The page title is "Quasar App" and the Quasar version is "Quasar v1.9.1". A blue button labeled "PRIMARY" is displayed, also highlighted with a green box. Below the browser window is a caption.

Perceba que o código ao lado cria o botão na tela

Mude o valor da propriedade “label” e veja que o texto do botão também muda.

The screenshot shows a development setup with two main windows. On the left is a Sublime Text editor window titled 'Index.vue' containing Vue.js code. A specific line of code, '`<q-btn color="primary" label="Novo botão" />`', is highlighted with a green rectangular selection. On the right is a web browser window titled 'Meu primeiro app' showing the Quasar App interface. The app has a header bar with the text 'Quasar App' and 'Quasar v1.9.10'. Below the header is a button labeled 'NOVO BOTÃO'. At the bottom of the browser window, there is a developer tools console showing several log messages from the application.

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
~/projects/meu_app/src/pages/Index.vue (meu_app) - Sublime Text (UNREGISTERED)
Index.vue
<template>
  <q-page class="flex flex-center">
    <q-btn color="primary" label="Novo botão" />
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}
</script>
<style>
</style>

```

Meu primeiro app - localhost:8080/# iPhone 6/7/8 375 x 667 100% □

☰ Quasar App Quasar v1.9.10

NOVO BOTÃO

Elements Console Sources Network ▶

[WUR] top not update...
[HMR] Checking for updates on the server...
[HMR] Updated modules:
[HMR] ./src/pages/Index.vue?
vue&type=template&id=4bc9d7de&
[HMR]/src/pages/Index.vue?
vue&type=template&id=4bc9d7de&
[HMR] App is up to date.
Log.js?1afdf:24

Seguindo na documentação de uso do componente, o Quasar te mostra todas as propriedades que você pode usar com aquele componente.

The screenshot shows the Quasar Framework documentation for the `QBtn` component. The left sidebar lists various Vue Components, with `Buttons` expanded and `Button` selected. The main content area displays the `QBtn API` with the following details:

- PROPS**: 31
 - Behavior**: 03
 - Name**: `label` (Type: String | Number)
 - Description: The text that will be shown on the button
 - Example: `Button Label`
 - Content**: 08
 - Name**: `icon` (Type: String)
 - Description: Icon name following Quasar convention; Make sure you have the icon library installed unless you are using 'img:' prefix
 - Examples:
 - `map ion-add img:https://cdn.quasar.dev/logo/svg/quasar-logo.svg`
 - `img:statics/path/to/some_image.png`
 - General**: 02
 - Router**: 02
 - State**: 02
 - Style**: 14
 - Name**: `icon-right` (Type: String)
 - Description: Icon name following Quasar convention; Make sure you have the icon library installed unless you are using 'img:' prefix
 - Examples:
 - `map ion-add img:https://cdn.quasar.dev/logo/svg/quasar-logo.svg`
 - `img:statics/path/to/some_image.png`
- SLOTS**: 2
- METHODS**: 1
- EVENTS**: 1

The sidebar on the right contains links to `Introduction`, `Installation`, `Usage`, and a list of additional components and options.

O Quasar conta até com algumas propriedades para modificar a aparência do componente:

Button | Quasar Framework - Mozilla Firefox

<https://quasar.dev/vue-components/button#QBtn-API>

Type ' / ' to focus here... Search Quasar v1.9.10 Tools Support

Vue Components

- Ajax Bar
- Avatar
- Badge
- Banner
- Bar
- Breadcrumbs
- Buttons**
 - Button**
 - Button Group
 - Button Dropdown
 - Card
 - Carousel
 - Chat Message
 - Chip
 - Circular Progress
 - Color Picker
 - Dialog
 - Editor - WYSIWYG
 - Expansion Item
 - Floating Action Button
- Form Components**
 - Input Textfield

QBtn API

Props 31 **Slots** 2 **Methods** 1 **Events** 1 Filter... Search

QBtn Vue Component

Name	Type
Behavior	String
Content	String
General	Boolean
Router	Object
State	Boolean
Style	14

size String
Description: Size in CSS units, including unit name or standard size name (xs|sm|md|lg|xl)
Examples: 16px, 2rem, xs, md

ripple Boolean | Object
Description: Configure material ripple (disable it by setting it to 'false' or supply a config object)
Default value: true
Examples: { early: true, center: true, color: 'teal', keyCodes: [] }

outline Boolean
Description: Use 'outline' design

flat Boolean
Description: Use 'flat' design

Introduction
Installation
Usage
• Standard
• Custom colors
• With icon
• Round
• Custom content
• Design
• Progress related
• Custom ripple
• Handling links
• Other options
• Disable
Controlling the button for form submission

QBtn API

Button | Quasar Framework - Mozilla Firefox

<https://quasar.dev/vue-components/button#QBtn-API>

Type ' / ' to focus here... Search Quasar v1.9.10 Tools Support

Vue Components

- Ajax Bar
- Avatar
- Badge
- Banner
- Bar
- Breadcrumbs
- Buttons**
 - Button**
 - Button Group
 - Button Dropdown
 - Card
 - Carousel
 - Chat Message
 - Chip
 - Circular Progress
 - Color Picker
 - Dialog
 - Editor - WYSIWYG
 - Expansion Item
 - Floating Action Button
- Form Components**
 - Input Textfield

QBtn API

Props 31 **Slots** 2 **Methods** 1 **Events** 1 Filter... Search

QBtn Vue Component

Name	Type
Behavior	String
Content	String
General	Boolean
Router	Object
State	Boolean
Style	14

fab-min Boolean
Description: Makes button size and shape to fit a Floating Action Button

color String
Description: Color name for component from the Quasar Color Palette
Examples: primary, teal-10

text-color String
Description: Overrides text color (if needed); Color name from the Quasar Color Palette
Examples: primary, teal-10

dense Boolean
Description: Dense mode; occupies less space

Introduction
Installation
Usage
• Standard
• Custom colors
• With icon
• Round
• Custom content
• Design
• Progress related
• Custom ripple
• Handling links
• Other options
• Disable
Controlling the button for form submission

QBtn API

Por exemplo, para deixar o nosso botão grande e roxo basta aplicar as propriedades “size” e “color”. A propriedade “color” precisa de um valor que pode ser qualquer cor da [paleta de cores do Quasar](#), e a propriedade “size” possui uma lista de valores possíveis que é descrita na tabela em destaque da imagem acima. Vamos fazer isso no nosso componente:

The screenshot shows a development environment for a Quasar application. On the left, a Sublime Text window displays the file `src/pages/Index.vue`. The code includes a button component with the following properties: `color="purple" size="xl"`. A green annotation next to the code explains: "'xl' significa 'eXtra Large', ou seja, algo bem grande.' (It means 'eXtra Large', or something very big.)

On the right, a browser window shows the application running at `localhost:8080/#`. The title bar says "Meu primeiro app". The page content shows a purple button labeled "NOVO BOTÃO". A green annotation below the button states: "Perceba que o botão ficou roxo e maior." (Notice that the button turned purple and became larger.)

At the bottom, a developer tools console window shows several log messages from the Hot Module Replacement (HMR) system:

```

[HMR] App not up-to-date...          100_1s71af0:24
[HMR] Checking for updates on the server... 100_1s71af0:24
[HMR] Updated modules:               100_1s71af0:16
> [HMR] - ./src/pages/Index.vue?    vue&type=template&id=4bc9d7de6
[HMR]   ./src/pages/Index.vue?      vue&type=script&id=4bc9d7de6
[HMR] App is up to date.            100_1s71af0:24

```

E voilà! Temos um botão grande e roxo do jeito que queríamos.

Nós fizemos esse processo com um botão, mas serve para qualquer outro componente do Quasar.

Na documentação do componente existem muitos outros exemplos e recursos que o Quasar oferece para você. Não vou

passar por todos eles, mas quis só te dar uma ideia do que você pode fazer e te mostrar que **usar o Quasar se resume** a:

- procurar o componente que você precisa na documentação;
- ver na documentação como utilizar o componente;
- copiar e colar o código do componente;
- adaptar o componente para o que você quer.
- ser feliz :)

Agora que você já sabe como usar o Quasar, vamos usá-lo para criar um aplicativo simples de lista de tarefas no próximo capítulo.

Parte 7 - Criando o seu app

Agora sim, estamos prontos para criar o nosso app!

Vamos criar um aplicativo de lista de tarefas, como o [Google Keep](#). Só que nós vamos fazer o nosso aplicativo um pouco mais simples que o da Google (hahaha).

O nosso vai ser apenas uma lista onde você pode ir adicionando tarefas e marcar essas tarefas como finalizadas clicando em cima delas. Ok?!

O código final pode ser encontrado [nesse link](#), mas vamos ir construindo esse código passo a passo neste capítulo.

Primeiramente, devemos pensar: qual componente poderia representar uma lista de tarefas? E a melhor resposta é: **uma Lista!**

Procure por “List” na documentação dos componentes do Quasar e você vai encontrar “[Lists and List Items](#)”.

The screenshot shows a browser window displaying the Quasar Framework documentation. The URL is <https://quasar.dev/vue-components/list-and-list-items>. The page title is "List and List Items". On the left, there is a sidebar with a tree view of components, where "List & List Items" is selected and highlighted with a green border. Other components listed include Img, Infinite Scroll, Inner Loading, Intersection, Knob, Linear Progress, Markup Table, Menu, No SSR, Observers, Pagination, Parallax, Popup Edit, Popup Proxy, Pull to refresh, Rating, Responsive, and Scroll Area. Some items have "update" or "new" buttons next to them. The main content area contains a section titled "List and List Items" with a brief description of the QList and QItem components. It lists several bullet points about their usage, such as "QItemSection" and "QItemLabel". To the right of the main content, there is a sidebar with links to "Introduction", "Installation", "Usage" (with sub-links for "Basic", "QItemSection", "Active state", "QItemLabel", "More involved examples", and "Connecting to Vue Router"), "QList API", "QItem API", "QItemSection API", and "QItemLabel API". At the bottom of the main content, there are buttons for "QUASAR CLI", "UMD", and "VUE CLI".

Documentação do componente de Listas do Quasar

Você já sabe! Só ver nessa página como instala e um exemplo básico de uso e depois adaptar.

Lembrando que a parte da instalação pode ser pulada porque nós estamos usando a estratégia de “Auto-import” do Quasar.

Então, vamos para a parte de uso básico ver o que o Quasar tem para nos oferecer.

Vamos pegar esse [exemplo de lista mais básica](#).

The screenshot shows a Mozilla Firefox browser window displaying the Quasar Framework documentation at <https://quasar.dev/vue-components/list>. The page title is "List and List Items | Quasar Framework - Mozilla Firefox". The left sidebar lists various components: Img, Infinite Scroll, Inner Loading, Intersection, Knob, Linear Progress, List & List Items (which is highlighted with a green box), Markup Table, Menu, No SSR, Observers, Pagination, Parallax, Popup Edit, Popup Proxy, Pull to refresh, Rating, Responsive, Scroll Area, Separator, Skeleton, Slide Item, Slide Transition, Space, Spinners, Splitter, and Stepper. Below the sidebar, there's a "Basic" example section with a "TEMPLATE" tab containing the following code:

```
<template>
<div class="q-pa-md" style="max-width: 350px">
<q-list bordered separator>
<q-item clickable v-ripple>
<q-item-section>Single line item</q-item-section>
</q-item>

<q-item clickable v-ripple>
<q-item-section>
<q-item-label>Item with caption</q-item-label>
<q-item-label caption>Caption</q-item-label>
</q-item-section>
</q-item>
</q-list>
</div>
</template>
```

Below the template, there are three examples of list items:

- Single line item
- Item with caption
Caption
- OVERLINE**
Item with caption

A green box highlights the "Basic" example section. A green text overlay on the right side of the example box reads: "Vamos usar essa lista mais básica."

E lá está o exemplo de código, vamos copiá-lo e colá-lo na nossa página “src/pages/index.vue” e ver o que aparece.

Você vai ver que vai aparecer uma lista, como mostra a imagem a seguir.

The screenshot displays a development environment for a Quasar application. On the left, a Sublime Text window shows the file `src/pages/Index.vue`. The code contains a template section with a `<q-list>` component. This component is highlighted with a green box. The template includes several `<q-item>` components, some with `v-ripple` and `overline` directives. A script block defines a `methods` object. On the right, a browser window shows the resulting Quasar application. The page title is "Meu primeiro app" and it displays a header "Quasar App". Below the header is a list of items: "Single line item", "Item with caption", "Caption", "OVERLINE", and "Item with caption". The "OVERLINE" item has a horizontal line above it. The browser's developer tools console tab is open at the bottom, showing logs related to the application's build and startup.

```

<template>
  <q-page class="flex flex-center">
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Single line item</q-item-section>
        </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label>Item with caption</q-item-label>
          <q-item-label caption=Caption</q-item-label>
        </q-item-section>
        </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label overline>OVERLINE</q-item-label>
          <q-item-label>Item with caption</q-item-label>
        </q-item-section>
        </q-item>
    </q-list>
  </q-page>
</template>

<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}
</script>

<style>
</style>

```

A lista ficou um pouco pequena e bem no centro da tela. Isso é por causa das classes CSS “flex” e “flex-center” que estão no elemento **<q-page>**. Vamos remover essas classes para que a nossa lista ocupe toda a tela.

Se você só remover essas classes vai ver que a lista vai ficar colada nas bordas da página.

The screenshot shows a development setup with several windows:

- Sublime Text (Index.vue):** The code editor displays the file `index.vue`. A green box highlights the opening tag `<q-page>`. The code includes a template section with a list of items, a script block defining a component named `'PageIndex'`, and a style section.
- Browser (localhost:8080):** The browser window shows the application running. A green box highlights the list of items, which are now aligned to the left edge of the container. Below the list, a green text overlay reads: "Perceba que a lista ficou colada nas bordas da página."
- Console (Vue Component):** The developer tools' console tab shows logs related to the application's state and updates.

Para “descolar” a lista das bordas da página, você pode aplicar a propriedade “**padding**” no elemento **<q-page>**, que vai dar um espaçamento em todas as direções da página (em cima, embaixo, à direita e à esquerda).

OBS.: Eu não tirei essa propriedade da minha cabeça, [ela está descrita na documentação do elemento <q-page>](#).

The screenshot displays a development environment for a Quasar application. On the left, a Sublime Text window shows the file `index.vue` with the following code:

```

<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Single line item</q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label>Item with caption</q-item-label>
          <q-item-label caption>Caption</q-item-label>
        </q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label overline>OVERLINE</q-item-label>
          <q-item-label>Item with caption</q-item-label>
        </q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>

<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}
</script>

<style>
</style>

```

On the right, a browser window shows the Quasar application running on `localhost:8080`. The page title is "Quasar App" and the sub-title is "Quasar v1.9.10". The application displays a list of items with captions and an "OVERLINE" item. A green box highlights the list area. Below the browser window is a developer tools console showing logs:

```

vueType=template&id=4bc9d7de6
[HMR] App is up to date.
[HMR] Checking for updates on the server...
[HMR] Nothing hot updated.
[HMR] App is up to date.

```

No começo da documentação de lista do Quasar, ele mostra como as listas são estruturadas.

Gaste um tempo para entender essa estrutura dos componentes e olhe o código de exemplo e o que foi gerado na aplicação e veja que faz sentido o que está escrito no código com o que é apresentado na tela.

The screenshot shows a browser window displaying the Quasar Framework documentation. The title bar reads "List and List Items | Quasar Framework - Mozilla Firefox". The URL in the address bar is "https://quasar.dev/vue-components/list-and-list-items#Basic". The page content is titled "List and List Items". It contains a green-bordered box with text about QList and QListItem components, their usage, and child components like QItemSection and QItemLabel. Below this is a section titled "Installation" with a code snippet for QUASAR CLI configuration:

```

QUASAR CLI      UMD      VUE CLI
// quasar.conf.js
return {
  framework: {
    // NOT needed if using auto-import feature:
  }
}

```

The left sidebar lists various Quasar components: Img, Infinite Scroll, Inner Loading, Intersection, Knob, Linear Progress, List & List Items (which is selected), Markup Table, Menu, No SSR, Observers, Pagination, Parallax, Popup Edit, Popup Proxy, Pull to refresh, Rating, Responsive, Scroll Area, Separator, Skeleton, and Slide Item. Some items have "update" or "new" status indicators.

Basicamente nós temos:

- o componente `<q-list></q-list>`, que serve para delimitar a nossa lista. todos os outros componentes para montar de fato a lista ficam dentro dele;
- o componente `<q-item></q-item>`, que serve para representar cada item da lista;
- o componente `<q-item-section></q-item-section>`, que serve para criar diferentes seções dentro de um item da lista (veja os exemplos mais elaborados de listas na documentação e você vai entender);
- o componente `<q-item-label></q-item-label>`, que serve para definir textos dentro de uma seção de um item.

```

x ~/projects/meu_app/src/pages/Index.vue * (meu_app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ meu_app
    ├─ .quasar
    ├─ node_modules
    └─ src
        ├─ assets
        ├─ boot
        ├─ components
        ├─ css
        ├─ i18n
        ├─ layouts
        └─ pages
            └─ Error404.vue
            └─ Index.vue
        └─ router
        └─ statics
        └─ store
            └─ App.vue
            └─ index.template.html
    └─ .editorconfig
    └─ .gitignore
    └─ .postcssrc.js
    └─ babel.config.js
    └─ jsconfig.json
    └─ package-lock.json
    └─ package.json
    └─ quasar.conf.js
    └─ README.md

Index.vue
<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Single line item</q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label>Item with caption</q-item-label>
          <q-item-label caption>Caption</q-item-label>
        </q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>
          <q-item-label overline>OVERLINE</q-item-label>
          <q-item-label>Item with caption</q-item-label>
        </q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: [
  ]
}
</script>
<style>
</style>

```

Mas voltando ao código, eu espero que tenha dado para entender que o código que gera a lista faz sentido com o que foi renderizado na tela e que ele possui essa estrutura pré-definida:

The screenshot shows a development environment with two main components:

- Sublime Text (Left):** Displays the file `src/pages/Index.vue`. The code is a Vue template with a `<q-list>` component containing several `<q-item>` elements. Each item includes sections like `<q-item-section>` and `<q-item-label>` with various styling attributes like `v-ripple` and `overline`.
- Browser Preview (Right):** Shows a mobile application interface titled "Quasar App". It displays a list of items under the heading "Single line item". The items include "Item with caption" and "OVERLINE Item with caption".
- Console (Bottom):** Shows the browser's developer tools console tab, displaying logs related to the Quasar framework and Vue.js.

Agora é a hora de adaptar o código copiado do exemplo de uso.

Como o nosso app de lista de tarefas vai ser bem simples, nós só precisamos que cada item da lista tenha somente um texto, que vai ser a descrição da tarefa.

Então podemos remover as linhas 9 a 21 da imagem anterior e usar somente o `<q-item>` do primeiro exemplo nas linhas 5 a 7, como mostra a imagem a seguir.

The screenshot displays two windows side-by-side. On the left is a Sublime Text editor showing the file `src/pages/Index.vue`. A green box highlights the following code block:

```

<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Single line item</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}</script>
<style>
</style>

```

On the right is a browser window titled "Meu primeiro app" showing the result. The title bar says "Quasar App". The page content area contains the text "Single line item". Both the code editor and browser output are enclosed in green boxes.

Para nossa lista ficar um pouco melhor, vamos criar mais algumas tarefas.

Para adicionar uma tarefa na nossa lista é só adicionar mais um **<q-item>** igual o que está nas linhas 5 a 7 e só mudar o texto do **<q-item-section>**.

A imagem abaixo mostra como eu adicionei 4 tarefas na nossa lista.

The screenshot displays three windows related to a Quasar application:

- Code Editor:** Shows the file `src/pages/Index.vue` with the following code snippet highlighted by a green box:

```

<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Estudar HTML</q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>Estudar CSS</q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>Estudar JavaScript</q-item-section>
      </q-item>

      <q-item clickable v-ripple>
        <q-item-section>Acompanhar a @codifiq :)</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  methods: {
  }
}
</script>
<style>
</style>

```
- Browser Preview:** Shows the application running at `localhost:8080/#/`. The title is "Meu primeiro app". The page content is titled "Quasar App" and lists four items: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a Codifiq :)". This content is also highlighted by a green box.
- Developer Tools:** Shows the browser's developer tools with the "Console" tab selected. It displays several log messages from the application's runtime environment, including HMR updates and the status message "App is up to date." The console output is also highlighted by a green box.

Perceba que os elementos `<q-item>` da nossa lista são bem parecidos, a única coisa que realmente muda é o texto no `<q-item-section>` de cada item. Para não repetir o mesmo código vamos utilizar um recurso bastante usado na programação: **laços de repetição**.

A ideia é generalizar um componente e utilizar um recurso da linguagem para repetir esse código para cada elemento de uma lista automaticamente.

Ou seja, vamos criar uma lista de tarefas como dado do componente e vamos varrer essa lista, de forma que para cada elemento que estiver nessa lista a gente crie um `<q-item>` para ele.

Primeiro, vamos criar a nossa lista que vai armazenar as nossas tarefas. Uma lista é denotada pelos colchetes (“[]”) (abre na linha 30 e fecha na linha 35 na imagem abaixo).

Nós criamos esse dado na chave “`data`” do componente. Lembra dela?! Vai ficar assim:

The screenshot shows a Sublime Text editor and a browser window. The Sublime Text window displays the file `index.vue` with the following code:

```

<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple>
        <q-item-section>Estudar HTML</q-item-section>
        </q-item>
      <q-item clickable v-ripple>
        <q-item-section>Estudar CSS</q-item-section>
        </q-item>
      <q-item clickable v-ripple>
        <q-item-section>Estudar JavaScript</q-item-section>
        </q-item>
      <q-item clickable v-ripple>
        <q-item-section>Acompanhar a @codifiq :)</q-item-section>
        </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  },
  methods: {
  }
}
</script>
<style>
</style>

```

A green box highlights the `tarefas` array in the `data()` method. Below the code, there are two annotations:

- Criamos a nossa lista de tarefas.**
- Fazemos isso na chave "data" do componente.**

The browser window shows the application running at `localhost:8080/#`. The title bar says "Meu primeiro app" and "Quasar v1.9.10". The page content is a list titled "Quasar App" with items: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)".

The browser's developer tools Console tab shows the following log entries:

- [HMR] Checking for updates on the server... 100.15.1.101:24
- [HMR] Updated modules: 100.15.1.101:24
- > [HMR] - ./src/pages/Index.vue?vue6Type=scriptLang+jsd 100.15.1.101:16
- [HMR] ./src/pages/Index.vue?vue6Type=scriptLang+jsd 100.15.1.101:24
- [HMR]/src/pages/Index.vue 100.15.1.101:24
- [HMR] App is up to date. 100.15.1.101:24

Essa lista denotada pelos colchetes (“[]”) nós comumente chamamos de [Array](#). Um Array é um tipo de dado que nós usamos para criar um conjunto de elementos ordenados, ou seja, uma lista.

No JavaScript nós podemos criar uma lista (ou um Array) composta por qualquer tipo de elemento. No exemplo acima nós criamos um Array de Objetos (ou [Objects](#)).

Objects, no JavaScript, são um tipo de dado que é composto por pares de [chave:valor](#) e eles servem para representar qualquer coisa a partir de suas propriedades.

No exemplo acima nós abstraímos uma “tarefa” no nosso contexto como algo que possua a propriedade “[descricao](#)”.

Então, para representar as tarefas no nosso código nós estamos usando Objetos do JavaScript, que são denotados pelas chaves (“{}”).

```
{ descricao: 'Estudar JavaScript' }
```

Isso pra gente é uma tarefa, a nível de código. Beleza?!

No exemplo acima, a palavra [descricao](#) seria a [chave](#) do objeto que aponta para o [valor](#) ‘[Estudar JavaScript](#)’. Depois da chave de um objeto sempre tem que ter um ‘dois pontos’ (:) que indica para qual valor essa chave está apontando.

Esse valor ‘[Estudar JavaScript](#)’ apontado pela chave [descricao](#) é um tipo de dado que se chama [String](#), que é denotado pelas aspas simples (‘ ’) ou aspas duplas (“ ”) e serve para representar palavras ou textos.

Eu tentei fazer uma breve explicação sobre os tipos de dados no JavaScript para você não se perder no que nós fizemos aqui. Conhecer os tipos de dados da linguagem é muito importante, por isso vou deixar aqui um link para um [vídeo onde eu explico melhor os tipos do JavaScript](#) e mostro como praticar o uso deles.

Como você já está situado(a) sobre os tipos de dados, vamos voltar para o **laço de repetição**.

Para eliminar essa repetição dos componentes **<q-item>** vamos ler essa lista (ou Array) chamada ‘tarefas’ que nós criamos e criar um **<q-item>** para cada elemento nessa lista e colocar o valor da “descricao” de cada tarefa como o texto dentro do **<q-item-section>**.

No Vue, a sintaxe para repetir um elemento HTML de acordo com os elementos em uma lista é o **“v-for”**.

Basta você colocar o **“v-for”** no componente que você quer que seja repetido e o Vue faz isso para você. No **“v-for”**, você passa qual a lista que é para ser utilizada e um nome que vai representar o elemento corrente. Vai ficar assim:

The screenshot shows a development setup with two main windows. On the left is Sublime Text displaying the file `src/pages/Index.vue`. The code uses a `<q-list bordered separator>` component with a `v-for` loop to iterate over an array named `tarefas`. A green box highlights the `v-for` loop. On the right is a browser window titled "Meu primeiro app" showing the resulting Quasar app. The app has a sidebar with links like "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)" under the heading "Quasar App". A green callout box points to the sidebar with the text: "Perceba que um item foi criado para cada tarefa listada na propriedade 'tarefas' no 'data' do componente.". Below the browser window is a developer tools console showing logs related to a hot update.

```

<template>
  <q-page padding>
    <q-list bordered separator>
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' }
      ],
    };
  },
  methods: {
  }
}
</script>
<style>
</style>

```

Usando laço de repetição para criar elementos dinamicamente

Qual o benefício de usar um laço de repetição?! Se quisermos adicionar mais tarefas, basta adicionar na lista “tarefas” no “data” do componente e não precisa mudar nada no HTML. O HTML está programado para ler a lista e criar um `<q-item>` para cada elemento da lista.

Vai em frente e adicione mais tarefas no Array ‘`tarefas`’ (linha 20) para você ver que elas vão aparecer na tela automaticamente quando você salvar o arquivo.

Só para deixar mais claro o “**v-for**”, é como se você estivesse dizendo: “Para cada **tarefa** na lista de **tarefas**, repita o código do **<q-item>**”.

```
5      <q-item clickable v-ripple v-for="tarefa in tarefas">
6          <q-item-section>{{ tarefa.descricao }}</q-item-section>
7      </q-item>
8
9
```

```
5      <q-item clickable v-ripple v-for="tarefa in tarefas">
6          <q-item-section>{{ tarefa.descricao }}</q-item-section>
7      </q-item>
8
9
```

Perceba também que para renderizar o texto que está dentro da propriedade “**descricao**” de cada “**tarefa**”, nós usamos essa sintaxe **`{{ tarefa.descricao }}`**.

Quando o Vue ver essas duas chaves abrindo e duas chaves fechando **`{{ }}`** ele vai esperar que o que esteja lá dentro seja alguma expressão JavaScript e nós aproveitamos essa funcionalidade para exibir o texto da descrição de cada tarefa, que é acessado através da chave **descricao** do objeto **tarefa**.

Os dados de um objeto JavaScript sempre são acessados através das respectivas chaves. E nós usamos essa sintaxe do ponto final (**.**) em **`{{ tarefa.descricao }}`** para dizer que estamos acessando o valor apontado pela chave **descricao** no objeto **tarefa**.

Essa sintaxe das chaves duplas **`{{ }}`** é necessária para o Vue entender que é para pegar o valor que estiver em **tarefa.descricao** ao invés de escrever “**tarefa.descricao**”

literalmente. Experimente tirar as chaves {{ }} e veja o que acontece.

Fazendo o HTML reagir aos dados do componente (definidos no “**data**”) deixa mais fácil o desenvolvimento porque nós conseguimos controlar o Array de tarefas no JavaScript facilmente.

Agora já temos a listagem das tarefas do nosso app pronta!

Vamos criar agora um campo de texto com um botão para podermos adicionar novas tarefas pela aplicação.

Precisamos de um campo de texto, então vamos procurar na documentação dos componentes do Quasar um campo de texto.

Pesquise por “Text” na documentação do Quasar e você vai encontrar “[Input TextField](#)”, como mostra a imagem abaixo.

The screenshot shows a browser window displaying the Quasar Framework documentation for the 'Input' component. The URL in the address bar is <https://quasar.dev/vue-components/input>. The page title is 'Input | Quasar Framework'. The sidebar on the left lists various components, with 'Input Textfield' highlighted by a green box. The main content area shows the 'Input' component's documentation, including its purpose (to capture user text input), installation instructions (using QUASAR CLI, UMD, or VUE CLI), and design details (coloring, standard, filled, outlined, standout, borderless, rounded design, square borders, dark background). It also includes examples of how to use the component in code and a 'WARNING' section.

Vamos ver a os exemplos básicos de uso desse componente.

Design

WARNING
 For your QInput you can use only one of the main designs (`filled` , `outlined` , `standout` , `borderless`). You cannot use multiple as they are self-exclusive.

Design Overview

TEMPLATE
SCRIPT
□

```
<template>
<div class="q-pa-md">
  <div class="q-gutter-md" style="max-width: 300px">
    <q-input v-model="text" label="Standard" />

    <q-input filled v-model="text" label="Filled" />

    <q-input outlined v-model="text" label="Outlined" />

    <q-input standout v-model="text" label="Standout" />

    <q-input standout="bg-teal text-white" v-model="text" label="Custom standout" />

    <q-input borderless v-model="text" label="Borderless" />

    <q-input rounded filled v-model="text" label="Rounded filled" />

    <q-input rounded outlined v-model="text" label="Rounded outlined" />

    <q-input rounded standout v-model="text" label="Rounded standout" />

    <q-input square filled v-model="text" label="Square filled" />

    <q-input square outlined v-model="text" label="Square outlined" />

    <q-input square standout v-model="text" label="Square standout" />
  </div>
</div>
</template>
```

Standard

Filled

Vamos utilizar esse exemplo que está em destaque na imagem acima. Já sabe né?! Copiar o exemplo e colar na página. Vamos colocar antes da lista de tarefas (`<q-list>`).

The screenshot shows a development setup with two main windows:

- Sublime Text (Left):** A file named `Index.vue` is open. The code includes a `<q-input v-model="text" label="Standard" />` component and a `<q-list bordered separator>` component with several list items. The code is syntax-highlighted in purple, red, and yellow.
- Browser (Right):** A Quasar application titled "Quasar v1.9.10" is running at `localhost:8080/#/`. The page displays a header "Standard" and a sidebar with links: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)". Below the sidebar, there is a list of tasks: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)".

At the bottom of the browser window, a developer tools console shows logs related to a hot update:

```

[WS] App hot update... reloadApp_1s77f56:19
[HMR] Checking for updates on the server... 100_1s77f56:24
[HMR] Nothing hot updated. 100_1s77f56:24
[HMR] App is up to date. 100_1s77f56:24

```

Vamos adicionar um botão para poder adicionar a tarefa. O botão nós já conhecemos, mas se ficar com dúvida basta ir lá na documentação.

Nesse botão eu usei algumas propriedades diferentes para deixar ele mais agradável, como `outline`, `class="full-width"` e `icon="add"`. Vou deixar você olhar essas propriedades [na documentação](#).

The screenshot shows a development environment for a Vue.js application. On the left, the Sublime Text editor displays the file `src/pages/Index.vue`. A specific line of code is highlighted:

```

<template>
  <q-page padding>
    <q-input v-model="text" label="Standard" />
    <q-btn outline
      class="full-width"
      color="primary"
      icon="add"
      label="Adicionar tarefa" />
    <q-list bordered separator>
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  },
  methods: {
  }
}
</script>
<style>
</style>

```

The middle part shows a mobile browser simulation for an iPhone 6/7/8 displaying the application's home screen. The page title is "Meu primeiro app" and it shows a list titled "Standard" with four items: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)". A green box highlights the "Adicionar tarefa" button. The bottom right corner of the browser window shows the Quasar logo.

On the far right, the browser's developer tools are open, specifically the Console tab. It shows several log messages related to hot updates and component lists:

- (Emitted value instead of an instance of Error) <q-item v-for="tarefa in tarefas">: component lists rendered with v-for should have explicit keys. See <https://vuesjs.org/guide/list.html#key-for-each>
- [HMR] App hot update... reloadApp, 1s7756.19
- [HMR] Checking for updates on the server... log, 1s214fd.24
- [HMR] Nothing hot updated. log, 1s214fd.24
- [HMR] App is up to date. log, 1s214fd.24

Agora vamos adaptar o código de exemplo para o nosso contexto. Vamos colocar textos mais descritivos na “label” do `<q-input>`. Vamos trocar também o valor do “**v-model**”.

```

<template>
  <q-page padding>
    <q-input
      v-model="novaTarefa"
      label="O que você precisa fazer hoje?" />
    <q-btm outline
      class="full-width"
      color="primary"
      icon="add"
      label="Adicionar tarefa" />
    <q-list bordered separator>
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      novaTarefa: '',
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  }
}
</script>
<style>
</style>

```

Criamos a propriedade **novaTarefa** como uma String vazia para o campo vir em branco inicialmente

O valor do “**v-model**” é um nome de um dado do componente (que fica no “`data`”) que vai armazenar o valor que for digitado no campo. Precisamos criar no “`data`” do componente uma entrada com esse nome (“`novaTarefa`”) para que possamos salvar o valor digitado.

O que o usuário digitar nesse campo de texto vai ser salvo dentro dessa propriedade “`novaTarefa`” que nós criamos no “`data`”.

Tá vendo que a lista e o botão de adicionar tarefa estão muito colados?! Vamos utilizar o CSS para criar uma classe que dê um espaço entre a lista e o que estiver em cima dela:

```

<template>
  <q-page padding>
    <q-input
      v-model="novaTarefa"
      label="O que você precisa fazer hoje?" />
    <q-btn outline
      class="full-width"
      color="primary"
      icon="add"
      label="Adicionar tarefa" />
    <q-list bordered separator class="espacado-em-cima">
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      novaTarefa: '',
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  },
  methods: {
  }
}
</script>
<style>
.espacado-em-cima {
  margin-top: 30px;
}
</style>

```

Agora ficou melhor!

A tela está pronta, mas se você digitar uma tarefa e clicar no botão, nada acontece. **Por quê?!**

Porque falta uma peça fundamental: descrever o comportamento da página. Lembra?! O que for relacionado à interação da página, nós usamos o JavaScript para escrever isso para a gente.

Como já estamos salvando o valor que for digitado no campo, agora só falta definir o comportamento do botão. Pensa aí comigo: qual deve ser o comportamento do botão?!

Eu pensei no seguinte: quando a pessoa **clicar no botão**, temos que **adicionar a tarefa** informada na nossa **lista de tarefas** no “data”.

Porque o nosso HTML está programado para reagir ao que estiver dentro dessa lista.

Concorda?!

E para isso criamos uma **função** que faça isso para nós e chamamos essa função quando o botão for clicado. Ficaria assim:

The screenshot shows the Quasar App running in a browser window titled "Meu primeiro app" at "localhost:8080/#". The app interface includes a text input field asking "O que você precisa fazer hoje?", a button labeled "+ ADICIONAR TAREFA", and a list of tasks: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)". Below the browser is a Sublime Text editor showing the file "/projects/meu_app/src/pages/Index.vue". The code defines a template with an input field and a button, a list component, and a script block containing a method named "adicionarTarefaNaLista". The method is triggered by the button's @click event. The browser's developer tools console shows logs related to the application's state and network activity.

```

<template>
  <q-page padding>
    <q-input v-model="novaTarefa" label="O que você precisa fazer hoje?" />
    <q-btn outline class="full-width" color="primary" icon="add" label="Adicionar tarefa" @click="adicionarTarefaNaLista()" />
    <q-list bordered separator class="espacado-em-cima">
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
  export default {
    name: 'PageIndex',
    data() {
      return {
        novaTarefa: '',
        tarefas: [
          { descricao: 'Estudar HTML' },
          { descricao: 'Estudar CSS' },
          { descricao: 'Estudar JavaScript' },
          { descricao: 'Acompanhar a @codifiq :)' },
        ],
      };
    },
    methods: {
      adicionarTarefaNaLista() {
        const tarefa = { descricao: this.novaTarefa };
        this.tarefas.push(tarefa);
      }
    }
  }
</script>
<style>
  .espacado-em-cima {
    margin-top: 10px;
  }
</style>

```

Criando função para adicionar a tarefa na lista

Vou explicar a lógica de adicionar uma tarefa na lista.

```

40     methods: {
41       adicionarTarefaNaLista() {
42         const tarefa = { descricao: this.novaTarefa };
43
44         this.tarefas.push(tarefa);
45       }
46     }
47 }
```

Primeiro, na linha 42 da imagem acima, nós criamos uma constante que cria uma tarefa no padrão que foi utilizado na lista de tarefas que está no “data”, ou seja, é um objeto que tem um atributo “descricao”. Temos que manter essa estrutura, senão o código que criamos antes para criar os itens HTML dinamicamente não funcionará.

Na linha 44 nós adicionamos a tarefa criada no padrão estabelecido dentro da lista de tarefas que está no “data”. [A função “push” é nativa do JavaScript](#) e serve para adicionar um elemento novo em uma lista (lembra que “tarefas” era uma lista que estava no “data” do componente?!).

E essa palavrinha nova, o “**this**”, serve para acessar os valores que estão definidos no “data” dentro do JavaScript. O “**this**” faz referência ao próprio componente, e como o “data” é do componente, usamos o “**this**” para acessar tudo que está dentro do componente.

Teste agora a aplicação com o comportamento do botão adicionado!

Não sei você, mas eu percebi que depois que eu adicionei uma nova tarefa o texto não sai do campo. Olha aí como ficou aqui:

The screenshot displays two windows. On the left is Sublime Text showing the file `Index.vue` with the following code:

```

<template>
  <q-page padding>
    <q-input v-model="novaTarefa" label="O que você precisa fazer hoje?" />
    <q-btn outline class="full-width" color="primary" icon="add" label="Adicionar tarefa" @click="adicionarTarefaNaLista()" />
    <q-list bordered separator class="espacado-em-cima">
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      novaTarefa: '',
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  },
  methods: {
    adicionarTarefaNaLista() {
      const tarefa = { descricao: this.novaTarefa };
      this.tarefas.push(tarefa);
    }
  }
}
</script>
<style>
  .espacado-em-cima {
    margin-top: 10px;
  }
</style>

```

On the right is a browser window titled "Meu primeiro app" showing the application running at `localhost:8080/#`. The page title is "Quasar App". The input field contains "Uma nova tarefa". Below it, the list of tasks shows "Uma nova tarefa" repeated twice, indicating a bug where the input value is not being cleared after submission.

Isso aconteceu porque o valor da variável “novaTarefa” que é usado no “**v-model**” tem que ser resetado depois que se adiciona a tarefa na lista. Vamos fazer essa pequena correção na nossa lógica:

The screenshot shows a development setup with two main windows. On the left is Sublime Text displaying the file `~/projects/meu_app/src/pages/Index.vue`. The code is a Quasar component with a form for adding tasks and a list of tasks. A green callout box highlights the line `this.novaTarefa = '';` with the text "Resetamos a propriedade novaTarefa para uma String vazia depois de adicionar na lista". On the right is a browser window titled "Meu primeiro app" showing the application's UI. The UI has a header "Quasar App" and "Quasar v1.9.10". Below it is a search bar with placeholder "O que você precisa fazer hoje?". A button "+ ADICIONAR TAREFA" is visible. A sidebar lists tasks: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", "Acompanhar a @codifiq :)", and "Uma nova tarefa". At the bottom of the browser window is a developer tools console showing log messages.

```

<template>
  <q-page padding>
    <q-input
      v-model="novaTarefa"
      label="O que você precisa fazer hoje?" />
    <q-btn outline
      class="full-width"
      color="primary"
      icon="add"
      label="Adicionar tarefa"
      @click="adicionarTarefaNaLista()" />
    <q-list bordered separator class="espacado-em-cima">
      <q-item clickable v-ripple v-for="tarefa in tarefas">
        <q-item-section>{{ tarefa.descricao }}</q-item-section>
      </q-item>
    </q-list>
  </q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      novaTarefa: '',
      tarefas: [
        { descricao: 'Estudar HTML' },
        { descricao: 'Estudar CSS' },
        { descricao: 'Estudar JavaScript' },
        { descricao: 'Acompanhar a @codifiq :)' },
      ],
    };
  },
  methods: {
    adicionarTarefaNaLista() {
      const tarefa = { descricao: this.novaTarefa };

      this.tarefas.push(tarefa);
      this.novaTarefa = '';
    }
  }
}
</script>

```

Resetamos a propriedade novaTarefa para uma String vazia depois de adicionar na lista

E é isso! Nossa aplicação já adiciona novas tarefas dinamicamente.

Agora vamos marcar as tarefas como finalizadas quando clicarmos em cima delas.

Já sabemos que para fazer isso basta usar o `@click` e passar uma função que será executada quando o item for clicado.

Antes de criar a nossa função, vamos precisar adicionar um atributo nas nossas tarefas para saber se elas estão finalizadas ou não. Podemos fazer isso adicionando o atributo “finalizada” em cada tarefa da nossa lista de tarefas que está no “data”. Esse atributo terá um valor booleano, ou seja, só pode ser **verdadeiro (true)** ou **falso (false)**. Se o valor de “finalizada” for **true**, significa que a tarefa está

finalizada, e se o valor for **false**, significa que a tarefa não está finalizada. A imagem a seguir mostra como fazer isso:

The screenshot shows a Sublime Text editor on the left displaying the file `Index.vue`. The code is a Vue component template and script. A green box highlights the following section of the script:

```

    tarefas: [
      { descricao: 'Estudar HTML', finalizada: false },
      { descricao: 'Estudar CSS', finalizada: false },
      { descricao: 'Estudar JavaScript', finalizada: false },
      { descricao: 'Acompanhar a @codifiq :)', finalizada: false },
    ],
  },
  methods: {
    adicionarTarefaNaLista() {
      const tarefa = { descricao: this.novaTarefa };

      this.tarefas.push(tarefa);

      this.novaTarefa = '';
    }
  }
}
</script>

```

To the right, a browser window shows the application running at `localhost:8080/#`. The page title is "Meu primeiro app". It has a header "Quasar App" and "Quasar v1.9.10". Below is a form with the placeholder "O que você precisa fazer hoje?". A button "+ ADICIONAR TAREFA" is visible. A sidebar lists tasks: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)". At the bottom, the developer tools' Console tab shows logs related to HMR (Hot Module Replacement) updates.

Temos que adicionar essa propriedade nas novas tarefas que forem criadas dinamicamente pela função “adicionaTarefaNaLista()”.

Para isso, basta adicionar a propriedade “finalizada” com o valor **false** na linha 42, depois da propriedade “descricao”, como mostra a imagem a seguir.

The screenshot shows a development setup with two main windows. On the left is a Sublime Text editor displaying the file 'Index.vue'. The code is a Vue component with a template section containing a form for adding a task and a list of tasks. A script block handles the logic for adding tasks to an array named 'tarefas'. A specific line of code, `this.tarefas.push(tarefa);`, is highlighted with a green box. On the right is a web browser window showing a Quasar application titled 'Meu primeiro app'. The app has a header 'Quasar App' and 'Quasar v1.9.10'. Below the header is a search bar with the placeholder 'O que você precisa fazer hoje?'. There is a button labeled '+ ADICIONAR TAREFA'. A list of tasks is displayed, including 'Estudar HTML', 'Estudar CSS', 'Estudar JavaScript', and 'Acompanhar a @codifiq :)'. At the bottom of the browser window, there is a developer tools console showing logs related to the app's update process.

Agora, podemos criar a nossa função que vai marcar a tarefa como finalizada ou não. O funcionamento tem que ser assim:

Quando você clicar na tarefa:

- se a tarefa estiver finalizada: “**desfinaliza**” ela;
- se a tarefa não estiver finalizada: **finaliza** ela.

Basicamente, a lógica é só inverter o valor da propriedade “finalizada” da tarefa: se for **true**, vira **false**; se for **false**, vira **true**. Nossa função ficaria assim então:

```

x ~/projects/meu_app/src/pages/Index.vue (meu_app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Index.vue
<q-input
  v-model="novaTarefa"
  label="O que você precisa fazer hoje?" />
<q-btn outline
  class="full-width"
  color="primary"
  icon="add"
  label="Adicionar tarefa"
  @click="adicionarTarefaNaLista()" />
<q-list bordered separator class="espacado-em-cima">
  <q-item clickable v-ripple v-for="tarefa in tarefas">
    <q-item-section>{{ tarefa.descricao }}</q-item-section>
  </q-item>
</q-list>
</q-page>
</template>
<script>
export default {
  name: 'PageIndex',
  data() {
    return {
      novaTarefa: '',
      tarefas: [
        { descricao: 'Estudar HTML', finalizada: false },
        { descricao: 'Estudar CSS', finalizada: false },
        { descricao: 'Estudar JavaScript', finalizada: false },
        { descricao: 'Acompanhar a @codifiq :)', finalizada: false },
      ],
    };
  },
  methods: {
    adicionarTarefaNaLista() {
      const tarefa = { descricao: this.novaTarefa, finalizada: false };

      this.tarefas.push(tarefa);
      this.novaTarefa = '';
    },
    alternarFinalizacaoDaTarefa(tarefa) {
      tarefa.finalizada = !tarefa.finalizada;
    }
  }
}
</script>

```

27 characters selected

Elements Console Sources Network > 32

[HMR] Checking for updates on the server... log_1571afdf:24
[HMR] Updated modules: log_1571afdf:24
[HMR] - ./src/pages/Index.vue? vue&type=script&lang=js log_1571afdf:16
[HMR] - ./src/pages/Index.vue? vue&type=script&lang=js log_1571afdf:24
[HMR] - ./src/pages/Index.vue? vue&type=script&lang=js log_1571afdf:24
[HMR] App is up to date. log_1571afdf:24

Perceba que na linha 49 da imagem acima nós usamos uma exclamação (“!”). No JavaScript, a exclamação significa uma **negação**. Ou seja, a exclamação nega o valor do que vier depois dela, que é justamente o que a gente quer: que quando o valor da propriedade “finalizada” de uma tarefa for **false**, ela troque para **true**, e vice-versa.

A linha 49 é como se estivéssemos falando para o JavaScript: “Atualize o valor de `tarefa.finalizada` para a negação do que estiver dentro de `tarefa.finalizada`”.

Agora é só colocar o `@click` no `<q-item>` chamando a função que criamos:

~/projects/meu_app/src/pages/Index.vue (meu_app) - Sublime Text (UNREGISTERED)

```

7 <q-btn outline
8   class="full-width"
9   color="primary"
10  icon="add"
11  label="Adicionar tarefa"
12  @click="adicionarTarefaNaLista()" />
13
14  <q-list bordered separator class="espacado-em-cima">
15    <q-item clickable v-ripple
16      v-for="tarefa in tarefas"
17      @click="alternarFinalizacaoDaTarefa(tarefa)">
18        <q-item-section>{{ tarefa.descricao }}</q-item-section>
19        </q-item>
20    </q-list>
21    Esse parâmetro tarefa que é passado
22    entre parênteses para a função
23    vem do v-for lá em cima.
24  </q-page>
25  </template>
26
27  <script>
28  export default {
29    name: 'PageIndex',
30    data() {
31      return {
32        novaTarefa: '',
33        tarefas: [
34          { descricao: 'Estudar HTML', finalizada: false },
35          { descricao: 'Estudar CSS', finalizada: false },
36          { descricao: 'Estudar JavaScript', finalizada: false },
37          { descricao: 'Acompanhar a @codifiq :)', finalizada: false },
38        ],
39      };
40    },
41  };
42  methods: {
43    adicionarTarefaNaLista() {
44      const tarefa = { descricao: this.novaTarefa, finalizada: false };
45
46      this.tarefas.push(tarefa);
47
48      this.novaTarefa = '';
49    },
50    alternarFinalizacaoDaTarefa(tarefa) {
51      tarefa.finalizada = !tarefa.finalizada;
52    }
53  }
54</script>

```

Line 28 Column 1

Meu primeiro app

localhost:8080/#/ iPhone 6/7/8 375 x 667 100%

Quasar App Quasar v1.9.10

O que você precisa fazer hoje?

+ ADICIONAR TAREFA

Estudar HTML
Estudar CSS
Estudar JavaScript
Acompanhar a @codifiq :)

Elements Console Sources Network Default levels ▾ 34 4 hidden

[HMR] ./src/pages/Index.vue log.js?1afdf:24

[HMR] ./src/pages/Index.vue? vuestype=style&index=0&lang=css log.js?1afdf:16

[HMR] ./src/pages/Index.vue? vuestype=template&id=abc9d7de0 log.js?1afdf:16

[HMR] ./src/pages/Index.vue? vuestype=template&id=abc9d7de0 log.js?1afdf:24

[HMR] App is up to date. log.js?1afdf:24

Lembra que o **v-for** repete o código para cada tarefa na lista?!

Por isso que nós passamos a tarefa como parâmetro (entre parênteses) para a função, para que ela possa modificar a tarefa correta da lista. Já pensou se você clica em uma tarefa e ele finaliza outra?! Não ia ser legal...

Clique nas tarefas e você vai ver que nada muda na tela. **Por que?!** Porque nós não implementamos nada visual para nos indicar que a tarefa está finalizada ou não. Vamos fazer isso agora!

Como o que queremos fazer agora é relacionado ao visual/estilo da página, adivinha com que linguagem vamos fazer isso?! Isso mesmo: **com CSS**.

Então, para as tarefas finalizadas, vamos adicionar uma classe CSS que riscá o texto, para indicar visualmente que a tarefa

foi finalizada. Para deixar o texto riscado (**tipo assim**), nós usamos a propriedade CSS “text-decoration” com o valor “line-through”.

A imagem a seguir mostra como ficaria uma classe CSS com essa propriedade:

The screenshot displays two windows. On the left is a Sublime Text editor showing the file `Index.vue`. A green box highlights the following CSS rule:

```
58     .espacado-em-cima {
59       margin-top: 30px;
60     }
61
62     .tarefa-finalizada {
63       text-decoration: line-through;
64     }
65   
```

On the right is a browser window titled "Meu primeiro app" showing a Quasar App interface. The list contains four items: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar a @codifiq :)". The last item is displayed with a horizontal line through it, indicating it is strikethrough.

Criando uma classe CSS para deixar um texto riscado

Assim como criar uma função é diferente de usar/chamar uma função, **criar uma classe CSS e aplicar essa classe CSS** em algum componente **são duas coisas diferentes**.

Precisamos aplicar essa classe no componente que tem o texto da tarefa. Para usar uma classe CSS, já vimos que é só adicionar a propriedade “class” no componente desejado e colocar o nome da classe CSS como valor. No nosso caso, o componente que contém a descrição da tarefa é o componente **<q-item-section>**.

Perceba que se você colocar essa classe diretamente no componente, todas as tarefas ficarão riscadas, como mostra a imagem a seguir.

The screenshot shows the development environment for a Quasar App. On the left, the `Index.vue` file is open in Sublime Text, displaying the Vue.js template and script. Two specific parts of the code are highlighted with green boxes: one around the CSS rule in the `<style>` block, and another around the `.tarefa-finalizada` class definition in the template. On the right, the browser window shows the application running at `localhost:8080/#`, with an iPhone 6/7/8 simulator selected. The app interface includes a header "Quasar App" and "Quasar v1.9.10". Below is a text input field with placeholder "O que você precisa fazer hoje?", a button "+ ADICIONAR TAREFA", and a list of tasks: "Estudar HTML", "Estudar CSS", "Estudar JavaScript", and "Acompanhar @odifiq :)".

Isso acontece porque o uso dessa classe tem que ser **condicional**, de acordo com o valor da propriedade “finalizada” de cada tarefa individualmente.

E o Vue.js tem um recurso muito interessante para fazer isso para a gente, ele tem uma [forma de aplicar classes CSS baseado no valor de uma variável](#).

A sintaxe para fazer isso é a seguinte (essa é a sintaxe de um [objeto JavaScript](#) que nós discutimos anteriormente só que um pouco diferente):

```
{ 'classe-css-condicional-desejada': variavelBooleana }
```

Você coloca a classe CSS que você quer que seja aplicada entre aspas e do lado direito dos dois pontos (“：“) você coloca uma variável que possua um **valor booleano** (que só pode ser true or false), tudo isso dentro de chaves (“{}”).

Quando o valor da variável for **true**, a classe CSS será aplicada no componente, e quando for **false**, a classe CSS será retirada do componente.

Isso faz sentido para a gente porque é exatamente isso que estamos fazendo: quando clicamos na tarefa nós mudamos o valor da propriedade “finalizada” dela, então colocando essa propriedade “finalizada” como condição para aplicação da classe CSS, a classe CSS sempre acompanhará o valor da propriedade.

Ficaria assim então:

```
:class = "{ 'tarefa-finalizada': tarefa.finalizada }"
```

Perceba que isso { ‘tarefa-finalizada’: tarefa.finalizada } é um objeto JavaScript, onde a chave ‘tarefa-finalizada’ aponta para o valor **tarefa.finalizada**.

The screenshot displays a development environment with two main windows. On the left is a Sublime Text editor showing the file `Index.vue`. The code is a Vue component with a template containing a form and a list of tasks, and a script section defining a method `adicionarTarefaNaLista()`. A style section applies a margin-top of 30px to the first item in the list. On the right is a web browser window titled "Meu primeiro app" showing the application's interface. The browser's developer tools console at the bottom shows log messages related to hot updates and server checks.

Perceba também que colocamos dois pontos (“：“) antes da propriedade “class”, isso é preciso porque como vamos atribuir o valor de “class” dinamicamente, precisamos passar um objeto JavaScript como valor.

E para o Vue entender que é para ele interpretar o valor dessa propriedade “class” como uma expressão JavaScript ao invés de uma string literal, ele utiliza os dois pontos (“：“) antes do nome da propriedade. O Vue chama isso de [bind](#), porque estamos vinculando o valor de uma propriedade a um objeto Javascript. Funciona de forma parecida como as chaves duplas `{} {}`.

Ao fazer isso, perceba que **agora temos um indicativo visual de que as tarefas foram finalizadas!**

Já que o nosso app está funcionando, nós podemos apagar as tarefas de teste que criamos e deixar uma lista vazia no “data” do componente, assim como na imagem a seguir.

Dessa forma, só terão as tarefas que forem adicionadas pela aplicação.

The screenshot displays the development environment for a Quasar app. On the left, the Sublime Text editor shows the 'Index.vue' component code. A green box highlights the note in the script section:

```

<script>
  export default {
    name: 'PageIndex',
    data() {
      return {
        novaTarefa: '',
        tarefas: []
      };
    },
    methods: {
      adicionarTarefaNaLista() {
        const tarefa = { descricao: this.novaTarefa, finalizada: false };
        this.tarefas.push(tarefa);
        this.novaTarefa = '';
      },
      alternarFinalizacaoDaTarefa(tarefa) {
        tarefa.finalizada = !tarefa.finalizada;
      }
    }
  }

```

The browser window on the right shows the 'Meu primeiro app' application running at localhost:8080. The page has a header 'Quasar App' and a sub-header 'Quasar v1.9.10'. It contains a text input field with placeholder 'O que você precisa fazer hoje?' and a button labeled '+ ADICIONAR TAREFA'. Below these, there is a list area outlined in red, which is currently empty. The browser's developer tools console at the bottom shows some log messages related to hot updates.

Agora sim! Nossa aplicação está pronta! Faça os testes aí!

O código final pode ser encontrado [nesse link](#).

Parte 8 - Rodando o seu app

Como agora temos o código pronto, podemos gerar o nosso app.

Embora o Quasar ofereça o [suporte para gerar aplicativos para ser disponibilizados nas lojas do Android e do iOS](#), não vou te ensinar a fazer isso agora.

Vou te ensinar a fazer algo que é uma tendência no mundo web e mobile: **vamos gerar uma PWA!** Várias empresas como Alibaba (que é um dos maiores e-commerce do mundo!), Instagram e Twitter já estão utilizando PWAs.

Uma Progressive Web App (PWA) é uma aplicação web (como essa que criamos nesse e-book) que roda nos dispositivos móveis como se fosse um aplicativo nativo, só que alguns benefícios. Uma PWA se propõe a ser rápida, confiável e imersiva.

Funciona assim:

- você acessa um site que possui suporte para ser uma PWA (não funciona com qualquer site);
- você adiciona esse site na tela inicial do seu celular e ele vai ser instalado como se fosse um aplicativo normal. Você não precisa ficar procurando aplicativo na loja do seu celular e gastar sua internet para baixar aplicativos grandes.
- quando você abrir o ícone que você adicionou na tela inicial, a aplicação vai rodar como se fosse um app nativo, com total imersão.

Para criar uma PWA, existem [critérios que precisam ser respeitados](#).

O Quasar já tem o suporte necessário para gerar as PWAs, e como já temos o código da aplicação basta rodar alguns comandos e nossa PWA estará pronta! Vamos ver o que precisa ser feito, de acordo com a [documentação do Quasar](#):

PWA Build Commands

Developing

```
$ quasar dev -m pwa
# ...or the longer form:
$ quasar dev --mode pwa
```

WARNING
Do not run [Lighthouse](#) on your development build because at this stage the code is intentionally not optimized and contains embedded source maps (among many other things).

Building for Production

```
$ quasar build -m pwa
# ...or the longer form:
$ quasar build --mode pwa
```

Comandos para criar nossa PWA

Então, é só rodar o comando e os arquivos para a nossa PWA serão criados. Rode esse comando no terminal dentro da pasta do projeto:

```
quasar build -m pwa
```

```
quasar build -m pwa
quasar build -m pwa 115x31
→ meu_app quasar build -m pwa

Build mode..... pwa
Pkg quasar..... v1.9.10
Pkg @quasar/app... v1.6.0
Debugging..... no
Publishing..... no

app:mode Quasar PWA is missing. Installing it... +0ms
app:mode-pwa Creating PWA source folder... +0ms
app:mode-pwa PWA support was added +27ms
app:quasar-conf Reading quasar.conf.js +730ms
app:workbox [GenerateSW] Will generate a service-worker file. Ignoring your custom written one. +1s
app:webpack Extending PWA Webpack config +16ms
app:artifacts Cleaned build artifact: "/home/italopaiva/projects/meu_app/dist/pwa" +10ms
app:generator Generating Webpack entry point +0ms
app:build Building... +11ms

PWA building [56%] 390/390 modules 0 active
```

O Quasar vai gerar os arquivos necessários da PWA e colocar na pasta dist/

Depois que esse comando terminar de rodar, perceba que uma pasta “dist” foi criada no diretório do app. Essa pasta contém todo o código da nossa PWA.

```
italopaiva@italopaivapc:~/projects/meu_app
italopaiva@italopaivapc:~/projects/meu_app 115x31
→ meu_app ls
babel.config.js jsconfig.json package.json quasar.conf.js src
dist node_modules package-lock.json README.md src-pwa
→ meu_app ls dist
pwa
→ meu_app ls dist/pwa
css index.html manifest.json service-worker.js
fonts js precache-manifest.c204066762300bb84419b7716660c60f.js statics
→ meu_app
```

Perceba que a pasta dist/ foi criada no diretório do app.

Essa pasta contém todo o código pronto da nossa PWA dentro da pasta dist/pwa/.

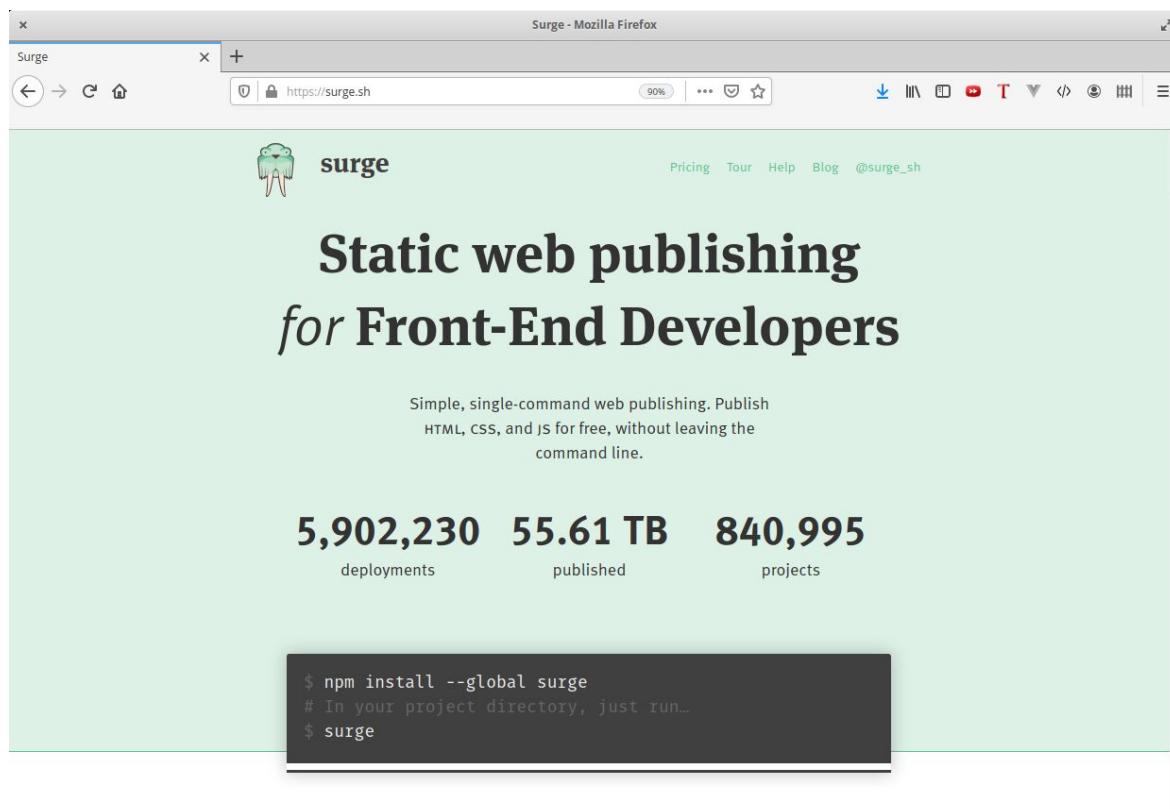
Agora só precisamos servir essa pasta dist/pwa/ via HTTP.

Os comandos que eu rodei na imagem acima foram (troque o **ls** pelo **dir**, se estiver no windows):

```
ls
ls dist/
ls dist/pwa
```

Para deixar nossa PWA rodando e acessível pela internet, nós precisamos subir um servidor HTTP para servir os arquivos da pasta “*dist/pwa*”.

Para isso, nós podemos usar o [Surge](#), que tem uma linha de comando que facilita o upload de arquivos estáticos para um servidor na nuvem. O Surge é uma boa opção para você subir a sua aplicação na internet para testá-la no seu próprio celular.



Para instalar a linha de comando do Surge no seu computador rode o seguinte comando no terminal:

npm install -g surge

```
italopaiva@italopaivapc:~/projects/meu_app
italopaiva@italopaivapc:~/projects/meu_app 110x31
→ meu_app npm install -g surge
npm [WARN] deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
/home/italopaiva/.npm-global/bin/surge -> /home/italopaiva/.npm-global/lib/node_modules/surge/lib/cli.js
+ surge@0.21.3
added 158 packages from 113 contributors in 7.975s

New minor version of npm available! 6.9.0 → 6.14.2
Changelog: https://github.com/npm/cli/releases/tag/v6.14.2
Run npm install -g npm to update!

→ meu_app
```

Depois de instalar a CLI do Surge, rode o comando abaixo para fazer o upload dos arquivos da sua PWA para um servidor do Surge. Tenha certeza de estar dentro da pasta do projeto.

surge dist/pwa

```
italopaiva@italopaivapc:~/projects/meu_app
italopaiva@italopaivapc:~/projects/meu_app 110x31
→ meu_app surge dist/pwa
Running as [Seu email vai aparecer aqui] (Student)

project: dist/pwa
domain: mature-stick.surge.sh
upload: [=====] 100% eta: 0.0s (35 files, 1053924 bytes)
CDN: [=====] 100%
IP: 45.55.110.124
Sua aplicação vai estar no ar a partir dessa URL.
Success! - Published to [mature-stick.surge.sh]
Digite a URL gerada no navegador do seu celular para ver!
→ meu_app
```

Servindo a nossa aplicação para o mundo!

Se for a sua primeira vez com Surge, ele vai te pedir um email e uma senha para realizar o seu cadastro e logo em seguida vai subir a sua aplicação.

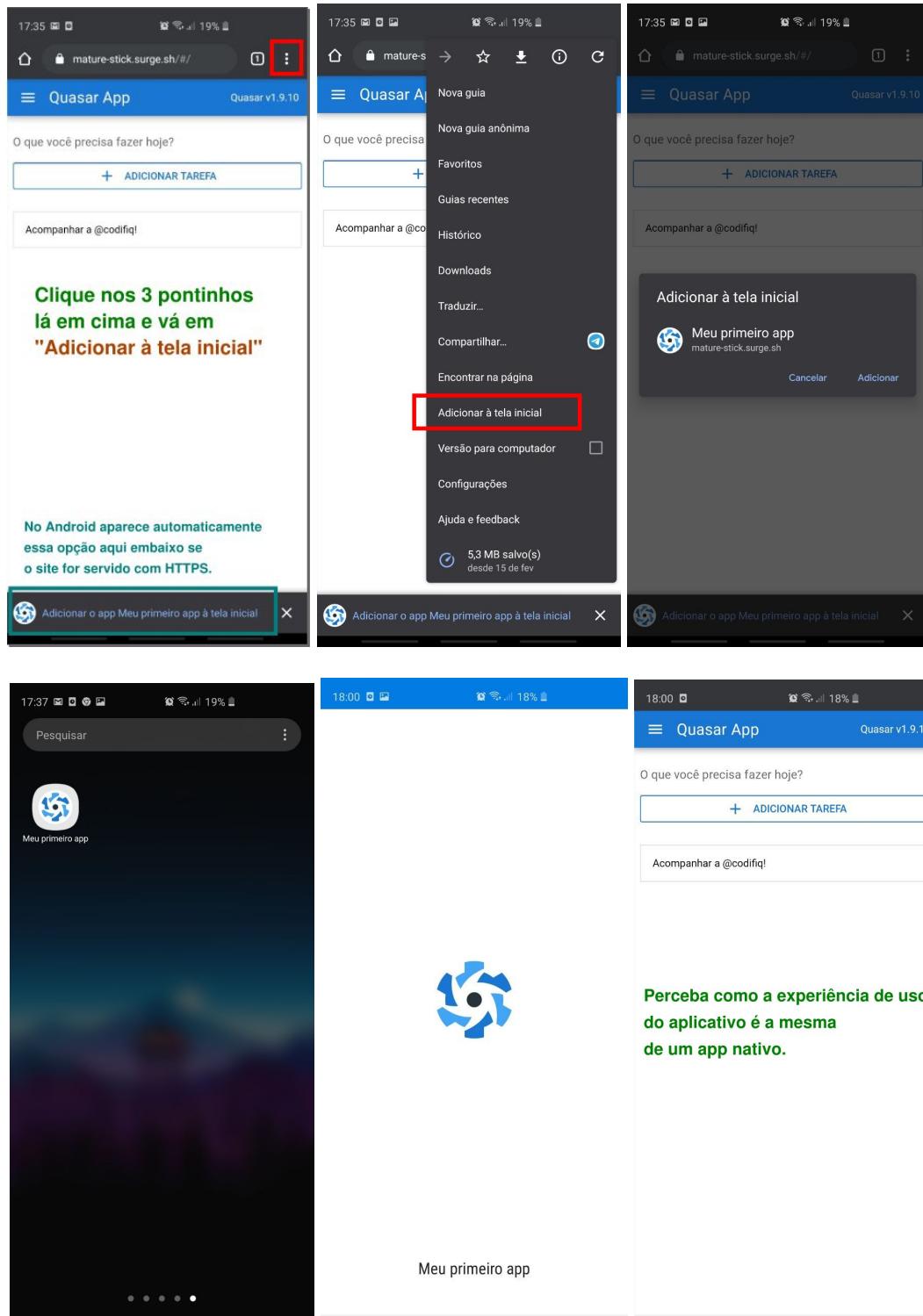
O Surge gera um nome de domínio aleatório para a sua aplicação. No meu caso aqui foi “mature-stick.surge.sh”.

Agora abra o endereço gerado pelo Surge (não o meu, o seu aí que apareceu no seu terminal) no navegador do seu celular e adicione a aplicação à sua tela inicial.

Abra esse endereço com HTTPS no navegador do seu celular. Por exemplo, no meu caso eu abri como “<https://mature-stick.surge.sh>”.

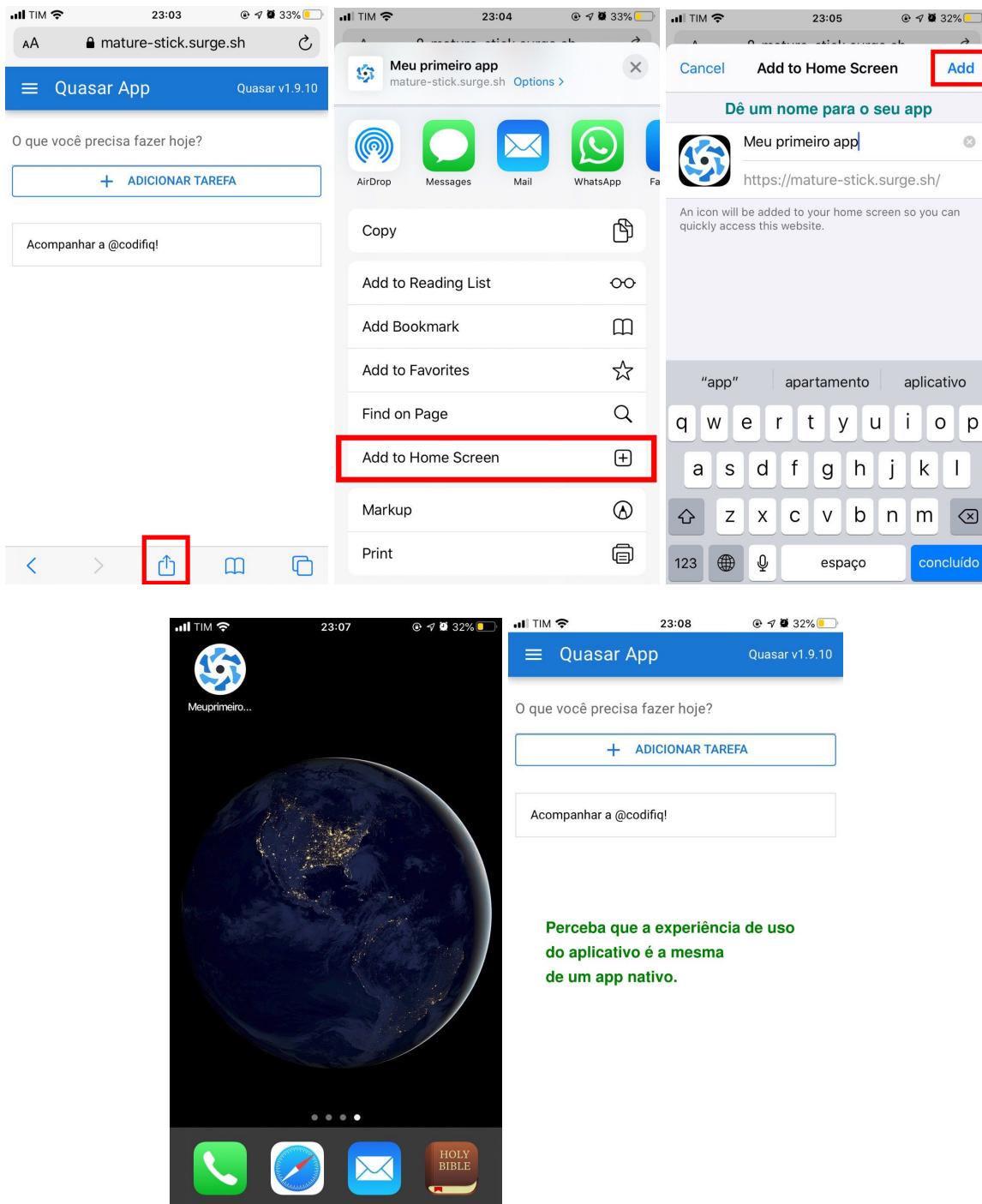
Você vai quer o app que nós construímos aqui vai estar rodando lá como se fosse um app nativo!

No Android seria assim:



Adicionando uma PWA na tela inicial (Android)

No iPhone seria assim:



Adicionando uma PWA na tela inicial (iOS)

Procure seu novo app na tela inicial e abra ele para ver que é a mesma aplicação que criamos e que ela está rodando como se fosse um app nativo!

É isso! Nosso app está pronto!

Talvez este seja o seu primeiro app! **Parabéns!**

Eu coloquei muito trabalho e carinho neste e-book, porque realmente me importo com o aprendizado das pessoas!

Então, eu gostaria de saber como foi para você criar um app seguindo este tutorial. Me manda uma mensagem (pode ser no [Instagram](#) ou no [Facebook](#))!

Conclusão

Nesse e-book pudemos ver que a criação de aplicativos utilizando tecnologias web está muito mais fácil e prática. E o mais importante: vimos que **É POSSÍVEL UMA PESSOA INICIANTE CRIAR UM APP do zero.**

Você também aprendeu o que são PWAs e que são uma tendência no mercado web e mobile, podendo se tornar o futuro das aplicações móveis.

Você talvez tenha percebido que quando você recarrega a página as suas tarefas criadas somem. Isso acontece porque nós não salvamos nenhum dado em um **banco de dados**, ficou tudo na memória do navegador. Para salvar esses dados teríamos que criar uma outra aplicação para registrar isso em um banco de dados para a gente. Mas isso é assunto para outra hora! Se você tem interesse em saber mais sobre isso, acompanhe nossas redes sociais, que estamos sempre lançando conteúdos bacanas para iniciantes!

Por enquanto, explore mais a documentação do Quasar e teste os outros componentes. Tente mudar algumas propriedades e ver o que acontece.

Se você quiser ir mais a fundo, [pesquise também um pouco mais sobre o Vue.js](#), que é um excelente framework que está ganhando seu espaço no mercado e é a base do Quasar.

No próximo capítulo vou deixar alguns desafios para você evoluir mais no que foi aprendido aqui.

Um abraço e até a próxima!

Desafios

“Só se conhece o que se pratica.”

Barão de Montesquieu

Se você chegou até aqui, provavelmente quer crescer ainda mais nessa área e dominar mais o desenvolvimento de software. Para isso **você precisa praticar!**

Então, deixei aqui três desafios para você praticar enquanto continua a aplicação que foi feita aqui:

- **Não adicionar tarefas com descrição vazia;**
 - Do jeito que deixamos, a aplicação permite você adicionar tarefas com a descrição vazia. Isso não deveria acontecer. Então, o seu primeiro desafio é corrigir isso!
- **Implementar a exclusão de uma tarefa da lista;**
 - E se você tiver adicionado uma tarefa errada?! Pois é, nós não implementamos a exclusão de tarefas. Portanto, o seu segundo desafio é criar um botão do lado das tarefas para podermos excluir as tarefas que desejarmos.
- **Separar as tarefas finalizadas das não finalizadas na tela.**
 - Quando você marca uma tarefa como finalizada, ela fica no meio das tarefas não finalizadas. Portanto, o seu terceiro desafio é separar a listagem de tarefas finalizadas das não finalizadas.

Eu adoraria ver as suas soluções! Me manda uma mensagem no [Instagram](#) ou no [Facebook](#) falando que você resolveu os desafios!

Contato

“Transforme-se. Codifiq.”

Você me acha em:



<https://www.instagram.com/codifiq/>



<https://www.facebook.com/codifiq>



<https://codifiq.com.br/>

{codifiq}