Como funciona a aplicação – aula conhecendo local storage

Para o main.js é importado a classe FavoritesView e é criado um novo objeto que contenha como parâmetro = ‘#app’

import {FavoritesView} from "./favorites.js"

**new** FavoritesView('#app')

Portanto, vamos abrir o FavoritesView(parâmetro)

export class FavoritesView extends Favorites {

  constructor(*root*) {

*super*(*root*)

*this*.tbody = document.querySelector('table tbody')

*this*.update()

*this*.onadd()

  }

  onadd() {

    const addButton = *this*.root.querySelector('.search button')

    addButton.onclick = () => {

      const {value} = *this*.root.querySelector('.search input')

*this*.add(value)

    }

  }

  update() {

*this*.removeAllTr()

*this*.entries

        .forEach(*user* => {const row = *this*.createRow()

        row.querySelector('.user img').src = `https://github.com/${*user*.login}.png`

        row.querySelector('.user img').alt = `Imagem de ${*user*.name}`

        row.querySelector('.User-name').textContent = *user*.name

        row.querySelector('.followers').textContent = *user*.followers

        row.querySelector('.repo').textContent = *user*.public\_repos

        row.querySelector('.btn-remove').onclick = () => {

          const isOK= confirm('Tem certeza que quer deletar essa linha?')

          if(isOK) {

*this*.delete(*user*)

          }

        }

*this*.tbody.append(row)

      })

  }

  createRow() {

      const tr = document.createElement('tr')

      tr.innerHTML = `<td class="user">

          <img src="https://github.com/kinhoreis2000.png" alt="Imagem do henrique">

          <a href="https://github.com/kinhoreis2000" target="\_blank">

          <p class="User-name" >Luiz Henrique Reis</p>

          </a>

        </td>

        <td class="repo">

          76

        </td>

        <td class="followers">

          120000

        </td>

        <td>

          <button class="btn-remove">remove</button>

        </td>`

      return tr

  }

  removeAllTr() {

*this*.tbody.querySelectorAll('tr')

      .forEach(

        function (*tr*) {

          tr.remove()

        }

        )

  }

}

Perceba que a classe favoritesView se extende pra classe Favorites, ou seja, o que acontece na Favorites também vai existir na FavoritesView.

**PARA A EQ1 – MÉTODO CONSTUTOR**

O método construtor é uma função constructor que tem como objetivo obrigar o usuário a colocar as informações de inicialização do método, neste caso vai ser a div ‘#app’ que criamos no objeto

  constructor(*root*) { //EQ1

*super*(*root*)

*this*.tbody = document.querySelector('table tbody') //linha 3

*this*.update() //linha 4

*this*.onadd()

  }

A primeira linha do método construtor tem a keyword *super*(*root*) que ela é responsável por rodar o método construtor da classe Favorites que está sendo extendida, portanto nós iremos ler o construtor da classe favorites, veja:

 constructor(*root*) {

*this*.root = document.querySelector(root)

*this*.load()

GithubUser.search('kinhoreis2000').then(*user* => console.log(*user*))

    }

Este construtor, no momento, está realizando três coisas.

A primeira é criando a variável this.root como sendo a div ‘#app’ selecionada pela DOM.

A segunda é rodar a função load() com a palvra chave this.

A função this.load() é responsável por criar a base de dados.

load() {

*this*.entries =  JSON.parse(localStorage.getItem('@github-favorites:')) || [] // Essa linha vai transformar o dado em um objeto, ou seja, vai pegar o que tá entre as aspas e transformar no que ele realmente é (pode ser array, objeto, funçao e etc)

    }

Esta função cria uma variável chamada this.entries, ELA VAI PEGAR TUDO QUE TEM COMO CHAVE ‘@github-favorites:’ e transformar em um objeto JSON, colocando tudo no this.entries.

A terceira coisa que o construtor da classe Favorites vai fazer é rodar a classe GithubUser, que neste caso está sendo usado como exemplo com o parâmetro do login ‘kinhoreis200’.

export class GithubUser {

  static search (*username*) { // static é utilizado para usar um endereço de memória fixo, em todo o sistema. Em todos os objetos iniciados nesse método ele vai retornar exatamente o mesmo endereço de memória

    const endpoint = `https://api.github.com/users/${*username*}`

      return fetch(endpoint).then(*data* => *data*.json()).then((*data*) => {

        const {login, name, public\_repos, followers} = *data*

        return {

          login : login,

          name : name,

          public\_repos : public\_repos,

          followers : followers,

        }

      }) // FETCH É BUSCAR ENDPOINT (Ou seja, buscar na url, na internet) e então você vai rodar uma função com parâmetro data e vai retornar toda a data em formato de json. No final das contas essa linha vai me retornar um objeto com os dados do usuário

  }

}

Esta classe irá criar um método estático Search com parâmetro ‘username’. Portanto, vamos criar uma constante chamada ENDPOINT que será o https com o parâmetro username

Essa função nos irá retornar uma promessa com parâmetro do https solicitado, vai transformar todos os dados obtidos em json e, depois vai retornar uma constante que contenha os dados do usuário. Neste momento,

Terminamos, agora, o construtor da classe Favorites, vamos terminar o construtor da classe FavoritesView

A partir da LINHA 3, citada no construtor da classe Favorites view podemos perceber que ela cria uma variável que se chama this.tbody que vai selecionar com a DOM o tbody da tabela HTML.

Logo após, a função this.update() vai rodar, que será descrita pelo próximo item

**PARA A EQ4 – REMOVEALLTR**

    removeAllTr() {

*this*.tbody.querySelectorAll('tr')

      .forEach(

        function (*tr*) {

*tr*.remove()

        }

        )

    }

Essa função vai pegar o tbody que foi criado no construtor e selecionar com a DOM todos os TRs. A partir de então, uma funcionalidade/método chamada forEach vai rodar uma função para cada TR selecionado.

A função irá, portanto, criar um argumento/parâmetro tr para cada tr selecionado pela DOM e irá acionar o parâmetro remove() para cada tr.

**PARA A EQ3 – CREATE ROW**

Esse método será para criar linhas das tabelas

Primeiro, uma constante chamda ‘tr’ é criada e, com a DOM, é criado um elemento HTML de div ‘tr’ pra ela. Após isso, é alterado o HTML de dentro da tr com a propriedade innerHTML.

createRow() {

      const tr = document.createElement('tr')

      tr.innerHTML = `<td class="user">

          <img src="https://github.com/kinhoreis2000.png" alt="Imagem do henrique">

          <a href="https://github.com/kinhoreis2000" target="\_blank">

          <p class="User-name" >Luiz Henrique Reis</p>

          </a>

        </td>

        <td class="repo">

          76

        </td>

        <td class="followers">

          120000

        </td>

        <td>

          <button class="btn-remove">remove</button>

        </td>`

      return tr

    }

O método createRow irá retornar o tr criado, já com o innerHTML inserido

**PARA A EQ2 – UPDATE**

Para inicializar o método, primeiro a função this.removeAllTr() é realizada e todas as linhas que foram criadas estão sendo apagadas.

update() {

*this*.removeAllTr()

*this*.entries.forEach(*user* => {const row = *this*.createRow()

        row.querySelector('.user img').src = `https://github.com/${*user*.login}.png`

        row.querySelector('.user img').alt = `Imagem de ${*user*.name}`

        row.querySelector('.User-name').textContent = *user*.name

        row.querySelector('.followers').textContent = *user*.followers

        row.querySelector('.repo').textContent = *user*.public\_repos

        row.querySelector('.btn-remove').onclick = () => {

          const isOK= confirm('Tem certeza que quer deletar essa linha?')

          if(isOK) {

*this*.delete(*user*)

          }

        }

*this*.tbody.append(row)

      })

    }

Após as linhas terem sido apagadas, a variável this.entries criada no construtor da classe Favorites irá realizar um método para cada entrada de dados.

A função utulizada para cada entrie terá como argumento/parâmetro a palavra *user.* Portanto, *a primeira coisa que a função faz é criar uma constante chamada row.   
A constante row será o tr criado no this.Createrow(), que já consta todo o HTML desejado.*

*A partir daí, é selecionado as entidades que queremos com a DOM por meio do método queryselector, e como cada user corresponde a cada entrie, podemos destacar as o que queremos alterar. Perceba:*

   row.querySelector('.repo').textContent = *user*.public\_repos

Neste caso, é selecionado a linha pela DOM utilizando o querySelector(‘.repo’) e utilizamos a propriedade textContent, igualamos o conteúdo dela para user.publicrepos (user tem cada usuário das entries)

Outra coisa que é feita, neste momento, é selecionar o botão de remover e adicionar um evento quando clicar no botão. Neste caso, é uma funão que vai criar um bolean que, caso seja verdadeiro, vai rodar o método this.delete(user)

row.querySelector('.btn-remove').onclick = () => {

          const isOK= confirm('Tem certeza que quer deletar essa linha?')

          if(isOK) {

*this*.delete(*user*)

          }

        }

Este método está na classe Favorites, veja

delete(*user*) {

      //higher order functions (map, filter, find, reduce)

      const filteredEntries = *this*.entries

      .filter(*entry* => *entry*.login !== *user*.login)

      console.log(filteredEntries)

*this*.entries = filteredEntries

*this*.update()

    }

O método delete irá, em suma, criar uma constante chamada filteredEntries e colocar o this.entries (todas as entradas de dados) nele.   
Como o FilteredEntries é um array, a funcionalidade filter de array irá rodar uma função com o parâmetro entry. A Entry será todas as pessoas do Array;.

Essa função, caso retorne verdadeiro, ele colocará os dados no array de volta. Caso retorne falso, ele irá excluir aquele dado do array   
  
Para retornar um verdadeiro ou falso, é comparado a entry.login (que é todos as pessoas que estão no array) com o user.login(que é o user que colocamos um evento ao selecionar o delete)

Após isso, é colocado no this.entries o filteredEntries, dessa forma, só os dados solicitados irão aparecer

Depois roda a função this.update() que está com o this.entries

**PARA A EQ5 – ONADD**

Essa função é responsável por adicionar o evento de click no botão de adicionar github

  onadd() {

    const addButton = *this*.root.querySelector('.search button')

    addButton.onclick = () => {

      const {value} = *this*.root.querySelector('.search input')

*this*.add(value)

    }

  }

Primeiro é criado uma constante chamada addButton que será o butão

Após isso, é colocado um evento de click que rodará uma função que criará uma constante chamada value, que será o valor do input do botão

Após isso, um método chamada add(parametro) de parâmetro value será ativada na classe Favorites

Essa função é responsável por criar uma constante que se chama user, ela irá aguardar o resultado da Search da classe GithubUser

async add(*username*) {

      const user = await GithubUser.search(*username*)

      console.log(user)

    }

O user agora é os dados do usuário pesquisado do github.

O AWAIT é utilizado para mandar o JS aguardar aquela linha ser lida para então ir pra próxima. Para isso poder acontecer é necessário colocar o ASYNC no inicio do método também;