Notas Fase2

Checklist para quem não esteve na última aula presencial

Geral

1. Em termos de novas operações, a fase 2 comtempla apenas 2 novas operações de API: Update e Delete de Rentals
2. Nesta fase, deverão reutilizar o report da Fase 1 acrescentando o necessário relativo a esta fase
3. A fase 2 poderá ser entregue até 26 Abril
4. A Fase 2 consiste na criação de uma SPA com as seguintes regras: Uso de HTML, Javascript e CSS "Vanilla", ou seja, **sem recorrer** a nenhuma Framework (Angular, React, ...) ou bibliotecas javascript (jquery ou outro)

Navegação

1. A aplicação deverá seguir a navegação descrita no enunciado https://github.com/isel-leic-ls/2425-2-common/wiki/Phase-2#single-page-application

A SPA

2. Deverão clonar ou copia o código do Common relativo á SPA
3. O código HTML e Javascript deverá estar na pasta "static-content"
4. Deverão alterar o "Server" para incluir referencia ao conteúdo estático: "singlePageApp".

```
routes(
    "students" bind studentRoutes,
    "date" bind GET to ::getDate,
    singlePageApp(ResourceLoader.Directory("static-content")),
)
```

5. Lançando o url no browser http://localhost:9000/ aparecerá a aplicação de exemplo

Estrutura da SPA

1. Para referencia, deverão considerar a aplicação e conteúdo da pasta "sparouter" pois assemelha-se mais ao nosso trabalho.

2. Estrutura:

Index.html	`<div id="mainContent"> </div>`	É no mainContent que as vistas serão "pintadas"
Index.js	`window.addEventListener('load', loadHandler)` `window.addEventListener('hashchange', hashChangeHandler)`	Criar os listener para os eventos. O Evento hashChange é disparado sempre que o url a seguir ao "#" sofre alteração
Index.js	`function loadHandler()`	Carregar as rotas (parte do url após #) os os handlers respectivos que irão chamar as APIs e "pintar" o html da vista
Router.js		Módulo para gestão das rotas e obtenção do Handler dado um Url (deverá ter a lógica para identificar qual a rota faz match. Ter em conta que alguns dos urls terão path parameters)

Handlers.js	`unction getStudents(mainContent){`
	`  fetch(API_BASE_URL + "students")`
	`  .then(res => res.json())`
	`  .then(students => {`
	`  const div = document.createElement("div")`
	
	`  const h1 = document.createElement("h1")`
	`  const text = document.createTextNode("Students")`
	`  h1.appendChild(text)`
	`  div.appendChild(h1)`
	
	`  students.forEach(s => {`
	`  const p = document.createElement("p")`
	`  const a = document.createElement("a")`
	`  const aText = document.createTextNode("Link ISEL students/" + s.number);`
	`  a.appendChild(aText)`
	`  a.href="#students/" + s.number`
	`  p.appendChild(a)`
		Tem os handlers para cada rota. Cada Função deverá ter a lógica de chamada á API (função javscritpt fetch) e a lógica de "pintura" do HTML
		
		A função que gera o HTML deverá obter ou receber referencia para a TAG div "mainContent" (document.getElementById())

Estrutura de pastas e ficheiros

1. A estrutura é decisão do grupo, mas recomenda-se a segregação dos ficheiros por domino e/ou funcionalidade. Por exemplo, ter um HanderHome.js, HandlerRentals.js, etc

Construção do HTML das vistas

1. O HTML deverá ser produzido recorrendo ás funções do DOM (e.g. document.createElement(), ...).
2. No entanto, estas funções nativas não deverão ser usadas diretamente pelas vistas.

3. O Grupo Deverá criar uma “library” de DSL (e.g. dsl.js) que contenha funções mais genéricas. Exemplo extraído do common:
   Apenas a implementação destas funções deverá usar a API do DOM.

```
ul(
    li("Name : " + student.name),
    li("Number : " + student.name)
)
```

Ou

```
createElement("ul",
    createElement("li", "Name : " + student.name),
    createElement("li", "Number : " +
student.number)
)
```