Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação na Internet

Teste Global de Época Normal, Verão de 2018/2019

GRUPO 1

1. [2] Considere o código seguinte, com a iniciação de uma aplicação Node.js, utilizando os módulos **`express`** e **`passport`**.

```
const LOGIN_URL = "/login";
const express = require('express')
const passport = require('passport')
const validateAuthenticationMw = require('validate-auth')(LOGIN_URL)
const app = express()
// passport initialization on app

...
```

   a. [1] Implemente o módulo **`validate-auth`**, de modo a que na variável **`validateAuthenticationMw`** fique um *middleware* Express que apenas deixa prosseguir pedidos autenticados. Se o pedido não for de um utilizador autenticado, é retornada uma resposta HTTP redirecionando o cliente para o URI recebido como argumento na função exportada pelo módulo.

   b. [1] Registe o *middleware* **`validateAuthenticationMw`** na aplicação `Express`, de modo a que apenas utilizadores autenticados possam aceder a recursos presentes na *path* **`/private`** ou descendentes, independentemente do tipo de pedido HTTP.

2. [3] Para a aplicação B4 desenvolvida durante as aulas, foram definidos os seguintes endpoints para manipulação dos grupos de livros (*bundles*). Para cada *endpoint* comente a decisão tomada, justificando se a considera correta e, em caso negativo, apresente uma proposta alternativa.
   a. Obter os detalhes de um grupo - GET /bundles/getDetails/:bundleId
   b. Obter todos os grupos - GET /bundles/getAll
   c. Criar um novo grupo - GET /bundles/create/:bundleId
   d. Remover um grupo - GET /bundles/delete/:bundleId
   e. Adicionar um livro a um grupo - GET /bundles/addBook/:bundleId/:bookId
   f. Remover um livro a um grupo - GET /bundles/deleteBook/:bundleId/:bookId

3. [2,5] A função seguinte usa a API `fetch()` para obter os títulos de todos os livros pertencentes a todos os grupos de livros (bundles) existentes na base de dados ElasticSearch da aplicação B4, desenvolvida durante as aulas. Reescreva a função de modo a ter o mesmo comportamento, mas sem usar a keyword `await`.

```
async function getAllBundlesBookTitles() {
 let response = await fetch('http://localhost:9200/b4/bundle/_search')
 let bundles = await response.json();
 const bookUrlBase = 'http://localhost:9200/books/book/'
 const requests = bundles
                    .flatMap(bundle => bundle.books)
                    .map(b => fetch(bookUrlBase + b.id))
 const responses = await Promise.all(requests);
 const books = await Promise.all(responses.map(async resp => await resp.json()))
 const booksTitles = books.map(b => b._source.title)
 return booksTitles;
}
```

4. [2,5] Altere o comportamento da função fetch() presente no objeto global de um *web browser*. Para cada Uri com que a função alterada é chamada, é invocada a função original caso o conteúdo da resposta não tenha sido guardado em cache anteriormente. O conteúdo de uma resposta só é guardada em *cache* se, na resposta, o *header* `Cache-Control`, tiver os valores **"public"** ou **"private"**. Seguem-se exemplos de utilização da função fetch() após a alteração do comportamento da função original:

```
1. fetch('http://server/path1')
Pedido realizado e a resposta contém nos headers:
Cache-Control: private
Conteúdo da resposta é guardado em cache e a Promise é resolvida com a
resposta obtida

2. fetch('http://server/path1')
Não é realizado qualquer pedido e a Promise é resolvida com o valor guardado
em cache em 1

3. fetch('http://server/path2')
Pedido realizado e a resposta contém nos headers:
Cache-Control: no-cache
Conteúdo da resposta não é guardado em cache e a Promise é resolvida com a
resposta obtida

4. fetch('http://server/path3')
Pedido realizado e a resposta não contém o header Cache-Control.
Conteúdo da resposta não é guardado em cache e a Promise é resolvida com a
resposta obtida.
```

NOTA: Na resolução deste exercício não deve ser usada qualquer variável global.

GRUPO 2

5. [8] Pretende-se implementar na aplicação YAMA desenvolvida no trabalho prático, a funcionalidade de *auto-complete* na pesquisa de artistas. Esta funcionalidade apresenta uma lista de sugestões ao utilizador, à medida que ele escreve o nome do artista na caixa de texto de pesquisa.

   a. [2] Especifique o *endpoint* que está disponível na componente servidora da aplicação, para suportar esta funcionalidade. Note que não se exige que o implemente; defina apenas o URL e conteúdo da resposta. O *endpoint* está acessível através do método HTTP GET e produz uma resposta em JSON. Para definir este *endpoint*, considere os requisitos que estão implícitos na alínea que se segue.

   b. [3] Implemente a função **`getArtistsNames(nameStart, resultsCount)`** que, no *browser*, realiza um pedido ao *endpoint* da alínea anterior para obter a lista das artistas cujo nome começa por **`nameStart`**. A lista tem dimensão máxima especificada em **`resultsCount`**. A função retorna uma Promise, que quando resolvida produz um objeto com a conversão do JSON obtido na resposta.

   c. [3] Considere o seguinte excerto da página HTML de pesquisa de artistas. Acrescente o que considerar necessário para que, à medida que o utilizador escreve texto, o conteúdo da caixa **`suggestions`** seja actualizado com as sugestões obtidas usando a função da alínea anterior. Garanta que essa caixa só está visível se **`artists`** tiver pelo menos três caracteres e se as sugestões foram obtidas com sucesso. Na implementação considere que já estão definidas as regras CSS necessárias, **`.hiddenAutoComplete { ... }`** e **`.visibleAutoComplete { ... }`**.

```
<input type="text" id="artists" />
<div class="hiddenAutoComplete" id="suggestions"></div>
```

6. [2] Sempre que programaticamente, através da propriedade `window.location`, é alterado o URI, o *browser* realiza um novo pedido GET para esse novo URI? Justifique.

Duração: 2 horas
ISEL, 14 de Junho de 2019