



Plataformas de aplica  es Web

Aula 14 - PetTopStore - Loja on-line - Parte 1



Material Did tico do Instituto Metr pole Digital - IMD

Termo de uso

Os materiais did ticos aqui disponibilizados est o licenciados atrav s de Creative Commons **Atribui  o-SemDeriva  es-SemDerivados CC BY-NC-ND**. Voc  possui a permiss o para realizar o download e compartilhar, desde que atribua os cr ditos do autor. N o poder  alter -los e nem utiliza-los para fins comerciais.

Atribui  o-SemDeriva  es-SemDerivados

CC BY-NC-ND



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Apresentação

Nesta aula vamos iniciar o desenvolvimento da nossa última aplicação que faz parte do projeto Pet Top Store: A loja on-line.

A loja terá as funcionalidades de listar produtos, carrinho de compras e finalização do pedido.

Usaremos o framework front-end Svelte para desenvolver a aplicação da loja, que irá consumir a API disponível na aplicação admin, desenvolvida em aulas anteriores.

Iremos nesta aula ter uma visão geral do Svelte e do SvelteKit e iniciaremos o projeto da loja on-line com o recurso de listar produtos vindos da API. Então é necessário que você execute a aplicação Admin da PetTopStore para que a loja funcione pois ele contém a API que será usada.

Projeto admin: [Clique aqui para download](#)

Projeto PDV: [Clique aqui para download](#)

Projeto da loja: [Clique aqui para download](#)

Svelte

Website: <https://svelte.dev/>

O Svelte é um framework front-end reativo, similar em muitos pontos ao React e ao Vue.js, porém com suas próprias peculiaridades que o tornam uma alternativa muito interessante.



SVELTE

CYBERNETICALLY ENHANCED
WEB APPS

Write less code

Build boilerplate-free components using languages you already know – HTML, CSS and JavaScript

[learn more →](#)

No virtual DOM

Svelte compiles your code to tiny, framework-less vanilla JS – your app starts fast and stays fast

[learn more →](#)

Truly reactive

No more complex state management libraries – Svelte brings reactivity to JavaScript itself

[learn more →](#)

Svelte is a radical new approach to building user interfaces. Whereas traditional frameworks like React and Vue do the bulk of their work in the browser, Svelte shifts that work into a *compile step* that happens when you build your app.

Instead of using techniques like virtual DOM diffing, Svelte writes code that surgically updates the DOM when the state of your app changes.

We're proud that Svelte was recently voted the **most loved web framework** with the **most satisfied developers** drawing the **most interest in learning it** in a trio of industry surveys. We think you'll love it too. [Read the introductory blog post](#) to learn more.

```
npm create svelte@latest myapp
cd myapp
npm install
npm run dev
```

[Learn Svelte →](#)

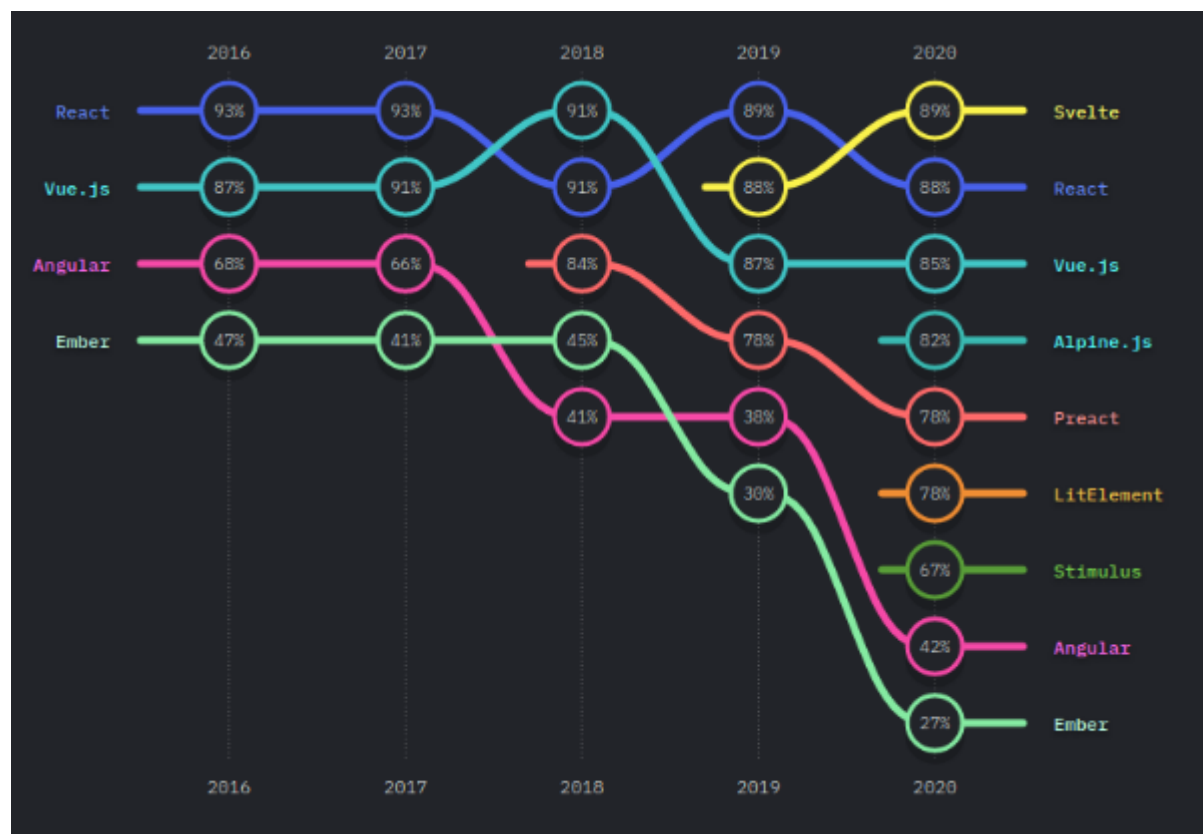
Website do Svelte

O principal diferencial do Svelte é que a maior parte do trabalho de prover reatividade de forma automática não é feito no navegador, como ocorre no React e no Vue.js, mas sim em um *passo de compilação* que acontece quando você constrói uma versão da sua aplicação para implantação final.

No lugar de usar um técnicas como o famoso "virtual DOM", o Svelte converte o seus componentes em Javascript puro que atualiza o DOM real de um documento de forma cirúrgica e imperativa, de acordo com a mudança do estado da sua aplicação.

Essas características podem parecer um pouco complicadas mas na prática elas significam que o Svelte é super eficiente e utiliza poucos recursos para executar no navegador, além de ter menores bundles(o código compactado final que é enviado ao navegador) pois não existe muito código extra rodando no navegador, fora o que

você mesmo escreveu.



Svelte em primeiro lugar na categoria "Satisfação" no site StateOfJs. Fonte: <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>

Mais novo que os frameworks front-end mais populares o Svelte chama atenção pela alta performance e pela facilidade de uso.

O criador do Svelte, Rich Harris, editor gráfico do New York Times, tem uma excelente palestra que demonstra algumas das funcionalidades do framework. Vale a pena conferir nesse link (em Inglês): <https://www.youtube.com/watch?v=AdNJ3fydeao>

Svelte - Reatividade

Vamos ver abaixo um pequeno exemplo de um componente no Svelte.

| App.svelte | Result | JS output | CSS output |
|--|-----------------------------|-----------|------------|
| <pre>1 <script> 2 let nome = 'Pereira Andrade'; 3 </script> 4 5 <h1>Olá {nome}!</h1></pre> | Olá Pereira Andrade! | | |

Simples não é?

Repare que existe no exemplo acima um componente Svelte nada mais é que um arquivo com a extensão ".svelte".

Existe um exemplo uma tag script comum, com uma variável declarada chamada "nome" com o valor "Pereira Andrade". Logo abaixo existe uma tag H1 que usa a variável nome com a notação {nome} que é parte da linguagem de template do Svelte e ela substitui o seu conteúdo pelo conteúdo da variável.

Isso é só o começo. No Svelte é possível se obter reatividade de maneira muito fácil. Veja esse outro exemplo:

```
1 <script>
2   let nome = 'Pereira Andrade';
3   let estresse = 1;
4
5   function aumentarEstresse() {
6     estresse = estresse + 1;
7   }
8 </script>
9
10 <h1>Olá {nome}!</h1>
11 Nível de estresse: {estresse}
12 <button on:click={ aumentarEstresse }>
13   Aumentar estresse
14 </button>
15 |
16 {#if estresse > 5}
17   <span style="color: red">
18     Cuidado! perigosamente estressado
19   </span>
20 {/if}
```

Olá Pereira Andrade!

Nível de estresse: 1 Aumentar estresse

Nesse exemplo, se você pressionar 5 vezes ou mais o botão "Aumentar estresse" aparecerá uma mensagem "Cuidado! perigosamente estressado":

```
1 <script>
2   let nome = 'Pereira Andrade';
3   let estresse = 1;
4
5   function aumentarEstresse() {
6     estresse = estresse + 1;
7   }
8 </script>
9
10 <h1>Olá {nome}!</h1>
11   Nivel de estresse: {estresse}
12 <button on:click={ aumentarEstresse }>
13   Aumentar estresse
14 </button>
15
16 {#if estresse > 5}
17   <span style="color: red">
18     Cuidado! perigosamente estressado
19   </span>
20 {/if}
```

Olá Pereira Andrade!

Nível de estresse: 6

Cuidado! perigosamente estressado

Veja que mesmo com um código muito curto, conseguimos obter um comportamento reativo. Vamos aproveitar para detalhar esse exemplo:

Nas linhas 2 e 3, são declaradas duas variáveis: nome e estresse, com valores iniciais.

Na linha 5, existe uma função chamada `aumentarEstresse()` que sua implementação simplesmente aumenta em 1 o valor da variável "estresse" com o comando "`estresse = estresse + 1;`". Simples assim, não precisa de nenhum artifício especial para se alterar o estado de um componente Svelte já que as variáveis dele são automaticamente o seu estado.

Na linha 11 o valor do estresse é exibido com o `{estresse}` e na linha 12 existe um botão com "`on:click={ aumentarEstresse }`", indicando que ao se clicar nesse botão essa função será chamada. Repare que não é o `onClick` comum do HTML e sim a sintaxe especial do Svelte ("`on:click={ ... }`").

Mais abaixo, na linha 16, existe outro recurso do sistema de template do Svelte, o condicional `{#if ... }` que nesse caso verifica se estresse é maior que 5, se for ele mostra um block de HTML com o alerta em vermelho. o `{#if ... }` é finalizado com o `{/if}`.

Pronto. Quando você clica no botão o valor da variável aumenta e automaticamente o componente decide se vai ou não exibir o alerta baseado no valor da variável estresse, sem você precisar fazer nada mais. Esse comportamento simplificado só é possível pois o compilador do Svelte converte esse código para um código Javascript "por debaixo dos panos" que realiza para você todas as checagens e mudanças necessárias no DOM, como por exemplo adicionar o alerta quando necessário.

Svelte - Blocos de repetição

No Svelte é super fácil criar iteradores de arrays e blocos de código que se repetem em um componente. Para isso basta se usar o `{#each ...}`, como no exemplo abaixo:

```
<script>
  let frutas = ['Maçã', 'Banana', 'Tamarindo'];
</script>

Frutas: <br/>

<ul>
  {#each frutas as fruta}
    <li>{fruta}</li>
  {/each}
</ul>
```

Resultado:

Frutas:

- Maçã
- Banana
- Tamarindo

Veja que o comando `{#each frutas as fruta} ... {/each}` itera no array "frutas" e cada iteração faz com que a variável "fruta" referencia um dos elementos do array.

Vamos alterar exemplo para que seja possível adicionar uma fruta nova na lista:

```
<script>
  let frutas = ['Maçã', 'Banana', 'Tamarindo'];
  let novaFruta = '';
</script>

Frutas: <br/>

<input bind:value={ novaFruta } />
```

```
<button on:click={ () => {  
    frutas = [...frutas, novaFruta];  
    novaFruta = '';  
  }  
  
}  
  
>  
  Adicionar fruta  
  
</button>  
  
<ul>  
  {#each frutas as fruta}  
    <li>{fruta}</li>  
  {/each}  
  
</ul>
```

Resultado:

Frutas:

Adicionar fruta

- Maçã
- Banana
- Tamarindo
- Limão
- Acabate
- Morango
- Pêra
- Abacaxi

Agora toda vez que se digitar uma fruta nova e se clicar em "Adicionar Fruta" ela entra na lista de frutas e o HTML reage, alterando a exibição na listagem.

Repare que usamos um recurso no INPUT com o atributo "bind:value={ novaFruta }", isso faz com que o valor do INPUT fique associado à variável novaFruta, ou seja, se você digitar algo no INPUT o valor da variável muda automaticamente, e se você alterar o valor da variável "novaFruta" o que aparece no INPUT também muda

automaticamente.

No on:click do botão "Adicionar Fruta" nós não chamamos uma função previamente declarada, ao invés foi passado um lambda, ou seja, uma função anônima, que tem o mesmo efeito e seu código vai rodar no clique do botão.

Para adicionar a fruta com o nome na variável "novaFruta" no array "frutas" usamos o comando: "frutas = [...frutas, novaFruta]".

Esse comando significa que o array "frutas" será igual a um novo array com todos os elementos do anterior "...frutas", adicionado de novaFruta no final.

Vale observar que se você decidisse usar algo como "frutas.push(novaFruta)" o código não estaria errado, mas o Svelte não iria disparar a reatividade no HTML pois no Svelte a reatividade depende de atribuições com o uso do operador igual "=". Por isso foi necessário adicionar um elemento no array da forma que fizemos. Você até poderia usar o "push", mas em seguida deveria fazer algo como "frutas = frutas" para obter reatividade, o que é um pouco estranho e desnecessário. Essa é uma restrição do Svelte, mas tecnicamente é uma das coisas que faz ele ser tão rápido pois ele só reage quando algo muda explicitamente com o uso do operador igual "=".

Svelte - Componentes

Como já dito, um componente Svelte é um arquivo com a extensão ".svelte" como por exemplo: Usuario.svelte, Fatura.svelte, Cabecalho.svelte, etc. Você pode importar e usar componentes dentro de outros componentes, assim como passar atributos para os componentes, como se eles fossem novas TAGs HTML disponíveis no seu documento.

Vamos ver um exemplo em Svelte que tem um componente principal (App.svelte) que utiliza o componente Playlist.svelte, e este por sua vez utiliza o Musica.svelte.

Detalhando cada componente, do mais abaixo para o mais acima:

Musica.svelte

```
<script>
  export let nome;

</script>

<style>
  div {
```

```
        background-color: palegreen;
        margin-bottom: 10px;
        border: 2px dashed gray;
        padding: 5px;
    }

</style>

<div>

  □ Tocando: {nome}...

</div>
```

Começando com o `Musica.svelte`. Ele tem uma variável chamada "nome". Repare que ela foi declarada com "export let nome;". O uso do export, faz com que a variável não só seja parte do estado do componente música, mas também ela automaticamente vira um atributo do componente, ou seja, quando você for usar o componente música, deve passar qual o seu nome, assim: .

A declaração de variáveis com o "export" sempre a converte em um atributo.

Esse componente é bem simples, somente exibindo o nome da música (atributo "nome") que será passado para ele quando ele for utilizado em um documento.

Repare também que o existe uma estilização do elemento "DIV" no `Musica.svelte`. É interessante observar que no Svelte estilos que são aplicados a elementos dentro de um componente não passam para outros componentes, então somente as DIVs dentro de Música recebem esse estilo.

Playlist.svelte

```
<script>
  import Musica from './Musica.svelte';

  export let nome;
  export let musicas;

</script>

<style>
  div {
    border: 3px solid green;
    margin: 10px;
```

```
        padding: 2px;
    }
    h1 {
        color: green;
        font-size: 20px;
    }
</style>

<div>
    <h1>
        Playlist: {nome}
    </h1>
    {#each musicas as nomeDaMusica}
        <Musica nome={nomeDaMusica} />
    {/each}

</div>
```

O Playlist.svelte começa importando o componente Musica assim: `import Musica from './Musica.svelte'`;

A Playlist por sua vez tem dois atributos que devem ser passados para ela quando ela for criada em um documento: o nome e a lista de músicas. Por enquanto esses atributos não tem valor definido, mas quando o App.svelte vou criar playlists, ele deve passar o nome e lista de músicas de cada uma que criar.

Esse componente somente exibe o nome da playlyst em um H1 e depois faz uma iteração na lista de músicas que será passado pra ela com o `{#each}` e para cada uma, cria um componente .

App.svelte

```
<script>
    import Playlist from './Playlist.svelte'

    let lancamentos = [
        'Cai cai balão',
        'Billie jean'
    ];

    let musicasQuestionaveis = [
        'Let it go',
        'Baby Shark'
```

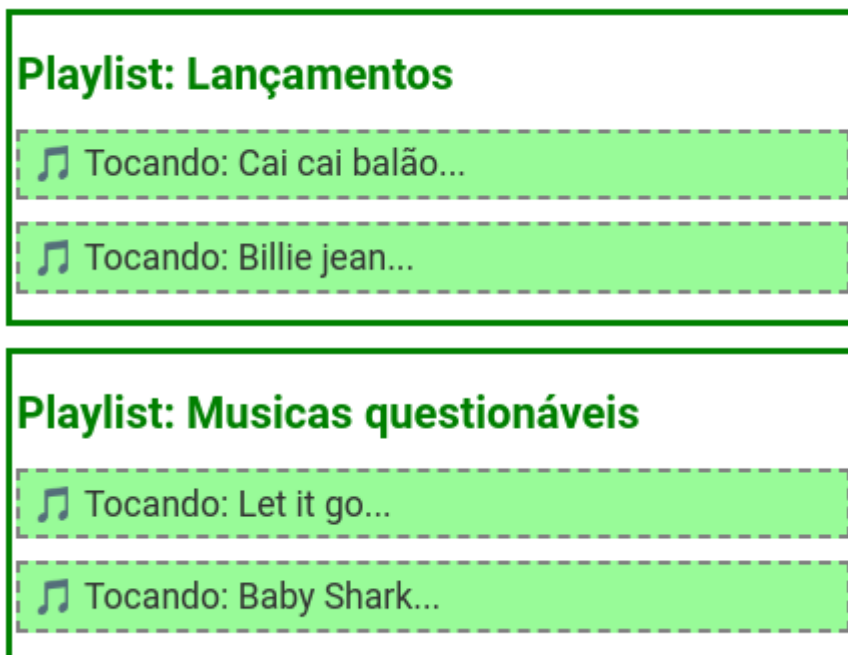
```
];  
  
</script>  
  
<Playlist nome="Lançamentos" musicas={lancamentos} />  
  
<Playlist nome="Musicas questionáveis"  
musicas={musicasQuestionaveis} />
```

Esse é o componente raiz da aplicação exemplo que estamos criando.

Veja que ele inicia importando o componente Playlist e logo depois cria duas listas de músicas, um chamado "lancamentos" e outro chamado "musicasQuestionaveis", cada array com uma lista de nomes.

Ao final são criadas duas Playlists, passando-se os atributos "nome" e "musicas", uma para cada array dos citados acima.

Resultado:



Só mostramos o básico do Svelte e existe muito mais para se aprender.

Para saber mais sobre o Svelte acesse <https://svelte.dev/>

É recomendável que você faça o tutorial oficial on-line disponível no site do Svelte. É realmente muito bem guiado e vai te dá uma visão muito mais profunda das funcionalidades disponíveis.

SvelteKit


Website: <https://kit.svelte.dev/>

Até agora vimos uma visão geral do Svelte, que se trata de uma ferramenta poderosa para se criar interfaces com componentes reativos.

Isso é interessante e suficiente quando vocês está criando um SPA (Single Page Application - Aplicação de página única) onde não existe o conceito de múltiplas páginas e sim de uma única página principal com um componente que engloba todos os outros.

Entretanto nem toda aplicação é assim e nossa loja on-line não é diferente. Na loja precisamos de múltiplas páginas, parâmetros dinâmicos, rotas personalizadas, etc.

Para essas funcionalidades o Svelte tem um framework chamado SvelteKit que adiciona ao Svelte puro todos esses recursos e muitos outros.

 KIT.SVELTE.DEV

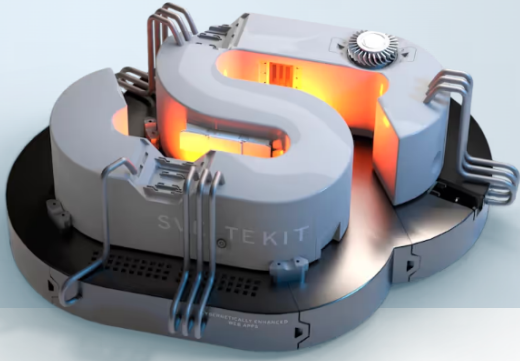
SEARCH

CTRL K

Docs FAQ Svelte

SVELTE KIT

web development, streamlined

[read the docs](#)

fast

Powered by [Svelte](#) and [Vite](#), speed is baked into every crevice: fast setup, fast dev, fast builds, fast page loads, fast navigation. Did we mention it's fast?

fun

No more wasted days figuring out bundler configuration, routing, SSR, CSP, TypeScript, deployment settings and all the other boring stuff. Code with joy.

flexible

SPA? MPA? SSR? SSG? Check. SvelteKit gives you the tools to succeed whatever it is you're building. And it runs wherever JavaScript does.

```
terminal
$ npm create svelte@latest my-app
$ cd my-app
$ npm install
$ npm run dev -- --open
```

← see for yourself

...or [create an app](#) on StackBlitz.

Website do SvelteKit

O SvelteKit é similar aos projetos Next.js (do ecossistema React) e Nuxt.js (do ecossistema Vue.js) e ele é a ferramenta recomendada oficialmente para se iniciar um projeto Svelte.

SvelteKit - Funcionalidades extras

Como já dito anteriormente, o Svelte é somente uma biblioteca para criação de componentes reativos. Vimos algumas de suas funcionalidades e exemplos.

O SvelteKit por outro lado é uma plataforma mais completa, que utiliza o Svelte como seu mecanismo de criar páginas e componentes, mas também algumas funcionalidades que permitem a criação de aplicações completas, inclusive com recursos de processamento do lado do servidor.

Algumas das funcionalidades são:

Páginas e roteamento (routing)

No SvelteKit existe o conceito de páginas. Cada página é simplesmente um componente Svelte, mas que é mapeado para uma rota específica, baseado no seu nome e na pasta que se encontra no projeto.

O Sveltekit utiliza uma convenção chamada filesystem-based routing, ou seja, rotas baseadas no sistema de arquivos. Em resumo isso significa que as rotas (o que seu aplicativo deve fazer quando um usuário navega em uma URL específica) são definidas por pastas no seu projeto.

Cada arquivo chamado "+page.svelte" dentro da pasta "src/routes" cria uma página no seu sistema. Por exemplo, se o usuário navegar para a raiz do site "/" ele irá exibir o conteúdo gerado pelo arquivo "src/routes/+page.svelte". Da mesma forma, se ele tentar acessar "/produtos" o SvelteKit tentará exibir o conteúdo do arquivo "src/routes/produtos/+page.svelte". Dessa forma, cada pasta é uma rota, e o conteúdo que será exibido será o que estiver no arquivo "+page.svelte" dentro dessa pasta.

Exemplos de rotas em uma aplicação SvelteKit qualquer:

| rota | arquivo |
|-------------|--------------------------------------|
| / | src/routes/+page.svelte |
| /quem-somos | src/routes/quem-somos/+page.svelte |
| /cliente/35 | src/routes/cliente/[id]/+page.svelte |
| /cliente/72 | src/routes/cliente/[id]/+page.svelte |

Explicação

- A rota "/" (raiz do site) é direcionada para exibir o componente localizado no arquivo "src/routes/+page.svelte"
- A rota "/quem-somos" é direcionada para exibir o componente localizado no arquivo "src/routes/quem-somos/+page.svelte".
- A rota dinâmica "/cliente/35" ou "/cliente/72" vai exibir o componente "src/routes/cliente/[id]/+page.svelte". Repare que neste caso a pasta "src/routes/cliente" tem um subpasta chamada "[id]" (sim, com colchetes no nome mesmo) e dentro dela um arquivo chamado "+page.svelte". Isso significa que o essa rota tem no nome um atributo dinâmico chamado "id" que é substituído pelo valor que o usuário passar na rota. Nesse exemplo "/cliente/35" vai passar 35 como o "id" e "/cliente/72" vai passar 72 com o "id". O valor do ID pode ser obtido dentro do arquivo "+page.svelte" como veremos mais à frente, assim ele consegue saber que usuário deve obter e exibir.

Mais informações sobre roteamento e páginas em:

<https://kit.svelte.dev/docs/routing>

Endpoints

No SvelteKit existe o conceito de endpoints, que são rotas especiais que não mostram componentes Svelte, mas sim processam uma função javascript no lado do servidor e retornam um conteúdo desejado (um JSON por exemplo). Servem para se criar APIs dentro do próprio SvelteKit para uso pelos componentes da aplicação ou externo.

As rotas dos endpoints seguem um padrão similar ao das rotas das páginas, sendo a única diferença que o arquivo responsável por eles se chama "+server.js" (ou a "+server.ts" se está usando Typescript) e nele terá a implementação de uma função responsável por retornar o conteúdo que desejar.

Não vamos explorar endpoints no nosso projeto da loja pois usaremos uma API externa que já foi criada no admin.

Para saber mais sobre endpoints acesse a documentação do SvelteKit:

<https://kit.svelte.dev/docs/routing#server>

Layouts

Layouts são componentes especiais que permitem que você crie um HTML que "envolve" todas as páginas que estão na mesma pasta que ele ou abaixo. É muito útil para se criar cabeçalhos e rodapés comuns a muitas páginas, por exemplo.

Os layouts sempre tem o nome "+layout.svelte" e ficam dentro de qualquer pasta ou subpasta em "src/routes".

Exemplo de um layout:

src/routes/+layout.svelte

```
<H1>Minha Loja Virtual</H1>

<DIV>Menu da loja</DIV>

<DIV>

<!-- No lugar da tag slot, fica o conteúdo da página atual que se
está acessando -->

<slot></slot>
```



```
</DIV>

<DIV>
  Rodapé da loja
</DIV>
```

Repare que o Layout tem um tag chamada `<+page.svelte>` que é substituída pela página que estamos acessando. Exemplo, se você acessar a rota `/produtos/ativos/banana` o SvelteKit vai procurar pelo componente `"src/routes/produtos/ativos/[nomeDoProduto]/+page.svelte"` e irá renderizar seu conteúdo no lugar da tag `<+page.svelte>` do layout `"src/routes/+layout.svelte"` retornando o HTML completo, com o conteúdo do Layout com o conteúdo da página no centro. É um recurso muito prático e útil.

Para mais informações sobre o `+layout.svelte` verifique a documentação oficial: <https://kit.svelte.dev/docs/routing#layout>

Existem vários outros recursos no SvelteKit e vamos explorar ainda alguns deles na nossa loja on-line.

Pet Top Store - Loja on-line

O projeto da loja está disponível em `LINK_PARA_A_LOJA` e você pode utilizá-lo para acompanhar a aula. O projeto já está completo, com todos os recursos, inclusive os que serão vistos em detalhes na aula seguinte.

Criando o projeto da loja do zero

Se você deseja criar o projeto da loja do zero então faça:

```
npm create svelte@latest loja
```

```
[isaac@ixi pettopstore]$ npm create svelte@latest loja
create-svelte version 4.2.0

Welcome to SvelteKit!

◆ Which Svelte app template?
  ○ SvelteKit demo app
  ● Skeleton project (Barebones scaffolding for your new SvelteKit app)
  ○ Library project
```

Escolha a opção "Skeleton project" e Pressione Enter

```
[isaac@ixi pettopstore]$ npm create svelte@latest loja  
  
create-svelte version 4.2.0  
  
Welcome to SvelteKit!  
◇ Which Svelte app template?  
  Skeleton project  
◆ Add type checking with TypeScript?  
  ● Yes, using JavaScript with JSDoc comments  
  ○ Yes, using TypeScript syntax  
  ○ No
```

Escolha JavaScript with JSDoc comments e Pressione Enter

```
[isaac@ixi pettopstore]$ npm create svelte@latest loja  
  
create-svelte version 4.2.0  
  
Welcome to SvelteKit!  
◇ Which Svelte app template?  
  Skeleton project  
◇ Add type checking with TypeScript?  
  Yes, using JavaScript with JSDoc comments  
◆ Select additional options (use arrow keys/space bar)  
  □ Add ESLint for code linting  
  □ Add Prettier for code formatting  
  □ Add Playwright for browser testing  
  □ Add Vitest for unit testing
```

Não precisa marcar nenhum opcional. Pressione Enter

```
[isaac@ixi pettopstore]$ npm create svelte@latest loja

create-svelte version 4.2.0

Welcome to SvelteKit!

◇ Which Svelte app template?
  Skeleton project

◇ Add type checking with TypeScript?
  Yes, using JavaScript with JSDoc comments

◇ Select additional options (use arrow keys/space bar)
  none

Your project is ready!

✓ Type-checked JavaScript
https://www.typescriptlang.org/tsconfig#checkJs

Install community-maintained integrations:
https://github.com/svelte-add/svelte-add

Next steps:
  1: cd loja
  2: npm install (or pnpm install, etc)
  3: git init && git add -A && git commit -m "Initial commit" (optional)
  4: npm run dev -- --open

To close the dev server, hit Ctrl-C

Stuck? Visit us at https://svelte.dev/chat
```

Sistema criado!

Repare que é possível escolher outras opções na criação do projeto como iniciar com um sistema básico mais completo (diferente do skeleton), usar Typescript no lugar do Javascript, já usa o ESLint, Prettier, etc como dependências. Não vamos cobrir esses tópicos aqui, mas você pode pesquisar e estudar mais sobre essas tecnologias que podem ajudar no desenvolvimento de sistemas maiores. Vamos nos manter com o básico por enquanto.

Agora entre na pasta "loja" criada e instale as dependências:

```
cd loja
npm install
```

```
[isaac@ixi pettopstore]$ cd loja/  
[isaac@ixi loja]$ npm install  
  
added 97 packages, and audited 98 packages in 33s  
  
10 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Dependências instaladas

Mudando a porta padrão da loja para 5000

O Sveltekit por padrão usa a porta 5173 quando se inicia o servidor, mas você pode mudar isso. Vamos mudar essa porta para 5000 no nosso projeto.

O Sveltekit utiliza o Vite () como ferramenta base para iniciar o servidor, dentre outras coisas e é no arquivo de configuração do Vite que vamos trocar a porta padrão.

Edite o arquivo **vite.config.js** e troque esse trecho:

```
export default defineConfig({  
  plugins: [sveltekit()]  
});
```

Por esse:

```
export default defineConfig({  
  plugins: [sveltekit()],  
  server: {  
    port: 5000  
  }  
});
```

Isso fará que o servidor da loja agora inicia na porta 5000.

Se desejar, dê uma olhada no arquivo **package.json** e veja as dependências instaladas, assim como os scripts que já estão configurados.

Repare uma coisa interessante: No Sveltekit, a maioria das dependências sempre são instaladas como devDependencies, ou seja, se você está adicionando uma nova biblioteca com o npm, provavelmente precisará adicionar a opção **--save-dev** para salvar ela como dependências de

desenvolvimento no lugar de dependência comum. Isso ocorre pois o Svelte é um compilador, ou seja, ele durante o processo de build, converte seu código em um Javascript super otimizado. Portanto o node precisa ter acesso às bibliotecas extras que você adicionou no projeto durante essa compilação, e daí a necessidade deles estarem como devDependencies. Não se preocupe com isso agora, mas é algo que deve ser mencionado caso venha a criar outros sistemas com o Sveltekit.

Para iniciar o servidor com o projeto da loja digite:

```
npm run dev
```

```
[isaac@ixi loja]$ npm run dev

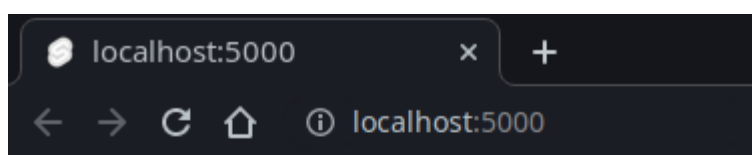
> loja@0.0.1 dev
> vite dev

VITE v4.3.5 ready in 314 ms

→ Local:   http://localhost:5000/
→ Network: use --host to expose
→ press h to show help
```

Servidor iniciado

Acesse a loja em <http://localhost:5000/>



Welcome to SvelteKit

Visit kit.svelte.dev to read the documentation

Abra o arquivo **src/routes/+page.svelte** e substitua o seu conteúdo por:

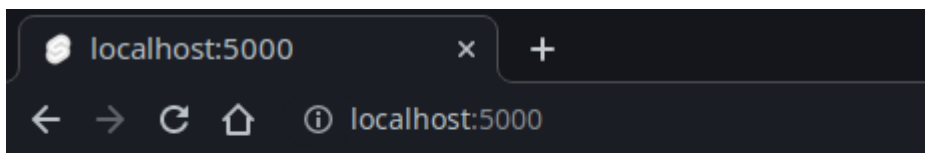
```
<style>
  .centro {
    font-size: 30px;
    padding: 30px;
    margin: auto;
  }

</style>

<div class="centro">
  A melhor Petshop do universo
</div>
```

Este arquivo é a página inicial da loja, que só mostra uma mensagem de boas vindas.

O resultado da sua loja deve ficar similar ao da imagem abaixo:



A melhor Petshop do universo

Loja on-line - Layout global

Agora vamos criar um arquivo de layout, ou seja, um arquivo que contém um conteúdo comum a todas as páginas da loja virtual, com cabeçalho, menu e espaço para o conteúdo das outras páginas (tag `<slot />`).

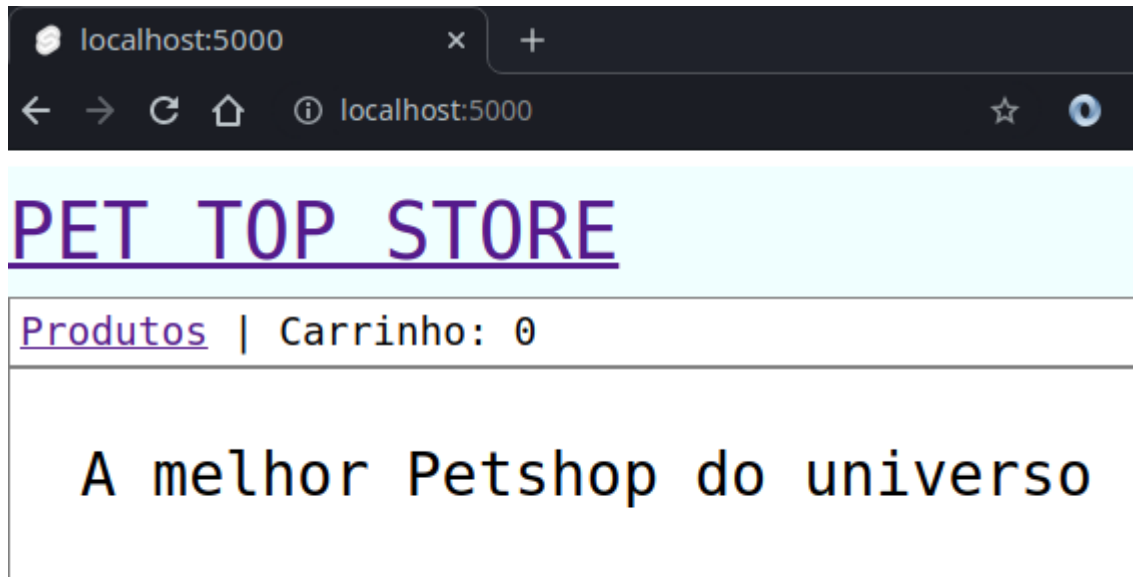
Para isso crie um arquivo chamado **src/routes/+layout.svelte** com o seguinte conteúdo:

```
<div>
  <div class="cabecalho">
    <a href="/">PET TOP STORE</a>
  </div>
  <div class="menu">
    <a href="/produtos">Produtos</a> |
    Carrinho: 0
  </div>
  <div class="conteudo">
    <slot />
  </div>
</div>

<style>
.cabecalho {
  background-color: azure;
  font-size: 3em;
  font-family: monospace;
  padding-top: 10px;
  padding-bottom: 10px;
}
.menu {
  font-size: 1.5em;
  font-family: monospace;
  border: 1px solid gray;
  padding: 5px;
}
.conteudo {
  font-size: 1em;
  font-family: monospace;
  border: 1px solid gray;
  padding: 5px;
}
</style>
```

Repare que mesmo o **src/routes/+layout.svelte** não tem as tags HTML, BODY, HEAD, etc. Isso se dá pois elas estão já escritas no arquivo **src/app.html** que é o ponto de entrada da aplicação. Dessa forma você precisa escrever ainda menos código e ter um sistema ainda mais modular e com reuso de código.

Agora a loja virtual deve ficar com um visual similar ao da imagem abaixo:



Veja que estamos utilizando alguns estilos CSS bem simples somente para ter uma melhor divisão dos elementos. Claro que você pode aplicar qualquer estilo que desejar ou até usar uma biblioteca de componentes UI que achar melhor. A nossa intenção na apresentação da loja on-line é manter o código para estilização mínimo e focar mais nas funcionalidades.

Você já pode ver que temos um cabeçalho com o nome PET TOP STORE, um menu com um link para produtos (ainda não funcional) e um resumo do carrinho de compras (que implementamos na aula seguinte).

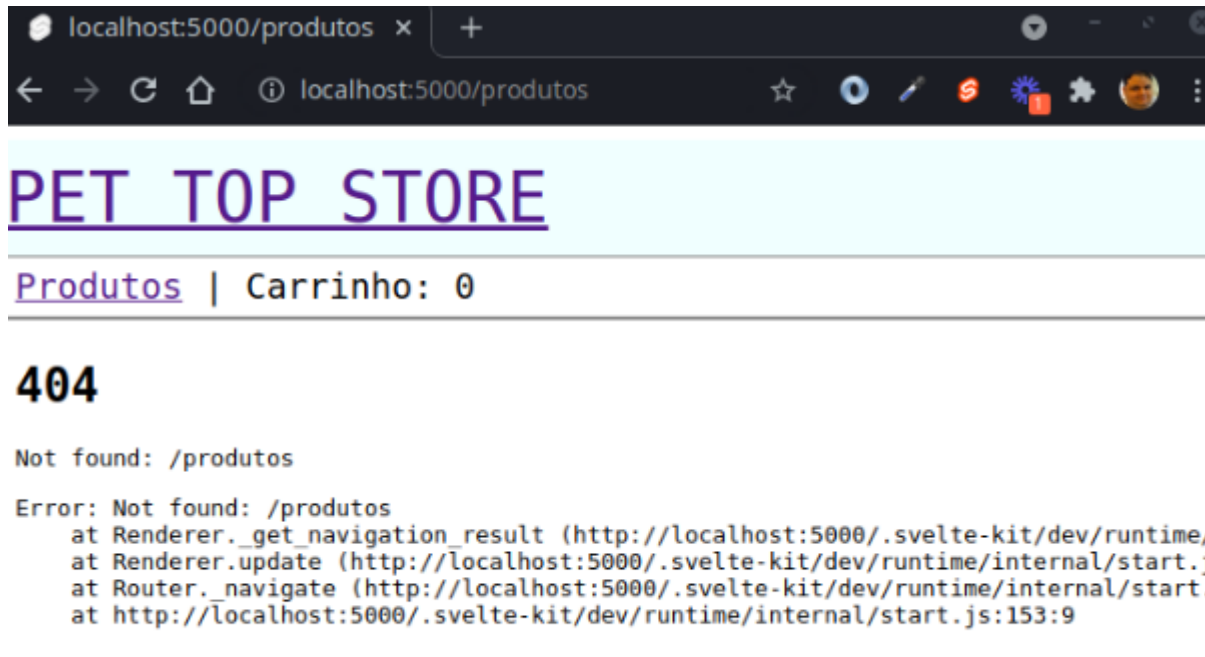
Esse conteúdo do layout será exibido em todas as páginas, sendo elas "embutidas" dentro do layout onde tem a tag `<slot />`.

Isso deixa você livre para criar páginas com um HTML/Javascript somente referente a página em si, e não se preocupar com o que é comum a todas pois o SvelteKit e o layout que criou toma conta disso pra você.

Loja on-line - lista de produtos

Até agora você criou um layout que já tem um link para a lista de produtos ("/produtos") mas essa página ainda não está criada e se você acessar esse link nesse momento obterá um resultado como o abaixo:

Acesse <http://localhost:5000/produtos>



Veja que o layout até foi exibido, mas o a página que seria renderizada no lugar da tag `<slot />` foi um erro 404, ou seja, a página não foi encontrada. Isso é esperado já que ela não existe ainda.

Para criar uma página que representa a rota `"/produtos"` é super fácil, vamos ver como.

Poderíamos simplesmente criar um arquivo chamado `produtos.svelte` dentro de `src/routes`, porém vamos fazer de uma segunda maneira:

Primeiro, crie uma PASTA chamada **`src/routes/produtos`** e então dentro dela iremos criar o arquivo **`+page.svelte`**. Dessa forma, quando o usuário visitar a URL `"/produtos"` esse arquivo será renderizado.

No arquivo chamado **`src/routes/produtos/+page.svelte`**, que mencionamos, e coloque o seguinte conteúdo nele:

```
<script>
  // Importando o hook onMount, responsável por executar uma função
  quando a página é carregada
  import { onMount } from "svelte";

  // Criando uma variável para armazenar os produtos
  let products = [];

  // Função que será executada quando a página for carregada
  onMount(async () => {
    // Fazendo uma requisição para a API, para buscar os produtos
```

```
const res = await
fetch('http://localhost:3000/api/products/search');

// Verificando se a requisição foi bem sucedida
if (res.ok) {
  // Se sim, pegamos o JSON da resposta
  const json = await res.json()
  // E armazenamos os produtos na variável criada anteriormente
  products = json.products;
} else {
  // Em caso de erro, limpamos a variável
  products = [];
}
});
</script>

<h1>Produtos</h1>

<div>
  {#each products as product}
    <div class="produto">
      <div class="image">
        
      </div>
      <div class="info">
        <div class="nome">
          {product.name}
        </div>
        <div class="descricao">
          {product.description}
        </div>
        <div class="categoria">
          Categoria:
          {product.categoryName || 'Sem Categoria'}<br/>
        </div>
      </div>

      <div class="detalhe">
        <a href="/produtos/{product.id}">Detalhar produto</a>
      </div>
    </div>
  {/each}
</div>
```

```
</div>

<style>
  .produto {
    border: 2px solid #ddd;
    margin: 10px;
    height: 100px;
    display: flex;
  }
  .produto .info {
    display: flex;
    flex-direction: column;
    margin: 10px;
  }
  .produto .detalhe {
    margin-left: auto;
    margin-top: 20px;
  }
  .produto .detalhe a {
    background-color: cadetblue;
    color: white;
    font-weight: bold;
    text-decoration: none;
    padding: 5px;
  }
</style>
```

Esse arquivo lista os produtos da loja, buscando os dados na API do projeto **admin**. Então é importante você tenha o projeto **admin** rodando na sua máquina (lembre que na porta 3000) para que a API esteja disponível para a loja on-line.

Ao salvar o arquivo **src/routes/produtos/+page.svelte** que criamos, e com a API rodando como mencionado, você deve acessar <http://localhost:5000/produtos> e obter um resultado como o da imagem abaixo:

PET TOP STORE

[Produtos](#) | Carrinho: 0

Produtos



Balança de ração
Categoria: Alimentos

[Detalhar produto](#)

Coleira
Categoria: Alimentos

[Detalhar produto](#)

Legal né? Temos um projeto administrativo (admin) com uma API servindo conteúdo para a nossa loja on-line que os exibe em um projeto diferente utilizando outras tecnologias.

O conteúdo do arquivo **src/routes/produtos/+page.svelte** é composto por três partes:

A primeira é um script que tem uma única variável de estado chamada **products**, que é iniciada com um array vazio. Nesse script, usamos o hook **onMount** do Svelte (importado no início do script) para rodar uma função quando a página é iniciada. Essa função, passada para o onMount, obtém os produtos da nossa API no endereço "<http://localhost:3000/api/products/search>" através de uma chamada fetch padrão do javascript e armazena o resultado (res.products) no array **products**, declarado anteriormente.

A segunda parte é um HTML simples utilizando a linguagem de template do Svelte para exibir o conteúdo do array "products" usando o comando `{#each products as product}` e a cada iteração ele cria um block de HTML com a imagem, nome, descrição e categoria do produto, além de um link para detalhar o produto por id: `Detalhar produto`

A terceira parte é simplesmente um estilo mínimo para página na tab `<style>`.

Observe que estamos usando aqui o hook onMount, disponível no Svelte e que executa somente quando a página já está no lado do cliente. O SvelteKit tem mecanismos mais avançados para carregar dados, que permite que suas páginas já sejam carregadas inicialmente com dados

renderizados pelo servidor, evitando que o navegador faça uma segunda requisição para obter os produtos e assim aumentando a performance da aplicação. Essa técnica se chama SSR (Server Side Rendering) e pode ser utilizada utilizando uma função chamada **load** implementada em um arquivo separado. Não iremos cobrir essa técnica nesse exemplo, mas você pode aprender mais sobre ela na documentação oficial do Sveltekit: <https://kit.svelte.dev/docs/load>

Loja on-line - Detalhar produto

Por enquanto se você clicar em "Detalhar produto" irá se deparar com uma mensagem de erro 404 pois a rota `/produto/ID_DO_PRODUTO` ainda não existe. Essa rota será uma rota dinâmica, ou seja, o `ID_DO_PRODUTO` pode mudar, mas exibiremos a mesma página sempre, porém com um parâmetro que será o ID do produto para cada visita.

Para se criar essa rota dinâmica no SvelteKit basta criar uma pasta chamada **src/routes/produtos/[id]** e dentro dela um arquivo chamado **src/routes/produtos/[id]/+page.svelte**. Observe que o nome do arquivo é assim mesmo `"[id]"` (com esses colchetes) e é uma subpasta de `"src/routes/produtos"` e o arquivo dentro dessa pasta se chama `"+page.svelte"`, como todas as outras páginas.

O arquivo **src/routes/produtos/[id]/+page.svelte** terá o seguinte conteúdo:

```
<script>
  // importando onMount do svelte, para executar uma função quando a
  página for carregada
  import { onMount } from 'svelte';
  // importando o store page, para pegar o id do produto
  import { page } from '$app/stores';

  // criando uma variável para armazenar o ID do produto, que vem da
  URL
  const productId = $page.params.id;

  // criando uma variável para armazenar o produto, que será buscado
  na API
  let product;

  // função que será executada quando a página for carregada
  onMount(async () => {
    // fazendo uma requisição para a API, para buscar o produto
```

```
const res = await
fetch(`http://localhost:3000/api/products/${productId}`);

// verificando se a requisição foi bem sucedida
if (res.ok) {
  // se sim, pegamos o JSON da resposta
  const json = await res.json();
  // e armazenamos o produto na variável criada anteriormente
  product = json.product;
}
})

</script>

<a href="/produtos">Voltar</a>

{#if product}
  Nome: {product.name}

  {product.name}<br/>
  {product.description}<br/>
  
  {product.categoryName || 'Sem Categoria'}<br/>
  Link para Adicionar no carrinho não implementado ainda
{:else}
  Carregando produto...
{/if}
```

Você deve obter um resultado similar a esse:



Repare que no exemplo da imagem estamos visitando <http://localhost:5000/produtos/4>, ou seja, visitando o produto com id=4. No seu banco de dados esse link pode ser diferente, com outro id, dependendo do produto que você clicar no link de detalhar, mas o resultado deve ser similar, mudando somente o conteúdo, claro.

O arquivo **src/routes/produtos/[id]/+page.svelte** é bem simples, tem um script com uma variável de estado chamada **productId**, que é inicializado com o ID do produto passado como parâmetro da rota dinâmica. Esse ID é obtido através de uma store especial do SvelteKit chamada "page".

Stores no Svelte e no SvelteKit são objetos especiais que guardam dados que podem ser compartilhados entre componentes e páginas.

Veja mais sobre stores no Svelte em: <https://svelte.dev/docs#run-time-svelte-store>

Veja mais sobre as stores que o SvelteKit já traz prontas em: [https://kit.svelte.dev/docs/modules#\\$app-stores](https://kit.svelte.dev/docs/modules#$app-stores)

A store "page" é uma store especial do SvelteKit onde você pode acessar informações como: host, path, params e query da requisição atual para a página.

Para acessar valores de stores é necessário usar o "\$" na frente do nome da store, a não ser que ela esteja sendo usada dentro da função "load" que é um recurso do SvelteKit que não estaremos usando nessa aula, mas que você pode aprender mais na documentação já mencionada.

Então em resumo para se obter o id do produto nessa nossa página basta importar a store "page" com o comando `import {page} from '$app/stores'`; e depois pegar o parâmetro "id" e colocar em uma variável com o comando `const productId = $page.params.id;`. Lembre que esse parâmetro da URL se chama "id" pois existe uma pasta chamada "[id]" dentro de src/produtos (rota dinâmica)

A outra variável que existe nessa página é "product" que guardará o objeto com o produto que vem da API, mas que na sua declaração "let product;" não tem nada ainda.

Existe também o uso do hook onMount que roda quando a página é inicializada e para ele se passa uma função que carrega o produto com o ID obtido do parâmetro da página (exemplo /produto/4 torna o \$page.params.id igual 4) direto da API com o uso da função fetch e o resultado é colocado na variável **product**.

A parte do HTML da página iniciar verificando se o produto já está definido com o comando `{#if product}` e, caso esteja, é exibido um HTML utilizando a linguagem de template do Svelte com o nome, descrição, categoria e imagem do produto. Caso não esteja, é porque ele ainda não foi carregado da API e uma mensagem

"Carregando produto..." aparece brevemente enquanto os dados do produto não são retornados pela API. Usamos o `{:else}` da linguagem de template do Svelte para isso.

O interessante dessa abordagem é que assim que a página é exibida o produto é indefinido e a mensagem "Carregando produto..." aparece para o usuário, mas assim que os dados retornam da API, o produto é definido no comando `"product = json.product;"` e o Svelte reage, exibindo os dados do produto.

Repare que existe no HTML um espaço para um botão de adicionar no carrinho, mas iremos criar esse botão somente na próxima aula.

Existe nessa página também um link para "Voltar" para a lista de produtos.

Vimos nessa aula uma introdução rápida ao Svelte e SvelteKit e também já iniciamos o desenvolvimento da nossa loja on-line da Pet Top Stop utilizando o SvelteKit e acessando a API criada no admin.

Na próxima aula iremos finalizar a loja on-line e o projeto da Pet Top Store.

Como também informado na apresentação da aula, você já pode ter acesso ao projeto completo no link:

Projeto admin: [Clique aqui para download](#)

Projeto PDV: [Clique aqui para download](#)

Projeto da loja: [Clique aqui para download](#)