Aplicativo de Tarefas com arquitetura MVC em JavaScript

Trabalho de Engenhria de Software

Aluna: Letícia Capitani

MVC é um padrão de arquitetura para organização de código que se baseia nos elementos:

- `model` → Gerencia os dados de uma aplicação;
- `view` → Representação visual do modelo;
- `controller` → Vincula o usuário e o sistema;

O `model` são os dados. Neste aplicativo de tarefas, esses dados serão as tarefas e os métodos que irão adiciona-las.

A `view` é como os dados são exibidos. Nesse aplicativo de tarefas, será o HTML renderizado e CSS.

O `controller` conecta o `model` e a `view`. Ele recebe a entrada do usuário, como clicar ou digitar e lida com retornos de chamada para interação do usuário .

index.html

O foco deste trabalho é inteiramente em JavaScript, uma vez que nele será de fato implementado a arquitetura MVC, porém o HTML abaixo consiste em uma base para a página, assim como, o CSS representa a parte gráfica e não possui foco imediato.

```
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Tarefas</title>
    <link rel="stylesheet" type="text/css" href="style.css"> <!-- conecta com css -->
</head>
<body> <!-- conteúdo visível da página -->
    <h1>Lista de Tarefas</h1>

    <div class="tarefas"> <!-- classe para aplicar estilos css ou selecionar elementos via js -->
        <input type="text" id="nova-tarefa" placeholder="Adicionar uma nova tarefa"> <!-- cria campo de entrada de texto -->
        <button id="adicionar-tarefa">Adicionar</button> <!-- cria botão -->
    </div>
    <ul id="lista-de-tarefas"> <!-- lista não ordenada usada para exibir tarefas do usuário -->
    </ul>

    <script src="script.js"></script> <!-- conecta com js -->
</body>
</html>
```

script.js

Para facilitar o entendimento e implementação do MVC o arquivo JavaScript foi dividido em três classes, `model`, `view` e `controller`. Focaremos primeiramente no `model`, usado apenas para modificar dados.

`class model`

```
const model = {
  tarefas: [], /* vetor vazio usado para armazenar tarefas */
  adicionarTarefa: function(tarefa) { /* criação do metoro adicionarTarefas no objeto model, que aceita um parametro chamado tarefa */
      this.tarefas.push({ texto: tarefa, concluida: false }); /* usa o método push para adicionar um novo objeto ao array tarefas */
  },
};
```

A função `adicionarTarefa` é reponsavel por adicionar novas tarefas no array `tarefas` no objeto `model`. Para teste pode ser adicionado uma tarefa da seguinte forma:

```
model.adicionarTarefa("Lavar a louça") /* checklist virá como falso por padrão */
```

`class view`

```
const view = {
  renderizarListaDeTarefas: function() {
      const listaDeTarefas = document.getElementById("lista-de-tarefas"); /* seleciona o elemento e armazena na variável listaDeTarefas
      listaDeTarefas.innerHTML = ""; /* limpa a lista de tarefas existente antes de renderizar a nova lista */
      model.tarefas.forEach((tarefa, index) => { /* percorrer e retorna cada tarefa no array */
          const li = document.createElement("li"); /* cria um li (item da lista) para cada tarefa */
          /* criação do checkbox */
          const checkbox = document.createElement("input");
          checkbox.type = "checkbox";
          checkbox.checked = tarefa.concluida;
          checkbox.addEventListener("change", () => controller.alternarTarefa(index)); /* altera o status (marcada/desmarcada) */
          li.appendChild(checkbox); /* insere checkbox no item da lista (li) */
          const textoDaTarefa = document.createElement("span"); /* exibe o texto da tarefa */
          textoDaTarefa.textContent = tarefa.texto; /* exibe o conteúdo da tarefa na página */
          if (tarefa.concluida) {
              /* se a tarefa estiver marcada define estilos css específicos*/
              textoDaTarefa.style.textDecoration = "line-through";
              textoDaTarefa.style.color = "#ccc";
          }
          li.appendChild(textoDaTarefa); /* insere o elemento span (texto da tarefa) no item de lista (li) */
          listaDeTarefas.appendChild(li); /* insere o item de lista (li) na lista de tarefas na página */
      });
  },
};
```

No geral, a função `renderizarListaDeTarefas` cria elementos HTML dinamicamente para cada tarefa no objeto `model`. Ela também aplica estilos com base no status da tarefa (marcada ou desmarcada), permitindo que o usuário veja e interaja com o site.

`class controller`

```
const controller = {
  adicionarTarefa: function() { /* chamado quando usuário deseja adicionar uma nova tarefa na lista */
      const inputTarefa = document.getElementById("nova-tarefa"); /* seleciona o elemento e armazena ele na variável inputTarefa */
      const textoTarefa = inputTarefa.value.trim(); /* obtem valor do campo de entrada e armazena em textoTarefa */
      if (textoTarefa !== "") {
          model.adicionarTarefa(textoTarefa); /* chama o método adicionarTarefa do objeto model */
          inputTarefa.value = ""; /* limpa campo de entrada */
          view.renderizarListaDeTarefas(); /* atualiza a exibição da lista de tarefas na página web após adição */
      }
  },
  alternarTarefa: function(index) { /* alterana estado de uma tarefa (marcada/desmarcada) */
      model.tarefas[index].concluida = !model.tarefas[index].concluida; /* marcada = concluida e vice e versa (inverte booleano) */
      view.renderizarListaDeTarefas(); /* atualizar a exibição da lista de tarefas */
  },
};
```

Assim, o objeto `controller` lida com a interação do usuário com a página, sendo para adicionar tarefas á lista e alternar o estado de tarefas concluídas. Quando essas interações ocorrem, o objeto `controller` atualiza o modelo de dados (objeto `model`) e solicita que a exibição (objeto `view`) seja atualizada para refletir as mudanas na interface do usuário.

Ajustando botão de adicionar tarefas

```
/* quando o botão "adicionar-tarefa" é clicado, a função adicionarTarefa do objeto controller será chamada */
document.getElementById("adicionar-tarefa").addEventListener("click", controller.adicionarTarefa);
view.renderizarListaDeTarefas(); /* renderizar a lista de tarefas na página com base nos dados existentes no modelo */
```

Por último, é permitido adicionar tarefas quando o botão `adicionar-tarefa` for clicado, depois disso, renderiza a lista de tarefas existentes na página derante o carregamento inicial da página.

Conclusão

Finalizado o aplicativo de tarefas em JavaScript que demonstra os conceitos da arquitetura model-view-controller, abaixo há o link para o repositório e demonstração final do trabalho.

Trabalho no GitHub


Lista de Tarefas
Adicionar uma nova tarefa
Adicionar
Lavar a louça
Estudar REC
Estudar JavaScript
Limpar a casa
Assistir Hora de Aventura
Fazer Trabalho de SOFT