**Relatório Projeto 2: Lista de Compras**

Alunos: José Ricardo Schmitz Baptista e Pedro de Goes Furmam

1. **Formulário de Login**

* HTML

Para a parte de HTML foi utilizada a tag **<form>** para a criação de um formulário com os campos “usuário”, “senha”, um campo de checkbox para permanecer conectado e um botão para a confirmação do login.

* JavaScript

Na seção de JavaScript, o código começa selecionando elementos HTML específicos da página por meio de seus IDs e atribuindo-os a variáveis usando o método **document.getElementById()**. Essas variáveis são então utilizadas para acessar os elementos do formulário de login.

Em seguida, é adicionado um evento de clique ao botão de envio do formulário. Quando o botão é clicado, é acionada uma função que previne o comportamento padrão do formulário, que é recarregar a página, utilizando o comando **e.preventDefault()**.

Posteriormente, duas variáveis, "usuário" e "senha", são criadas para armazenar os valores dos campos do formulário. Isso é feito usando **const**, indicando que essas variáveis não serão reatribuídas.

Para verificar o login, é necessário recuperar os dados armazenados localmente relacionados ao usuário que está tentando fazer login. Para isso, o método **localStorage.getItem()** é utilizado para recuperar os dados de um campo específico armazenado no navegador. Além disso, é usado o método **JSON.parse()**, que analisa uma *string* JSON e a transforma em um objeto JavaScript. Isso permite acessar e manipular facilmente os dados armazenados no formato de objeto.

Essa parte foi feita pelo aluno José Ricardo.

1. **Cadastro de usuários**

* HTML

Da mesma maneira que o Login, foi criado um formulário com os campos “usuário” e “senha” para o cadastro dos novos usuários. Também foi adicionado o botão de “cadastrar”.

* JavaScript

Alguns procedimentos foram os mesmos explicados da criação do login, como a seleção dos elementos da página HTML e o evento de click do botão.

Nessa seção do cadastro foi definida uma classe chamada “Usuario” com o comando **constructor** que recebe dois parâmetros: “usuário” e “senha”. Essa classe é usada para criar objetos que representam usuários com suas credenciais.

Em seguida, um novo objeto “Usuario” é criado, passando os valores dos campos de usuário e senha do formulário de cadastro como argumentos para o **constructor** da classe, isso é feito por meio do comando **new Usuario().**

Por último, é armazenado os dados do novo usuário localmente no navegador. O método **setItem** do objeto **localStorage** é usado para armazenar os dados, onde a chave é o nome de usuário e o valor é um objeto “Usuario” convertido em uma *string* JSON usando **JSON.stringify()**.

Esse procedimento foi realizado pelo aluno Pedro.

1. **Cadastro, edição e remoção de listas de compras**

* HTML

Para o cadastro das listas feito pelo usuário foi criado um formulário com o campo “Nome da Lista”, onde o usuário pode inserir o nome que deseja. Para confirmar foi criado o botão “Adicionar”, quando clicado ele gera uma nova lista com o respectivo nome, id e data.

Para a edição, remoção e visualização das listas foi criado um campo na tabela com o nome “Ações”. Dentro do campo há dois botões, “Editar”, “Deletar” e “Ver lista”.

Para a visualização das listas criadas na tela foi utilizado a *tag* **<table>** para a criação de uma tabela de listas onde elas ficarão organizadas verticalmente.

* JavaScript

Novamente nesta seção existe alguns procedimentos já mencionados anteriormente como a seleção dos elementos da página HTML, o evento de click do botão e a captura de dados do formulário.

Para a criação das listas foi utilizado algumas funções. **obterObjetoUsuario(usuarioLogado)**: Esta função recebe o nome do usuário logado como parâmetro e retorna o objeto de usuário correspondente, recuperado do armazenamento local. **addCelula(linha, conteudo)**: Esta função recebe uma linha da tabela HTML e o conteúdo a ser adicionado como parâmetro e adiciona uma nova célula a essa linha com o conteúdo fornecido. **addBotoes(celula, id)**: Esta função recebe uma célula da tabela HTML e o id da lista como parâmetros. Ela adiciona botões de ação (deletar, editar nome, ver lista) a essa célula. **criarBotao(texto, acao)**: Esta função recebe o texto do botão e a ação a ser executada quando o botão é clicado como parâmetros. Ela cria um elemento de botão HTML, define seu texto e adiciona um ouvinte de evento para a ação fornecida. **verItensLista(id)**: Esta função recebe o id da lista a ser visualizada. Ela armazena o id e o nome da lista em edição no armazenamento local e redireciona o usuário para outra página HTML para visualizar os itens da lista. **criarNovaLista(nomeLista)**: Esta função recebe o nome da nova lista como parâmetro e cria um novo objeto de lista com um id aleatório e a data atual. **criarNovaListaAcao(e)**: Esta função é chamada quando o formulário para criar uma nova lista é enviado. Ela obtém o nome da lista do formulário, cria uma nova lista, adiciona-a ao objeto de usuário, atualiza o armazenamento local e adiciona uma nova linha à tabela HTML exibindo as informações da nova lista.

Três funções foram criadas para preencher a tabela de listas, uma delas é **generateRandomId()**, sua função é gerar um ID aleatório usando a API Crypto. Ela cria um *array* de bytes aleatórios e o converte em uma *string* hexadecimal. A outra função é **getCurrentDate()** que obtém a data atual e a formata no formato "dd/mm/aaaa" (dia/mês/ano) usando o método **toLocaleDateString()**. A última função é a **addRow()**, ela adiciona uma nova linha à tabela com uma nova lista de compras de acordo com os parâmetros fornecidos: ID, nome da lista e data de criação. Uma nova linha é inserida no corpo da tabela usando o método **insertRow()**. Para cada nova linha, são inseridas células (**<td>**) usando o método **insertCell()**. Neste caso, são criadas quatro células para cada nova linha. Os valores das variáveis id, nome e criadoEm são atribuídos ao conteúdo HTML das células correspondentes usando a propriedade **innerHTML**.

Para a edição de lista de compra foi criada a função **editarNomeLista(id)**: Esta função recebe o id da lista cujo nome deve ser editado. Ela permite que o usuário edite o nome da lista, atualiza o armazenamento local e atualiza o nome da lista na tabela HTML.

Para a remoção de uma lista foi criada a função **deletarLista(id)**: Esta função recebe o id da lista a ser deletada como parâmetro. Ela remove a lista correspondente do objeto de usuário, atualiza o armazenamento local e recarrega a página.

Essa seção foi desenvolvida pelos dois alunos.

1. **Cadastro, edição e remoção de itens.**

* HTML

Da mesma maneira que as listas, foi criado um formulário com o campo “Nome do item” para adicionar o nome do item e uma tabela para os exibir verticalmente na página. Para confirmar foi criado o botão “Adicionar”, quando clicado é adicionado o nome do item à lista.

Foi adicionado o campo “Ações” na tabela de itens com os botões “Editar” e “Deletar” para a edição e remoção dos itens.

* JavaScript

Algumas funções com os mesmos procedimentos das listas foram utilizadas para os itens como **obterObjetoUsuario(usuarioLogado)** e **addItem.addEventListener()** para o evento de clique do botão.

Para a visualização dos itens foram criadas as funções: **exibirID()**: Esta função obtém o ID da lista em edição do armazenamento local e exibe na página HTML. **exibirNome()**: Esta função obtém o nome da lista em edição do armazenamento local e exibe na página HTML. exibirItens(): Esta função exibe os itens da lista em edição na página HTML. Primeiro, ela obtém o objeto de usuário e o ID da lista em edição do armazenamento local. Em seguida, procura a lista correspondente ao ID e itera sobre seus itens, adicionando-os à tabela na página HTML e incluindo botões de edição e exclusão.

Para a edição e remoção de itens foram criadas duas funções: **deletarItem(nomeItem)**: Esta função é chamada para deletar um item da lista em edição. Ela recebe o nome do item como parâmetro, encontra a lista correspondente, remove o item dessa lista, atualiza o armazenamento local e remove a linha da tabela HTML correspondente ao item. **editarItem(nomeItem)**: Esta função é chamada para editar o nome de um item da lista em edição. Ela recebe o nome do item como parâmetro, solicita um novo nome ao usuário, atualiza o nome do item na lista, atualiza o armazenamento local e atualiza o nome do item na tabela HTML.

Por último, outras duas funções foram utilizadas para a adição dos itens á lista: **addCelula(row, value)**: Esta função adiciona uma nova célula a uma linha da tabela HTML com o valor fornecido**. addBotoesItem(row, itemName)**: Esta função adiciona botões de edição e exclusão a uma linha da tabela HTML para um item específico. Cada botão tem um ouvinte de evento associado para realizar a ação correspondente (editar ou deletar o item).

Essa seção foi desenvolvida pelos dois alunos.

1. **CSS**

Para a parte CSS foi utilizado alguns comandos para melhoras a visualização das páginas.

* **body**: Define o estilo do corpo da página, incluindo a fonte, cor de fundo e cor do texto.
* **h1**, **h3**: Define o estilo para os cabeçalhos de nível 1 e 3, respectivamente, especificando a cor do texto.
* **input[type="submit"]**: Define o estilo para botões de envio em formulários, especificando padding, cor de fundo, cor do texto, borda, efeito do cursor e borda arredondada.
* **form**: Define o estilo para formulários, especificando a margem inferior.
* **label**: Define o estilo para rótulos de formulário, especificando o peso da fonte como negrito.
* **input[type="text"]**, **input[type="password"]**: Define o estilo para campos de entrada de texto e senha, especificando a largura, preenchimento, margem inferior, borda e borda arredondada.
* **table**, **th**, **td**: Define o estilo para tabelas e suas células, especificando largura, colapso de borda, preenchimento, bordas e cor de fundo de cabeçalho.
* **tbody** **tr:nth-child(even)**: Define o estilo para linhas pares da tabela, especificando a cor de fundo.
* **#IDfield**, **#login**, **#cadastro**, **#dashboard**: Define o estilo para os elementos com IDs específicos, como largura, posição e translação.

Essa seção foi desenvolvida pelos dois alunos.