UNIVERSIDADE ESTADUAL DE MARINGÁ

DEPARTAMENTO DE INFORMÁTICA

Ingrid Lohmann

RA 117698

Trabalho II: Gerenciador de tarefas

Professor Paulo Roberto de Oliveira

Maringá – PR

2023

Introdução

O projeto de gerenciador de tarefas foi desenvolvido utilizando html e sass para a codificação do front-end e javascript orientado a objetos para o desenvolvimento da lógica de gerenciamento. Dos conceitos estudados em sala de aula sobre programação orientada a objetos foi utilizado, principalmente, o conceito de classes de objetos.

Desenvolvimento

O projeto foi desenvolvido em Javascript aplicando os conceitos de POO. O projeto consta com os seguintes arquivos:

1. index.html;
2. styles.scss
3. script.js
4. todoPage.html
5. todoStyles.scss
6. todoScript.js
7. reportPage.html
8. reportStyles.scss
9. reportScript.js

Os arquivos de 1 a 3 são referentes a tela de login, os de 4 a 6 são relativos ao gerenciador de tarefas e os de 7 a são responsáveis pela renderização do relatório em tela. Vale ressaltar que todos os arquivos do tipo css foram compilados por meio dos arquivos do tipo scss.

Neste relatório apenas o arquivo todoScript.js será abordado, pois o script.js serve apenas para a validação de login.

O projeto pode ser acessado através do endereço https://didzao.github.io/paradigmas-prog/

Os dados para login são:

usuário: qualquer valor de 10 caracteres;

senha: 117698.

todoScript.js

No arquivo foram utilizados os conceitos de POO, focando em classes. Abaixo estão listadas cada uma das classes e suas explicações. Todas as funções são precedidas pela palavra *static*, uma palavra reservada em javascript que não permite que o método possa ser acessado diretamente em instâncias da classe, em vez disso, eles são acessados na própria classe.

Classe LocalStorage

Usada para salvar e recuperar as tarefas na memória local do navegador, com isso quando uma tarefa é adicionada, removida ou alterada e o usuário recarregar a página, as informações permanecerão intactas. É formada por três métodos:

addTodoStorage(todoList): adiciona os itens que estão na *todoList* (a lista de afazeres) no storage do navegador.

getStorage(): recupera as informações do storage.

getUserStorage(): recupera o nome do usuário logado.

Classe Todo

Através do constructor a classe é iniciada com os atributos *id*, para identificação da tarefa e *todo*, sendo a tarefa em si . Essa classe não possui métodos.

Classe DisplayTodo

Usada para renderizar a lista de afazeres (*todoList*) na tela. É composta por nove métodos:

displayData(): responsável por renderizar a lista de afazeres na tela.

disableSaveButton(): controla quando o botão de salvar fica habilitado.

clearInput(): limpa o input após a tarefa ser salva e também chama o método *disableSaveButton*.

removeButton(): renderiza o botão de apagar tudo quando há mais de duas tarefas listadas.

removeAllTodo(): remove todas as tarefas listadas na tela, além de limpar o *storage*.

removeTodoList(id): apaga a tarefa em questão tanto da *todoList* quanto do *storage*.

removeTodo(): ao clicar no ícone de lixeira, chama o método *removeTodoList*, além de remover tarefa da tela.

editTodo(): ao clicar no ícone de lápis, é habilitado o modo de edição, com isso o ícone de lápis muda para o de check, que passa a ter função de salvar a alteração. Após clicar nesse ícone, a *todoList* e o *storage* são atualizados com a alteração e o ícone volta para a imagem de lápis.

Além das classes listadas acima há três ***listeners*** que foram usados através do método ***Element.addEventListener()***. São eles:

input.addEventListener: avalia se há valor no input de tarefa, caso exista, a função *disableSaveButton* é chamada com o parâmetro *false* e o botão é habilitado.

form.addEventListener: utilizando *submit* como o tipo de evento, esse listener tem como função chamar a função *saveButton* da classe *DisplayTodo*, para controlar quando o botão de salvar fica ativo ou não. É também iniciado uma constante *todo* tendo como base a classe *Todo*. Na *todoList* é usado o spread, para que todos valores anteriores sejam mantidos quando o novo todo for adicionado. Além disso, os métodos *displayData*, *clearInput* e *removeButton* da classe *DisplayTodo* são chamadas aqui. O método *addTodoStorage* da classe *LocalStorage* também é chamado.

window.addEventListener: utiliza o *DOMContentLoaded* como tipo de evento. Esse *lister* fica responsável por chamar as funções *disableSaveButton*, *displayData*, *removeTodo*, *editTodo*, *removeAllTodo*, *removeButton* da classe *DisplayTodo* assim que o arquivo *todoScript.js* for completamente carregado.

A variável todoList sempre inicia com o valor que está no storage, logo, sempre que o usuário recarregar a página será mostrado o que está armazenado na memória do navegador.

As constantes *userText* e *messageGreetings* são destinadas a formar a mensagem de cumprimentos ao usuário e de criar um novo nó de texto no arquivo *todoPage.html*, respectivamente. E, por fim

userSpan.appendChild(messageGreetings) adiciona a mensagem no nó criado previamente.