Plataformas de aplicações Web

Aula 15 - PetTopStore - Loja on-line - Parte 2

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Apresentação

Nessa nossa última aula, iremos finalizar o projeto da PetTopStore com a última parte da loja on-line criada com o SvelteKit e acessando a API do projeto admin.

Lembrando que temos 3 projeto na PetTopStore

- Admin+API criado com Express+EJS+knex, rodando na porta 3000
- PDV criado com React e acessando a API do Admin, rodando na porta 4000
- Loja on-line criado com SvelteKit e acessando a API do Admin, rodando na porta 5000

Portanto, para você testar o projeto completo na sua máquina é necessário ter todos eles rodando ao mesmo tempo e acessíveis em portas distintas.

Segue os links para os 3 projetos já completos:

Admin: Clique aqui para download

PDV: Clique aqui para download

Loja: Clique aqui para download

Observação: O projeto admin nesse link está com uma pequena correção de um erro de quando se fazia uma busca de produtos passando um termo de busca na API. Caso você deseje usar esse recurso ele está nessa versão funcionando corretamente.

Nesta aula iremos simplesmente explicar os códigos dos arquivos que estão implementados na loja final disponível no link acima, adicionado o recurso de carrinho de compras, cadastro de cliente e finalizar uma compra do carrinho.

Loja on-line - Carrinho de compras

Antes de falar sobre a implementação do carrinho de compras é importante aprendermos sobre um recurso muito importante do Svelte: stores

Stores

No Svelte, existe um recurso muito útil chamado stores, que são um mecanismo de compartilhar dados entre componentes. No caso do SvelteKit você pode usar stores para compartilhar dados entre componentes de em uma mesma página ou em páginas diferentes.

Uma store pode ser do tipo "writable", "readable" e "derived", e representam dados que são alteráveis, somente leitura ou derivados de outras stores, respectivamente.

Um exemplo de uma store que guarda pro exemplo uma lista de frutas pode ser criado em um projeto qualquer em um arquivo javascript chamado mercado_store.js com o seguinte conteúdo:

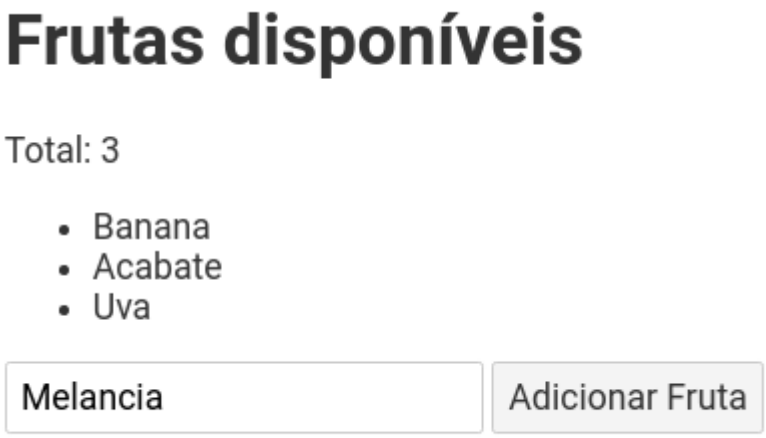
```
import { writable } from 'svelte/store';

// criação e exportação de uma store chamada frutasDisponiveis, com
o valor inicial sendo um array vazio
export const frutasDisponiveis = writable([]);
```

Esse arquivo exporta uma variável (store) chamada frutasDisponiveis que poderá ser importada por qualquer componente, em qualquer página, para acessar ou alterar o seu conteúdo basta acessar a variável importada com um $ na frente assim: $frutasDisponiveis. Se o seu valor alterar em um conponente, todos os outros que a utilizam irão reagir a essa mudança alterando o seu HTML de acordo.

Exemplo de um componente acessando e mudando o valor de frutasDisponiveis:

```
<script>
    import { frutasDisponiveis } from './mercado_store.js'
    let novaFrutaNome;
</script>

<h1>Frutas disponíveis</h1>
Total: {$frutasDisponiveis.length}<br/>

<ul>
    {#each $frutasDisponiveis as fruta}
    <li>{fruta}</li>
    {/each}
</ul>

<input type="text" bind:value={novaFrutaNome} />

<button on:click={() => {
        $frutasDisponiveis = [...$frutasDisponiveis, novaFrutaNome];
        novaFrutaNome = '';
    }}
>
    Adicionar Fruta
</button>
```

Resultado:



Repare que no exemplo se adiciona uma nova fruta na store (qua guarda um array como dados) de forma muito similar a alterar o valor de um array normal:

$frutasDisponiveis = [...$frutasDisponiveis, novaFrutaNome];

Só para deixar claro, esse exemplo de stores não faz parte do projeto da PetTopStore e não precisa ser adicionado ao projeto da loja on-line.

Para saber mais sobre sobre stores acesse:
https://svelte.dev/docs#run-time-svelte-store

Outra fonte interessante para se aprender mais sobre stores é você seguir o tutorial oficial do Svelte no site https://svelte.dev/tutorial até chegar a sessão sobre stores. Recomendo que faça o tutorial todo para um aprendizado melhor guiado.

As stores no Svelte usam a sintaxe com o $ para serem acessadas de forma mais simplificada, como se fossem uma simples variável com os dados. Porém isso é um artefato que pode não ser usado e por debaixo dos panos existem métodos especiais em uma store, como o "subscribe" que é chamado quando o valor dela muda com o método "set". Não vamos entrar em muitos detalhes, e você precisa estudar mais sobre esses métodos caso deseje.

Store do Carrinho de compras

É importante observar que os dados escritos nas stores são compartilhados entre páginas e componentes, mas somente enquanto o usuário não efetua um reload completo no navegador (está navegando somente clicando nos links). Para que os dados fiquem persistidos no navegador é necessário se usar o LocalStorage como já vimos anteriormente. No projeto da loja on-line vamos criar um carrinho de compras como uma store especial que persiste os seus dados no LocalStorage quando o seu valor muda, e tem um valor inicial também resgatado do LocalStorage. Isso faz com que possamos usar o recurso de Stores no SvelteKit, que é muito últil para compartilhamento entre componentes, mas que eles fiquem

persistidos no navegador mesmo quando a página é fechada.

Para isso existe um arquivo em **src/stores/carrinho.js** com o seguinte conteúdo:

```
import { browser } from '$app/environment';
import { writable } from 'svelte/store'

// valor padrão é uma lista vazia de produtos
let carrinhoInicial;

if (browser) {
  const dados = localStorage.getItem('carrinho');
  try {
    carrinhoInicial = dados ? JSON.parse(dados):[];
  } catch {
    carrinhoInicial = [];
  }
} else {
  carrinhoInicial = [];
}

export const carrinho = writable(carrinhoInicial);

carrinho.subscribe((carrinhoAtualizado) => {
  if (browser) {
    localStorage.setItem('carrinho',
JSON.stringify(carrinhoAtualizado));
  }
})
```

Essa store especial chamada "carrinho" tem algumas partes importantes no código:

Primeiramente é crido uma variável chamada carrinhoInicial, sem valor definido.

Depois é feita uma checagem if (browser). Isso pergunta se o código está rodando no navegador e não no servidor. Isso é importante pois o LocalStorage não está disponível no servidor (somente no navegador) e o SvelteKit tem recursos de renderização de páginas no servidor (SSR), mas não estamos usando isso no nosso projeto. De toda forma é necessário fazer essa checagem para não rodar código no servidor que é incompatível.

Se o localStorage está disponível, é resgatado o valor do carrinho dele com o comando:

```
const dados = localStorage.getItem('carrinho');
 try {
    carrinhoInicial = dados ? JSON.parse(dados):[];
  } catch {
    carrinhoInicial = [];
  }
```

Isso pega os dados do localStorage e tenta converter de JSON para objeto Javascript os dados. Caso um erro acontece, um array vazio é usado no lugar (representando um carrinho vazio).

Depois de definir nesses códigos acima o valor de "carrinhoInicial" é criada a store "carrinho" com esse valor e se configurado o método .subscrtibe que será invocado quando o valor mudar (um item for adicionado ao carrinho, por exemplo), alterando o localStorage com o novo valor também com o comando:

```
localStorage.setItem('carrinho',
JSON.stringify(carrinhoAtualizado));
```

Essa store do carrinho é assim sincronizada com o localStorage do navegador e mesmo que você feche a página e abra novamente, seus itens do carrinho estarão ainda lá, até que você deseje limpar o carrinho setando a store para um array vazio.

Layout - Exibir total de produtos no carrinho

Para exibir o total de itens no carrinho altere o arquivo **src/routes/+layout.svelte** e deixe-o com o seguinte conteúdo:

```
<script>
  import { carrinho } from '../stores/carrinho';
</script>

<div>
  <div class="cabecalho">
    <a href="/">PET TOP STORE</a>
  </div>
  <div class="menu">
    <a href="/produtos">Produtos</a> |
    Carrinho: {$carrinho.length}
    <a href="/carrinho">Ver itens</a>
  </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 só foi adicionado a importação da store do carrinho com:

```
import { carrinho } from '../stores/carrinho';
```

E depois o tamanho da store (que é um array), representando o total de itens nela, foi exibida no menu com:

```
Carrinho: {$carrinho.length}
    <a href="/carrinho">Ver itens</a>
```

Veja que também já foi criado um link para "/carrinho" rota que vamos criar mais a frente.

Adicionar produto no carrinho

Para adicionar produtos no carrinho de compras o arquivo **src/routes/produtos/[id]/+page.svelte** foi alterado e ficou com 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';

  // importando o store carrinho
  import { carrinho } from '../../../stores/carrinho';

  // 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;
    }
  })

  async function adicionarCarrinho(product) {
    // criando um novo carrinho com os valores antigos adicionados
do novo producID
    $carrinho = [...$carrinho, product];
```

```
    alert('Adicionado ao carrinho com sucesso!');
  }

</script>

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

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

  {product.name}<br/>
  {product.description}<br/>
  <img width="100px" alt="imagem do produto"
src="http://localhost:3000/images/{product.photo}" />
  {product.categoryName || 'Sem Categoria'}<br/>
  <button
    on:click={() => adicionarCarrinho(product)}
  >Adicionar no carrinho</button>
{:else}
  Carregando produto...
{/if}
```

Repare que foi importado a store carrinho:

```
import { carrinho } from '../../../stores/carrinho';
```

E que a função adicionarCarrinho(product) recebe um produto e adiciona na store do carrinho com o comando:

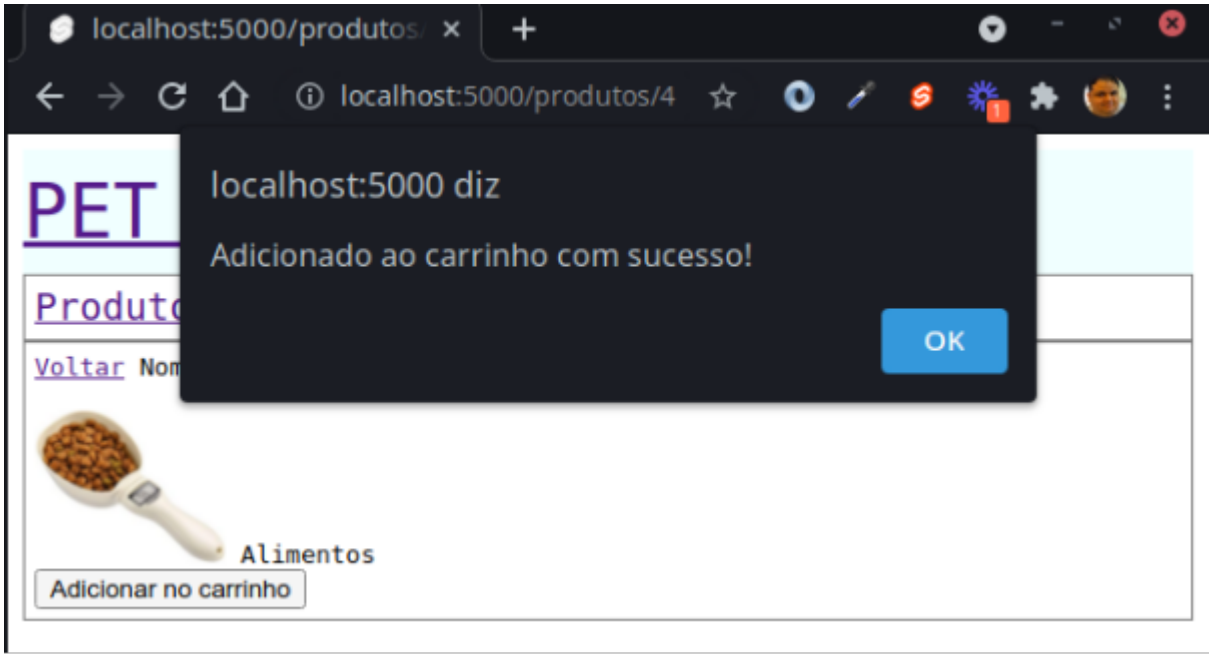
```
$carrinho = [...$carrinho, product];
```

Simples assim. Esse comando altera a store do carrinho adicionando mais um produto no array e o menu da loja, que está implementado no layout e mostra o total de itens do carrinho, já deve reagir e exibir o novo total caso for adicionado um produto novo nessa store.

Repare que nessa página também foi adicionado um botão que chama a função adicionarCarrinho passando o produto que está sendo visualizado:

```
<button
    on:click={() => adicionarCarrinho(product)}
  >Adicionar no carrinho</button>
```

O resultado da página de detalhar produto (/produto/1 por exemplo) deve ficar assim quando se pressiona o botão "Adicionar no carrinho":



Logo depois a contagem de itens no carrinho deve mudar para "1" no menu, assim:

Ver carrinho e finalizar compra

No menu existe um link chamado "Ver itens" do carrinho que leva para "/carrinho". Essa rota implementa 4 funcionalidades:

- Listar itens no carrinho
- Limpar carrinho
- Finalizar compra com e-mail/senha do cliente
- Link para "Quero me cadastrar" que é uma página/rota separada e cadastra o cliente caso não tenha cadastro ainda.

O conteúdo de **src/routes/carrinho/+page.svelte** é:

```
<script>

  import { carrinho } from '../../stores/carrinho';

  // dados do usuário
  let usuario = {
    email: '',
    senha: ''

  };

  async function limparCarrinho() {
    $carrinho = [];
  }

  async function finalizarCompra() {
    const productIDs = $carrinho.map((product) => product.id);
    const res = await
fetch(`http://localhost:3000/api/auth/client/sign_in`, {
        method: 'post',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          email: usuario.email,
          password: usuario.senha
        })
      });
      if (res.ok) {
        const json = await res.json();
        const client = json.client;
        const res2 = await fetch(`http://localhost:3000/api/sales`,
{
          method: 'post',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${client.token}`
          },
          body: JSON.stringify({
            productIDs: productIDs
          })
        });
```

```
        if (res2.ok) {
          const json2 = await res2.json();
          alert(`Compra codigo ${json2.sale.id} realizada com
sucesso`);
          $carrinho = [];
        } else {
          alert('Erro realizando compra')
        }
      } else {
        alert('Erro no e-mail/senha')
      }

  }

  </script>

  <h2>Carrinho de compras</h2>

  {#if $carrinho && $carrinho.length > 0}

  <br/>
    Itens: {$carrinho.length}<br/>
    <button on:click={limparCarrinho}>
      Limpar Carrinho
      </button>
    <ul>
    {#each $carrinho as product}
      <li>
        <img width="100px" alt="imagem do produto"
src="http://localhost:3000/images/{product.photo}" align="center"/>
        {product.name}
      </li>
    {/each}
    </ul>

    <h2>Finalizar compra</h2>

    <a href="/cadastro">Quero me cadastrar</a><br/>
    E-Mail: <input bind:value={usuario.email} /><br/>
    Senha: <input bind:value={usuario.senha} /><br/>
    <button on:click={finalizarCompra}>
      Finalizar Compra
    </button>
```

```
  {:else}
    Carrinho vazio

  {/if}
```

Essa página é a maior da loja on-line, mas observe que cada função realiza uma tarefa muito específca.

limparCarrinho() simplesmente atribui um array vazio à store $carrinho, e isso é suficiente para o limpar, inclusive no localStorage, como já vimos.

finalizarCompra() primeiramente extrai somente os IDs dos produtos que estão no carrinho e os guardam em productIDs. Logo em seguida, realiza 2 acessos à API: No primeiro ela tenta autenticar o cliente acessando http://localhost:3000/api/auth/client/sign_in com os dados digitados no formulário de finalizar compra (email e senha). Caso a autenticação passe com sucesso ele tenta fazer outro acesso a API, porém agora tentando criar uma venda acessando http://localhost:3000/api/sales com uma lista de IDs de produtos que foi extraído dos produtos do carrinho e também passando o token recebido no primeiro acesso (tentativa de autenticação).

Caso finalizarCompra() consiga realizar a operação com sucesso, ele ao final limpa o carrinho de compras com:

```
$carrinho = [];
```

Além dessas duas funções temos um HTML que simplesmente lista todos os produtos no carrinho, iterando nele com {#each $carrinho as product} e mostrando a sua imagem e nome.

No final do HTML temos um formulário para finalizar compras, solicitando o e-mail e senha do cliente e um botão que chama finalizarCompra().

Repare que também existe um link para "/cadastro" que é nossa última rota que precisa ser implementada.

Sua rota "/carrinho" deve ficar similar a essa (Exemplo já com 2 produtos quaisquer no carrinho):

Cadastro do cliente

Ao clicar em "Quero me cadastrar" na página do carrinho de compras, você é levado para "/cadastro" que deve ser implementado com o arquivo **src/routes/cadastro/+page.svelte** com o seguinte conteúdo:

```
<script>
  let usuario = {
    name: '',
    email: '',
    password: ''
  }
  async function cadastrarUsuario() {
    const res = await
fetch(`http://localhost:3000/api/auth/client/registration`, {
```

```
      method: 'post',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        name: usuario.name,
        email: usuario.email,
        password: usuario.password
      })
    });
    if (res.ok) {
      const json = await res.json();
      alert('Usuário cadastrado com sucesso', json);
      window.location.href = "/carrinho";
    } else {
      alert('Erro cadastrando usuário');
    }
  }

</script>

<h1>Cadastro de usuário</h1>

<form>
  Nome: <input type="text" bind:value={usuario.name} /><br/>
  E-Mail: <input type="email" bind:value={usuario.email} /><br/>
  Senha: <input type="password" bind:value={usuario.password}
/><br/>
  <button on:click|preventDefault={cadastrarUsuario}>
    Me cadastrar
  </button>

</form>
```

Esse arquivo tem no HTML um formulári ocomun, com os campos (INPUTs) associados aos atributos da variável de estado "usuario" (name, email, password) e quando se clica em "Me cadastrar" a função cadastrarUsuario() é chamada realizando um acesso à API em "http://localhost:3000/api/auth/client/registration", passando o nome, e-mail e senha do usuário. Caso a tentativa de cadastro seja realizada com sucesso o usuário é redirecionado para "/carrinho" novamente com o comando:

```
      window.location.href = "/carrinho";
```

O SvelteKit tem outras formas nativas de navegação mas é importante mostrar que operações comuns do Javascript funcionam bem com o SvelteKit, como por exemplo o comando do código acima.

Com o usuário cadastrado você pode agora finalizar a compra digitando o email e senha dele no carrinho de compras.

Página de cadastro do usuário:

Conclusão

Finalizamos aqui a nossa disciplina e o nosso último projeto, a Pet Top Store.

Links com as 3 aplicações do projeto final:

Admin: Clique aqui para download

PDV: Clique aqui para download

Loja: Clique aqui para download

Aprendemos os passos iniciais de diversas tecnologias desde bibliotecas de componentes UI, passando por frameworks front-end, back-end e full stack, utilizando diversas arquiteturas e até linguagens de programação diferentes.

A intenção dessa disciplina é de dar uma visão ampla sobre como tecnologias diferentes podem fazer parte de um projeto mais complexo em uma plataforma baseada em Internet e padrões abertos de interoperabilidade de dados. Nosso projeto final da PetTopStore tentou mostrar justamente isso: que tecnologias diferentes podem sim fazer parte de um mesmo projeto geral e trocar dados entre si, de maneira segura, e com o objetivo de entregar uma solução mais completa.

Espero que tenham tido a oportunidade de conhecer algo novo e que tenha despertado um novo interesse de expansão do seu aprendizado.

Até a próxima!