

Resoluções de exames de DAW

Época Normal, Semestre de Verão 20/21

1.1

No protocolo HTTP, o header **Content-Location** tem semântica definida:

- i. Apenas nas mensagens de pedido.
- ii. Apenas nas mensagens de resposta.
- iii. Nas mensagens de pedido e de resposta.
- iv. Nas mensagens de pedido e de resposta com um método idempotente

ii

No protocolo HTTP, o header **Content-Location** tem semântica definida **apenas nas mensagens de resposta** ¹. O header **Content-Location** indica uma localização alternativa para os dados retornados na resposta ¹. O uso mais comum do header **Content-Location** é indicar o URL de um recurso transmitido como resultado da negociação de conteúdo ². O header **Content-Location** é diferente do header **Location**, que indica a URL de um redirecionamento ³.

Source: Conversation with Bing, 1/7/2024

(1) Content-Location - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Location>.

(2) Location - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/web/http/headers/location>.

(3) Content-Location - HTTP | MDN. <http://devdoc.net/web/developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Location.html>.

(4) undefined. <https://example.com/documents/foo>.

1.2

A realização de um pedido de método GET para <https://example.com/projects/123/delete>, deve ser interpretado por um intermediário como sendo:

- i. Equivalente ao pedido de método DELETE para <https://example.com/projects/123>.
- ii. Um pedido idempotente mas não safe.
- iii. Um pedido idempotente e safe.
- iv. Nenhuma das anteriores.

iii

Um pedido de método GET para <https://example.com/projects/123/delete> deve ser interpretado por um intermediário como sendo um pedido idempotente e safe.

O método GET é usado para solicitar uma representação de um recurso específico, e não para alterar o estado do recurso. O método DELETE, por outro lado, é usado para excluir um recurso específico ¹. Portanto, um pedido de método GET para <https://example.com/projects/123/delete> não é equivalente a um pedido de método DELETE para <https://example.com/projects/123>.

1.3

Na plataforma Spring MVC, por omissão, o construtor de uma classe anotada com `@RestController` é chamado:

- i. Uma vez por cada pedido HTTP, independentemente do handler que processa o pedido.
- ii. Uma vez por cada pedido HTTP processado por um handler presente nessa classe.
- iii. Uma vez por cada utilizador distinto.
- iv. Uma vez por cada instância da aplicação.

iv

O construtor de uma classe anotada com `@RestController` na plataforma Spring MVC é chamado uma vez por cada instância da aplicação

1.4

Assumindo o processo de construção de aplicações para execução na plataforma browser usando na unidade curricular, qual o resultado da avaliação da seguinte expressão num módulo da aplicação:

```
const m = require('utils')
```

- *i. Erro de execução, porque a função `require` não está definida.
- ii. Pedido HTTP de método GET com caminho `/utils`.
- iii. Pedido HTTP de método GET com caminho `/utils.js`.
- iv. Referência para um objecto, se o módulo `utils` estiver presente em `node modules`.

iv

A expressão `const m = require('utils')` é uma declaração de variável que usa a função `require()` para importar um módulo chamado `utils`. Se o módulo `utils` estiver presente em `node_modules`, a expressão retornará uma referência para o objeto exportado pelo módulo. Caso contrário, a expressão resultará em um erro de tempo de execução

1.5

No contexto de uma single page application, a avaliação da seguinte expressão resulta em:

```
window.location.pathname = '/projects'
```

- i. Na realização de um pedido HTTP de método GET para o caminho `/index.html`.
- ii. Na realização de um pedido HTTP de método HEAD para o caminho `/index.html`.
- iii. Na realização de um pedido HTTP de método GET para o caminho `/projects`.
- iv. Na realização de um pedido HTTP de método GET para o caminho `/projects.html`.

iii

Explicação do professor:

- usar a API de `location` afetando o `pathname` realiza sempre um novo pedido
- já na API `history` (para navegação no browser), não efetua pedido

Explicação do Bing copilot:

Na avaliação da expressão `window.location.pathname = '/projects'` em uma single page application, a realização de um pedido HTTP de método GET para o caminho `'/projects'` é o resultado correto.

A propriedade `window.location.pathname` retorna o caminho da URL da página atual, sem o nome do domínio ou o protocolo. A expressão `window.location.pathname = '/projects'` define o caminho da URL como `/projects`

1.6

O resultado da avaliação da expressão JSX

é: i. Um elemento HTML, do mesmo tipo do obtido na avaliação da expressão `document.createElement('div')`. ii. Um elemento HTML, do mesmo tipo do obtido na avaliação da expressão `document.createElement('Div')`. iii. O resultado da avaliação da expressão `Div({})`. iv. Nenhuma das anteriores

i

A expressão

é uma expressão JSX que representa um elemento React. Quando essa expressão é avaliada, ela é transformada em uma chamada para a função `React.createElement()`. A expressão `document.createElement('div')` também cria um elemento HTML do tipo `div`. Portanto, o resultado da avaliação da expressão JSX

é um elemento HTML do mesmo tipo do obtido na avaliação da expressão `document.createElement('div')`.

2

No formato Siren, quais a diferenças entre links e ações?

No formato Siren, links e ações são duas estruturas diferentes que descrevem um hyperlink e seus metadados.

Links são usados para navegação do cliente e para comunicar informações sobre entidades. Eles podem ser usados para representar relacionamentos entre entidades, como uma entidade que contém outra entidade.

Ações são usadas para executar transições de estado em uma entidade. Elas descrevem um comportamento que pode ser executado em uma entidade, como adicionar um novo item a uma lista

3

No contexto do protocolo HTTP, descreva qual a relação entre negociação de conteúdos e caching. Nomeadamente, indique as consequências na organização do sistema de cache decorrentes da existência de negociação de conteúdos, bem como a informação extra que é necessário incluir nas mensagens HTTP.

A negociação de conteúdos é um mecanismo usado pelo protocolo HTTP para servir diferentes representações de um recurso para a mesma URI, ajudando o agente do usuário a especificar qual representação é mais adequada para o utilizador. A negociação de conteúdos pode ser realizada de duas maneiras: negociação impulsionada pelo servidor e negociação impulsionada pelo agente do utilizador (browser).

A negociação impulsionada pelo servidor é quando o servidor escolhe a representação mais adequada com base nas informações fornecidas pelo agente do utilizador (browser). A negociação impulsionada pelo agente do utilizador é quando o agente do utilizador escolhe a representação mais adequada com base nas informações fornecidas pelo servidor.

O sistema de cache deve ser organizado de forma a armazenar as diferentes representações de um recurso. O cache deve incluir informações adicionais nas mensagens HTTP, como o cabeçalho `Vary`, que indica quais cabeçalhos de solicitação foram usados na negociação de conteúdo. Isso permite que o cache trabalhe de forma otimizada

4

Tendo em consideração o modelo de construção de aplicações para a plataforma browser usado na unidade curricular, indique qual o propósito e forma da utilização do sistema NPM

O sistema NPM (Node Package Manager) é usado para gerenciar dependências de pacotes em projetos JavaScript. Permite que os desenvolvedores instalem e gerenciem pacotes de terceiros, bem como publiquem seus próprios pacotes.

No contexto de construção de aplicações browser, o NPM é usado para gerenciar as dependências do lado do cliente, como bibliotecas JavaScript, CSS e imagens. O NPM permite que os desenvolvedores instalem e gerenciem essas dependências de forma fácil e eficiente.

5

Realize um ou mais componentes para a plataforma Spring MVC de forma a expor um recurso no caminho `/handlers`. Um pedido de método GET a este recurso deve retornar um objecto JSON. Cada campo deste objecto representa um handler usado no processamento de pelo menos um pedido, sendo o valor do campo um objecto com:

- O número de vezes que o handler foi utilizado.
- O tempo médio de execução dos pedidos a esse handler.

Assuma que todos os handlers são do tipo `HandlerMethod`. Use o método `getShortLogMessage` para obter uma representação textual do handler. Valorizam-se soluções em que o cálculo do tempo de processamento inclua não só o tempo de execução do handler mas também o da maioria dos intermediários envolvidos (e.g. filtros e interceptores).

Para expor um recurso no caminho `/handlers` em uma aplicação Spring MVC, você pode criar um controlador com um método que retorna um objeto JSON com informações sobre os handlers usados no processamento de pelo menos um pedido. Cada campo do objeto representa um handler e contém o número de vezes que o handler foi utilizado e o tempo médio de execução dos pedidos a esse handler. Para obter uma representação textual do handler, você pode usar o método `getShortLogMessage`. Para calcular o tempo de processamento, você pode incluir o tempo de execução dos intermediários envolvidos, como filtros e interceptores.

```
@RestController
```

```

@RequestMapping("/handlers")
class HandlerInfoController @Autowired constructor(private val handlerMapping:
RequestMappingHandlerMapping) {

    data class HandlerStats(var usageCount: Int = 0)

    @GetMapping
    fun getHandlerInfo(): Map<String, Map<String, Any>> {
        val result = mutableMapOf<String, Map<String, Any>>()
        val handlerStatsMap = mutableMapOf<HandlerMethod, HandlerStats>()

        // Itera sobre todos os handlers mapeados
        for ((_, handlerMethod) in handlerMapping.handlerMethods) {
            // Verifica se o handler já está no mapa de estatísticas
            if (!handlerStatsMap.containsKey(handlerMethod)) {
                handlerStatsMap[handlerMethod] = HandlerStats()
            }

            // Atualiza as estatísticas do handler
            val handlerStats = handlerStatsMap[handlerMethod]!!
            handlerStats.incrementUsageCount()
            // Aqui, você pode adicionar lógica para calcular o tempo de execução
            incluindo interceptors, etc.
        }

        // Converte as estatísticas para o formato desejado no JSON
        for ((handlerMethod, handlerStats) in handlerStatsMap) {
            val handlerInfo = mapOf(
                "usageCount" to handlerStats.usageCount
                // Adicione outros campos conforme necessário
            )

            result[handlerMethod.method.toGenericString()] = handlerInfo
        }

        return result
    }
}

```

6

Realize um componente React que recebe um URI e que apresenta uma caixa de texto (textarea) e um botão. A caixa de texto deve apresentar o conteúdo do body presente na resposta a um pedido HTTP de método GET ao URI recebido, independentemente do status code da resposta. Caso o pedido resulte numa exceção, a caixa de texto deve apresentar o texto associado a essa exceção.

Um clique no botão deve desencadear um novo pedido HTTP ao URI definido e consequente apresentação da resposta. Enquanto um pedido estiver em curso, o botão deve permanecer inactivo (disabled).

O componente deve ser sensível a mudanças no URI definido. Caso o novo URI seja diferente do URI usado no último pedido (terminado ou em curso), deve ser desencadeado um pedido para o novo valor do URI e cancelado eventuais pedidos em curso.

```

import React, { useState, useEffect } from 'react';

function RequestComponent() {
  const [uri, setUri] = useState('');
  const [response, setResponse] = useState('');
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    async function fetchData() {
      try {
        setLoading(true);
        const res = await fetch(uri);
        const text = await res.text();
        setResponse(text);
      } catch (error) {
        setResponse(error.toString());
      } finally {
        setLoading(false);
      }
    }

    if (uri) {
      fetchData();
    }
  }, [uri]);

  function handleSubmit(event) {
    event.preventDefault();
    setResponse('');
    setLoading(true);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        URI:
        <input type="text" value={uri} onChange={(event) =>
setUri(event.target.value)} />
      </label>
      <button type="submit" disabled={loading}>
        {loading ? 'Loading...' : 'Fetch'}
      </button>
      <br />
      <textarea value={response} readOnly />
    </form>
  );
}

```

Este componente usa os hooks `useState` e `useEffect` para gerenciar o estado do URI, da resposta e do indicador de carregamento. Quando o usuário envia o formulário, o componente faz uma solicitação HTTP GET para o URI especificado usando a API `fetch` do navegador. Se a solicitação for bem-sucedida, o componente exibe o conteúdo do corpo da resposta na caixa de texto. Se a solicitação falhar, o componente exibe a mensagem de erro na caixa de texto. O botão de envio é desativado enquanto a solicitação está em andamento. O componente também é sensível a mudanças no URI definido, fazendo uma nova solicitação HTTP sempre que o URI muda.

Época de Recurso, Semestre de Verão, 20/21

1.1

No protocolo HTTP, o header Cache-Control tem semântica definida:

- i. Apenas nas mensagens de pedido.
- ii. Apenas nas mensagens de resposta.
- iii. Nas mensagens de pedido e de resposta.
- iv. Nenhuma das anteriores.

iii

O header Cache-Control no protocolo HTTP tem semântica definida em mensagens de pedido e de resposta 1. Ele é usado para especificar diretivas que controlam o armazenamento em cache de uma resposta HTTP ou a solicitação de uma nova cópia da resposta 1. As diretivas podem ser usadas para controlar o armazenamento em cache em navegadores e caches compartilhados, como proxies e CDNs

1.2

****A realização de um pedido de método POST para <https://example.com/projects/123/delete>, deve ser interpretado por um intermediário como sendo:**

- i. Equivalente ao pedido de método DELETE para <https://example.com/projects/123>.
- ii. Um pedido idempotente mas não safe.
- iii. Um pedido idempotente e safe.
- iv. Nenhuma das anteriores.

iv

POST não é safe nem idempotente

1.3

Na plataforma Spring MVC e assumindo a configuração por omissão:

- i. Os handlers presentes numa classe controller podem ser chamados em concorrência sobre a mesma instância, por threads distintas.
- ii. Os handlers presentes numa classe controller nunca são chamados em concorrência sobre a mesma instância, porque apenas existe uma thread a processar pedidos.
- iii. Os handlers presentes numa classe controller nunca são chamados em concorrência sobre a mesma instância, porque existe um lock que protege esses acessos.
- iv. Nenhuma das anteriores.

i

Os handlers presentes numa classe controller podem ser chamados em concorrência sobre a mesma instância, por threads distintas.

1.4

****No contexto da framework React, a utilização de [] como segundo argumento da função useEffect, significa que:**

- i. O efeito vai ser chamado uma vez durante o tempo de vida da instância do componente.
- ii. O efeito vai ser chamado uma vez durante o tempo de vida da aplicação.

- iii. O efeito vai ser chamado sempre que a função que define o componente for chamada.
- iv. Nenhuma das anteriores.

i

No contexto da framework React, a utilização de [] como segundo argumento da função useEffect significa que o efeito vai ser chamado uma vez durante o tempo de vida da instância do componente .

O segundo argumento da função useEffect é uma matriz de dependências que especifica quando o efeito deve ser executado . Se a matriz estiver vazia, o efeito será executado apenas uma vez, após a montagem do componente . Isso é útil para efeitos que precisam ser executados apenas uma vez, como a inicialização de um estado ou a configuração de um evento.

1.5

****No contexto de uma single page application, a avaliação da seguinte expressão resulta em:**

```
history.pushState({}, '', '/projects/123')
```

- i. Na realização de um pedido HTTP de método GET para o caminho '/index.html'.**
- ii. Na realização de um pedido HTTP de método GET para o caminho '/projects/123/index.html'.**
- iii. Na realização de um pedido HTTP de método GET para o caminho '/projects/123'.**
- iv. Nenhuma das anteriores.**

iv

Adiciona a página mas não faz nenhum pedido

1.6

O resultado da avaliação da expressão JSX

é: i. Um elemento HTML, do mesmo tipo do obtido na avaliação da expressão document.createElement('div'). ii. O resultado da avaliação da expressão React.createElement('div', null). iii. O resultado da avaliação da expressão React.createElement(div, null). iv. Nenhuma das anteriores.

iv

é diferente de

2

No contexto da plataforma Spring MVC, indique duas formas distintas para a definição de beans.

@Component:

A anotação @Component é usada para varredura automática e configuração automática de beans. Ela permite que o Spring detecte e configure automaticamente os beans com base na varredura do classpath.

Cada classe anotada com @Component (ou suas variantes, como @Service e @Repository) é mapeada implicitamente para um bean (ou seja, um bean por classe).

Essa abordagem é conveniente e eficaz para a maioria dos casos, especialmente quando você

possui o código-fonte das classes.

Exemplo:

```
import org.springframework.stereotype.Component;

@Component
public class MeuBean {
    // Implementação do bean
}
```

@Bean:

A anotação @Bean é usada para declarar explicitamente um bean no Spring. Ela permite que você configure e crie beans de acordo com suas necessidades específicas.

A anotação @Bean é aplicada a métodos, não a classes. Esses métodos retornam objetos que o Spring registra como beans no contexto da aplicação.

Você pode usar @Bean em uma classe anotada com @Configuration ou @Component.

Exemplo:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
    @Bean
    public MeuBean meuBean() {
        return new MeuBean();
    }
}
```

Diferença:

@Component é usado para componentes detectados automaticamente, enquanto @Bean é usado para declarar explicitamente um bean.

@Component é mais simples e automático, enquanto @Bean oferece maior controle sobre a configuração do bean.

Use @Component quando você possui o código-fonte da classe e deseja que o Spring cuide da configuração. Use @Bean quando você precisa configurar manualmente um bean ou quando trabalha com bibliotecas de terceiros sem acesso ao código-fonte.

3

No processo de desenvolvimento de aplicações para execução em browser usado na unidade curricular, qual a necessidade da existência de um passo de construção? Ou seja, porque é que os ficheiros fonte não são entregues directamente para execução no browser?

Browser só entende javascript (vanilla) e não TypeScript e React

4

**** Indique o que é necessário realizar para que uma aplicação single page application suporte deep-linking****

O uso de Siren pode ser uma solução eficaz para representar entidades e permitir deep-linking em APIs da Web

5

Realize um ou mais componentes para a plataforma Spring MVC de forma a expor um recurso no caminho /anonymous. Um pedido de método GET a este recurso deve retornar a lista contendo a contabilização de todos os acessos anônimos aos recursos da API. Cada elemento da lista é composto pelo URI do recurso e pelo número de acessos anônimos a esse recurso

Primeiro, um controlador que mapeie o caminho /anonymous e mantenha um registro dos acessos anônimos:

```
import org.springframework.stereotype.Controller
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.ResponseBody

@Controller
class AnonymousAccessController {

    // Mapa para registrar os acessos anônimos (URI -> Contagem)
    private val anonymousAccessMap = mutableMapOf<String, Int>()

    @GetMapping("/anonymous")
    @ResponseBody
    fun getAnonymousAccessStats(): Map<String, Int> {
        // Retorna o mapa de contagem de acessos anônimos
        return anonymousAccessMap
    }

    // Método para registrar um novo acesso anônimo
    fun registerAnonymousAccess(uri: String) {
        anonymousAccessMap[uri] = anonymousAccessMap.getOrDefault(uri, 0) + 1
    }
}
```

Agora, sempre que houver um acesso anônimo a um recurso, chamar o método registerAnonymousAccess para registrar o URI desse recurso:

```
// Exemplo de uso (em algum outro lugar do código)
@Autowired
private lateinit var anonymousAccessController: AnonymousAccessController

// Quando ocorrer um acesso anônimo a um recurso, registre-o
val uriAcessada = "/algum-recurso" // Substitua pelo URI real
anonymousAccessController.registerAnonymousAccess(uriAcessada)
```

ou sem @Autowired injetado no construtor:

```
@Controller
class MyController(private val anonymousAccessController:
AnonymousAccessController) {
    // Quando ocorrer um acesso anônimo a um recurso, registre-o
    val uriAcessada = "/algum-recurso" // Substitua pelo URI real
    anonymousAccessController.registerAnonymousAccess(uriAcessada)
}
```

O endpoint /anonymous retornará o mapa com a contagem de acessos anônimos para cada URI. Por exemplo:

```
{
  "/algum-recurso": 5,
  "/outro-recurso": 3,
  "/mais-um-recurso": 10
}
```

6

Realize um componente para a framework React que implementa um cronômetro com suporte para contagem de tempos parciais. O componente apresenta dois botões Start e Lap, bem como a lista de números com as contagens parciais.

Inicialmente o botão Start está ativo e o botão Lap está inativo. Um clique no botão Start reinicia o cronômetro, o que resulta na ativação do botão Lap e na remoção de todos os números da lista. Neste estado, um clique no botão Lap acrescenta à lista o valor em segundos entre o último clique em Lap e o último clique em Start

Se o botão Start receber um clique enquanto Lap está ativo, então Lap fica novamente inativo e o componente volta ao estado inicial. Note que a avaliação de Date.now() retorna o número de milissegundos desde 1 de janeiro de 1970.

```
import React, { useState, useEffect } from 'react';

const Stopwatch = () => {
  const [startTime, setStartTime] = useState(null);
  const [lapTime, setLapTime] = useState(null);
  const [laps, setLaps] = useState([]);
  const [isRunning, setIsRunning] = useState(false);

  const handleStartClick = () => {
    if (!isRunning) {
      setStartTime(Date.now());
      setLapTime(Date.now());
      setIsRunning(true);
      setLaps([]);
    } else {
      setIsRunning(false);
      setLapTime(null);
    }
  };
};
```

```

const handleLapClick = () => {
  if (isRunning) {
    const currentTime = Date.now();
    const lapDuration = Math.floor((currentTime - lapTime) / 1000); // Convert
to seconds
    setLaps([...laps, lapDuration]);
    setLapTime(currentTime);
  }
};

useEffect(() => {
  let interval;
  if (isRunning) {
    interval = setInterval(() => {
      setLapTime(Date.now());
    }, 1000); // Update every second
  } else {
    clearInterval(interval);
  }
  return () => clearInterval(interval);
}, [isRunning]);

return (
  <div>
    <h1>Stopwatch</h1>
    <p>Elapsed Time: {startTime ? Math.floor((Date.now() - startTime) / 1000) :
0} seconds</p>
    <button onClick={handleStartClick}>{isRunning ? 'Stop' : 'Start'}</button>
    <button onClick={handleLapClick} disabled={!isRunning}>Lap</button>
    <ul>
      {laps.map((lap, index) => (
        <li key={index}>Lap {index + 1}: {lap} seconds</li>
      ))}
    </ul>
  </div>
);
};

export default Stopwatch;

```

Época Especial, Semestre de Verão, 20/21

1.1

No protocolo HTTP, o header Authorization tem semântica definida:

- i. Apenas nas mensagens de pedido.
- ii. Apenas nas mensagens de resposta.
- iii. Nas mensagens de pedido e de resposta.
- iv. Nenhuma das anteriores.

O header Authorization no protocolo HTTP é usado para autenticação e autorização. Ele pode aparecer tanto nas mensagens de pedido (requests) quanto nas mensagens de resposta (responses). Portanto, a opção iii é a correta, pois abrange ambos os casos. Lembre-se de que o header Authorization é usado para incluir informações de autenticação (como tokens de acesso, credenciais de usuário etc.) nas requisições e respostas HTTP. Ele é essencial para proteger recursos e garantir que apenas usuários autorizados tenham acesso a determinadas partes de uma aplicação ou serviço. O exemplo de uma resposta poderia ser o envio de novo token:

```
HTTP/1.1 200 OK
Content-Type: application/json
Authorization: Bearer newAccessToken...
{
  "message": "Resource accessed successfully."
}
```

1.2

A realização de um pedido de método GET para <https://example.com/projects/123/delete>, deve ser interpretado por um intermediário como sendo:

- i. Equivalente ao pedido de método DELETE para <https://example.com/projects/123>.
- ii. Um pedido idempotente mas não safe.
- iii. Um pedido idempotente e safe.
- iv. Nenhuma das anteriores.

iii

GET é safe e idempotente

1.3

Segundo o RFC 8288 (Web Linking), um link relation type é

- i. O identificador do recurso alvo do link.
- ii. O identificador do recurso que define o contexto do link.
- iii. O identificador que expressa a semântica da ligação entre dois recursos.
- iv. Nenhuma das anteriores.

iii

De acordo com o RFC 8288 (Web Linking), um link relation type (tipo de relação de link) é uma string que expressa a semântica da relação entre dois recursos.

1.4

No contexto da framework React, a utilização de undefined como segundo argumento da função useEffect, significa que:

- i. O efeito vai ser chamado uma vez durante o tempo de vida da instância do componente.
- ii. O efeito vai ser chamado uma vez durante o tempo de vida da aplicação.
- iii. O efeito vai ser chamado sempre que a função que define o componente for chamada.
- iv. Nenhuma das anteriores.

iv

Quando o segundo argumento é undefined, o efeito não será chamado automaticamente em nenhum momento específico. Isso significa que o efeito não será executado durante o tempo de vida da instância do componente, nem uma única vez durante o tempo de vida da aplicação, nem sempre que a função que define o componente for chamada. O efeito só será executado manualmente quando você o invocar explicitamente.

Para que o efeito seja chamado apenas uma vez (no momento da montagem do componente), você pode passar um array vazio como segundo argumento:

```
useEffect(() => {  
  // Código do efeito  
}, []);
```

Isso garante que o efeito seja executado somente após a montagem inicial do componente. Quando o segundo argumento é um array vazio, o efeito não depende de nenhuma variável específica e, portanto, não será reexecutado em atualizações subsequentes.

1.5

No contexto de um browser, a avaliação da seguinte expressão resulta:

```
fetch('https://httpbin.org/status/404').status
```

- i. No número 404.
- ii. No número 200.
- iii. No valor null.
- iv. Nenhuma das anteriores.

ii

Resposta com sucesso a um fetch para um recurso retorna sempre 200

1.6

O resultado da avaliação da expressão JSX

é: i. Um elemento HTML, do mesmo tipo do obtido na avaliação da expressão `document.createElement('Div')`. ii. O resultado da avaliação da expressão `React.createElement(Div, null, document.createElement('div'))`. iii. O resultado da avaliação da expressão `React.createElement(Div, null, React.createElement('div', null))`. iv. O resultado da avaliação da expressão `React.createElement(Div, null, React.createElement(div, null))`.

iii

- No JSX, `<Div>` e `<div>` são componentes React.
- Quando o JSX é transpilado para código JavaScript, ele se torna uma chamada para `React.createElement`.
- Portanto, `<Div>` é equivalente a `React.createElement(Div, null, ...)` e `<div>` é equivalente a `React.createElement('div', null, ...)`.
- A expressão `<Div><div /></Div>` cria um componente `<Div>` que contém um elemento `<div>` como filho.

Lembre-se de que os nomes dos componentes no JSX devem começar com letra maiúscula para que o transpilador os reconheça como componentes React.

2

Indique dois motivos para a utilização de hypermedia nas representações dos recursos de uma API HTTP.

1. Descoberta Dinâmica de Recursos:

- O uso de hypermedia permite que os clientes descubram dinamicamente os recursos disponíveis na API.
- Cada recurso pode incluir links para outros recursos relacionados, permitindo que os clientes naveguem pela API sem depender de conhecimento prévio.
- Isso facilita a evolução da API, pois os clientes podem se adaptar automaticamente às mudanças nos recursos e relacionamentos.

2. Flexibilidade e Evolução:

- Através do hypermedia, a API pode fornecer informações contextuais sobre como interagir com os recursos.
- Se a API precisar adicionar novos endpoints ou alterar a estrutura dos recursos, os clientes que seguem os links hypermedia ainda poderão funcionar corretamente.
- Isso promove uma abordagem mais flexível e evolutiva para o design da API, permitindo que ela cresça e mude ao longo do tempo.

Em resumo, o uso de hypermedia em APIs HTTP melhora a descoberta de recursos e torna a API mais adaptável a mudanças futuras.

3

No contexto da plataforma Spring MVC, quais os critérios que se deve ter em conta para escolher um servlet filter ou um handler interceptor como ponto de extensibilidade a usar para criar um intermediário no processamento de pedidos.

No contexto da plataforma **Spring MVC**, a escolha entre usar um **servlet filter** ou um **handler interceptor** como ponto de extensibilidade para criar um intermediário no processamento de pedidos depende de vários critérios. Vamos analisar esses critérios:

1. Escopo e Propósito:

- **Servlet Filter:**
 - Os filtros são parte do servidor web (como Tomcat, Jetty etc.) e não fazem parte do framework Spring.
 - Eles operam em um nível mais baixo, antes que as solicitações alcancem qualquer componente do Spring MVC.
 - Filtros podem ser aplicados a todas as solicitações e respostas, independentemente do contexto do Spring.
 - São ideais para tarefas de nível mais geral, como autenticação, log e compressão de dados.
- **Handler Interceptor:**

- Os interceptores fazem parte do framework Spring MVC e operam no nível do DispatcherServlet.
- São específicos para o processamento de solicitações relacionadas a controladores (handlers) do Spring.
- Permitem a manipulação de solicitações antes que elas alcancem os controladores e também após a renderização da visão.
- São adequados para tarefas específicas do Spring MVC, como logging, autorização, manipulação de modelos, etc.

2. Ordem de Execução:

○ Servlet Filter:

- A ordem de execução dos filtros é definida no arquivo `web.xml` ou por meio de anotações.
- A ordem é importante, pois os filtros são executados na sequência em que são declarados.

○ Handler Interceptor:

- A ordem de execução dos interceptores é configurada no contexto do Spring MVC.
- Os interceptores são executados na ordem em que são registrados.

3. Acesso ao Contexto do Spring:

○ Servlet Filter:

- Os filtros não têm acesso direto ao contexto do Spring.
- Se você precisar de recursos específicos do Spring, como beans gerenciados pelo contêiner, não poderá acessá-los diretamente nos filtros.

○ Handler Interceptor:

- Os interceptores têm acesso ao contexto do Spring.
- Eles podem injetar beans do Spring e usar recursos específicos do Spring.

4. Granularidade e Flexibilidade:

○ Servlet Filter:

- São mais granulares e podem ser aplicados globalmente a todas as solicitações.
- Oferecem maior flexibilidade, mas também maior complexidade.

○ Handler Interceptor:

- São mais específicos para controladores e permitem manipulação mais focada.
- São mais fáceis de configurar e usar.

Em resumo, escolha um **servlet filter** quando precisar de funcionalidades globais ou de nível mais baixo, e um **handler interceptor** quando precisar de manipulação específica do Spring MVC. Ambos têm seus casos de uso e devem ser escolhidos com base nos requisitos específicos do seu aplicativo.

Source: Conversation with Bing, 1/6/2024

(1) HandlerInterceptors vs. Filters in Spring MVC | Baeldung. <https://www.baeldung.com/spring-mvc-handlerinterceptor-vs-filter>.

(2) What is the Difference Between Spring Interceptor and Filter? Explained <https://colorinfgfolder.com/what-is-the-difference-between-spring-interceptor-and-filter/>.

(3) Difference between Interceptor and Filter in Spring MVC. <https://stackoverflow.com/questions/>

4

Tendo em consideração o modelo de construção de aplicações para a plataforma browser usado na unidade curricular, indique qual o propósito e forma de utilização da ferramenta webpack

No contexto da construção de aplicações para a plataforma **browser**, o **Webpack** é uma ferramenta essencial com os seguintes propósitos e formas de utilização:

1. Propósito:

- O Webpack é um **module bundler** (empacotador de módulos).
- Seu principal objetivo é **empacotar** (ou seja, combinar e transformar) arquivos JavaScript e outros recursos (como CSS, imagens, fontes etc.) para uso em um navegador.
- Ele otimiza o carregamento de recursos, gerencia dependências e permite que os desenvolvedores organizem e modularizem seus códigos.

2. Forma de Utilização:

- O Webpack é configurado por meio de um arquivo chamado **webpack.config.js** (ou outros formatos de configuração, como **webpack.config.ts** para TypeScript).
- Nesse arquivo, você define:
 - **Entrypoints:** Os pontos de entrada da sua aplicação (geralmente arquivos JavaScript).
 - **Loaders:** Transformações aplicadas aos arquivos (por exemplo, transpilação de TypeScript, processamento de CSS etc.).
 - **Plugins:** Funcionalidades adicionais, como minificação, geração de HTML, otimização de imagens etc.
 - **Output:** Como os arquivos empacotados devem ser gerados (nome, localização etc.).

3. Exemplo de Configuração Básica:

- Aqui está um exemplo simplificado de um arquivo `webpack.config.js`:

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        use: 'babel-loader',
        exclude: /node_modules/,
      }
    ]
  }
}
```

```

    },
    // Outros loaders para CSS, imagens, etc.
  ],
  },
  // Plugins, otimizações, etc.
};

```

4. Execução do Webpack:

- Execute o Webpack usando o comando `npx webpack` ou configure um script no seu `package.json`.
- O Webpack criará o arquivo de saída (por exemplo, `bundle.js`) no diretório especificado (como `dist`).

Em resumo, o Webpack é uma ferramenta poderosa para empacotar e otimizar recursos para uso em navegadores, permitindo que os desenvolvedores criem aplicações mais eficientes e bem organizadas. □□

Source: Conversation with Bing, 1/6/2024

(1) Why webpack | webpack. <https://webpack.js.org/concepts/why-webpack/>.

(2) Concepts | webpack. <https://webpack.js.org/concepts/>.

(3) What is Webpack? - Javascript Tutorial. <https://www.javascripttutorial.org/what-is-webpack/>.

(4) github.com. <https://github.com/Technology-Notes/wbwangk-wbwangk.github.io-wiki/tree/f7071df9b12c4ca5b5b84aa3e495a9192ede6df7/webpack.md>.

(5) github.com. <https://github.com/QuabenaBoahen/eShop-frontend-react/tree/4320a417a9ef4069ef9fae307c9d18cceda34504/webpack.config.js>.

(6) github.com. https://github.com/MarinaOsborne/AJS_homework_2/tree/efb653488dbd9fad0a8dfae306abd6e3b20b153/webpack.config.js.

(7) github.com. <https://github.com/GanulAnton/rslang/tree/65f2966023a427540dc0112902d1bd3ec9dfb9c1/webpack.config.js>.

5

Realize um ou mais componentes para a plataforma Spring MVC de forma a que todas as respostas produzidas tenham o header HTTP Debug contendo o tempo em milissegundos que a resposta demorou a ser processada, bem como o eventual handler envolvido nesse processamento. Valorizam-se soluções em que o cálculo do tempo de processamento inclua não só o tempo de execução do eventual handler mas também o da maioria dos intermediários envolvidos (e.g. filtros e interceptores)

Para adicionar um header HTTP de depuração contendo o tempo de processamento em milissegundos, incluindo o tempo de execução do handler e outros intermediários (como filtros e interceptores), podemos criar um interceptor personalizado no Spring MVC.

1. Crie um Interceptor:

- Crie uma classe que implemente a interface `HandlerInterceptor`.
- Sobrescreva os métodos `preHandle`, `postHandle` e `afterCompletion`.
- Calcule o tempo de processamento no método `afterCompletion`.

2. Exemplo de Interceptor:

```
import org.springframework.web.servlet.HandlerInterceptor
```

```

import org.springframework.web.servlet.ModelAndView
import javax.servlet.http.HttpServletRequest
import javax.servlet.http.HttpServletResponse

class DebugHeaderInterceptor : HandlerInterceptor {

    override fun preHandle(request: HttpServletRequest, response:
HttpServletResponse, handler: Any): Boolean {
        // Executa antes do handler (controlador)
        return true
    }

    override fun postHandle(request: HttpServletRequest, response:
HttpServletResponse, handler: Any, modelAndView: ModelAndView?) {
        // Executa após o handler (controlador), mas antes da renderização da
visão
    }

    override fun afterCompletion(request: HttpServletRequest, response:
HttpServletResponse, handler: Any, ex: Exception?) {
        // Executa após a renderização da visão
        val startTime = request.getAttribute("startTime") as Long
        val endTime = System.currentTimeMillis()
        val processingTime = endTime - startTime

        response.addHeader("X-Debug-Processing-Time", "$processingTime ms")
        response.addHeader("X-Debug-Handler", handler.toString())
    }
}

```

3. Registre o Interceptor:

- No seu arquivo de configuração (por exemplo, `WebMvcConfigurer`), registre o interceptor:

```

import org.springframework.context.annotation.Configuration
import org.springframework.web.servlet.config.annotation.InterceptorRegistry
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer

@Configuration
class WebConfig : WebMvcConfigurer {

    override fun addInterceptors(registry: InterceptorRegistry) {
        registry.addInterceptor(DebugHeaderInterceptor())
    }
}

```

4. Uso do Header:

- Agora, todas as respostas terão os headers `X-Debug-Processing-Time` e `X-Debug-Handler` contendo o tempo de processamento e o handler envolvido.

Realize um componente para a framework React que:

- Apresenta o valor de um contador, iniciado com zero.
- Apresenta uma caixa de texto e um botão. Sempre que o botão for premido, o valor presente na caixa de texto deve ser usado como o período, em milissegundos, com que o contador é incrementado. Inicialmente, e sempre que o botão seja premido com a caixa de texto vazia, o período de contagem deve ser infinito

Tenha em consideração que a função de actualização de estado, retornada pelo hook `useState`, também pode receber uma função que recebe o valor de estado anterior e retorna o novo valor pretendido para o estado.

```
import React, { useState, useEffect } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);
  const [period, setPeriod] = useState(null);
  const [intervalId, setIntervalId] = useState(null);

  const handleStartClick = () => {
    if (period !== null) {
      clearInterval(intervalId);
      const newIntervalId = setInterval(() => {
        setCount(prevCount => prevCount + 1);
      }, period);
      setIntervalId(newIntervalId);
    }
  };

  const handlePeriodChange = (event) => {
    const newPeriod = parseInt(event.target.value, 10);
    setPeriod(newPeriod);
  };

  useEffect(() => {
    handleStartClick(); // Start the counter initially
    return () => clearInterval(intervalId); // Cleanup on unmount
  }, []);

  return (
    <div>
      <h1>Counter: {count}</h1>
      <input
        type="number"
        placeholder="Enter period in milliseconds"
        value={period || ''}
        onChange={handlePeriodChange}
      />
      <button onClick={handleStartClick}>Start</button>
    </div>
  );
};
```

```
export default Counter;
```

Neste componente:

- O contador é iniciado com zero e atualizado a cada período definido.
- O usuário pode inserir um período em milissegundos na caixa de texto.
- Quando o botão "Start" é pressionado, o contador começa a incrementar com o período especificado.
- Se o período for deixado em branco, o contador não será atualizado.

Época Normal, Semestre de Verão, 21/22

1.1

No protocolo HTTP, o conceito de interface uniforme significa que:

- i. A semântica de um header de resposta não depende do recurso alvo.
- ii. Os identificadores de recurso estão presentes no caminho e não na query string.
- iii. O mesmo media type tem de ser usado em todas as respostas.
- iv. Os únicos métodos que podem ser usados são o POST and GET.

i

- O conceito de **interface uniforme** no protocolo HTTP é uma das restrições fundamentais definidas por Roy Fielding em sua tese de doutorado.
- A interface uniforme tem quatro princípios:
 1. **Identificação de Recursos:** Cada recurso é identificado por um URI (Uniform Resource Identifier).
 2. **Manipulação de Recursos através de Representações:** Os clientes manipulam recursos por meio de representações (como HTML, JSON etc.) obtidas por meio de requisições HTTP.
 3. **Mensagens Autodescritivas:** As mensagens HTTP contêm informações suficientes para que os destinatários entendam como processar a mensagem.
 4. **Hiperlinks:** As representações contêm links (hiperlinks) para outros recursos relacionados.

Em resumo, a interface uniforme no HTTP promove a identificação clara dos recursos por meio de URIs no caminho e incentiva o uso consistente de representações e hiperlinks.

1.2

No protocolo HTTP, uma mensagem de resposta de status code 400:

- i. Não pode ter payload.
- ii. Pode ter um payload usando um qualquer media type.
- iii. Tem de ter um payload usando o media type `application/problem+json`.
- iv. Tem de ter um payload usando o media type `application/json`

ii

A utilização do media type `application/problem+json` é recomendada porque fornece um formato padronizado para comunicar informações detalhadas sobre o erro. No entanto, em teoria, **não há uma restrição absoluta** que impeça o uso de **outros tipos de payload** em uma resposta de **status code 400**.

Se um servidor optasse por incluir um payload diferente, como um formato personalizado ou outro tipo de mídia, tecnicamente seria possível. No entanto, isso **não seguiria as melhores práticas** e poderia dificultar a interpretação consistente do erro pelos clientes.

Em resumo, embora outras opções sejam possíveis, o uso do `application/problem+json` é altamente recomendado para garantir a clareza e a interoperabilidade na comunicação de erros no protocolo HTTP.

1.3

Uma mensagem de pedido com método GET para o recurso <https://example.com/projects/123.json>:

- i. Tem de resultar numa mensagem de resposta com Content-Type igual a `application/json`.
- ii. Tem de resultar numa mensagem de resposta com Content-Type igual a `text/json`.
- iii. Tem de resultar numa mensagem de resposta com Content-Type igual a `application/json` ou a `application/problem+json`.
- iv. Pode resultar numa mensagem de resposta com qualquer valor de Content-Type.

iv (mas é recomendado iii)

Em caso de dúvidas, o professor disse para escrevermos o nosso raciocínio a fomentar a escolha

Claro, é possível retornar **qualquer tipo de Content-Type** em uma resposta HTTP. No entanto, é importante seguir as convenções e as melhores práticas para garantir a interoperabilidade e a compreensão adequada entre os sistemas.

A escolha do **Content-Type** deve ser baseada no contexto e no propósito da resposta:

- **`application/json`**: É amplamente utilizado para retornar dados estruturados no formato JSON. Se o objetivo da resposta for fornecer informações sobre o recurso solicitado (como no caso de uma requisição GET para `https://example.com/projects/123.json`), esse é o Content-Type mais apropriado.
- **`application/problem+json`**: Esse media type é recomendado para respostas de erro (status codes 4xx e 5xx). Ele permite que o servidor forneça detalhes específicos sobre o problema encontrado no formato JSON padronizado. No entanto, para respostas bem-sucedidas, o uso do `application/json` é mais comum.
- **Outros tipos de Content-Type**: Embora seja possível retornar outros tipos de mídia (como `text/plain`, `text/html`, etc.), é importante considerar a expectativa do cliente e a compatibilidade com os padrões. O uso de `application/json` ou `application/problem+json` é preferível para manter a consistência e a clareza na comunicação.

Em resumo, embora seja tecnicamente possível usar qualquer Content-Type, é recomendado seguir as práticas estabelecidas para garantir uma comunicação eficaz entre os sistemas. 😊

1.4

** Na plataforma Spring MVC, por omissão, uma classe anotada com @RestController vai ter as seguintes instâncias criadas pelo contexto Spring:

- i. Uma única instância.
- ii. Uma instância distinta por pedido.
- iii. Uma instância distinta por cada ligação de um cliente.
- iv. O menor número de instâncias, de**

i

Aqui está o motivo para essa escolha:

1. **Uma única instância:** Por padrão, os controladores Spring MVC são **singletons**. Isso significa que uma única instância do controlador é criada e compartilhada entre todas as solicitações e sessões. No entanto, isso pode levar a problemas de concorrência se não houver sincronização adequada.
2. **Uma instância distinta por pedido:** Embora seja possível configurar controladores com escopo de requisição (request scope), essa não é a configuração padrão. Controladores com escopo de requisição seriam criados separadamente para cada solicitação, mas essa não é a abordagem padrão.
3. **Uma instância distinta por cada ligação de um cliente:** Essa é a configuração padrão e mais apropriada para handlers Spring MVC (@GetMapping, @PostMapping, @RequestMapping). Cada cliente (ou conexão) receberá uma instância independente do controlador. Isso garante que os dados não sejam compartilhados entre diferentes clientes.
4. **O menor número de instâncias para evitar uso simultâneo por threads diferentes:** Embora seja uma preocupação válida, o escopo padrão de singleton é otimizado para minimizar o uso de recursos e garantir a reutilização eficiente das instâncias.

Em resumo, a opção i é a correta.

Source: Conversation with Bing, 1/7/2024

(1) Scope of a Spring-Controller and its instance-variables. <https://stackoverflow.com/questions/11139571/scope-of-a-spring-controller-and-its-instance-variables>.

(2) java - Are new instances of Service, Repository and Component created ... <https://stackoverflow.com/questions/67265229/are-new-instances-of-service-repository-and-component-created-for-every-new-req>.

(3) java - Spring MVC RestController scope - Stack Overflow. <https://stackoverflow.com/questions/33142344/spring-mvc-restcontroller-scope>.

(4) spring - why @RestController by default use singleton scope? - Stack ... <https://stackoverflow.com/questions/67184451/why-restcontroller-by-default-use-singleton-scope>.

(5) undefined. <https://rules.sonarsource.com/java/RSPEC-2119>.

1.5

No contexto de uma single page application, a avaliação da seguinte expressão resulta em:

```
window.location.pathname = '/projects'
```

- i. Em nenhum pedido HTTP executado.
- ii. Na execução de um pedido de método GET para o caminho `/index.html`.
- iii. Na execução de um pedido de método GET para o caminho `/projects`.
- iv. Na execução de um pedido de método GET para o caminho `/projects.html`.

iii

A resposta correta é a **opção iii**: a avaliação da expressão `window.location.pathname = '/projects'` resultaria em **um pedido de método GET para o caminho `/projects`**.

Aqui está o raciocínio por trás dessa escolha:

1. **Em nenhum pedido HTTP executado:** A expressão `window.location.pathname = '/projects'` não é uma ação em si; é apenas uma atribuição de valor à propriedade `pathname` do objeto `window.location`. Portanto, não resulta em nenhum pedido HTTP.
2. **Na execução de um pedido de método GET para o caminho `/index.html`:** Isso não é correto, pois a expressão especifica explicitamente o caminho `/projects`, não `/index.html`.
3. **Na execução de um pedido de método GET para o caminho `/projects`:** Essa é a resposta correta. Quando a expressão é avaliada, ela atualiza o caminho da URL para `/projects`, o que indica que o cliente está solicitando o recurso no caminho `/projects`.
4. **Na execução de um pedido de método GET para o caminho `/projects.html`:** Isso não é correto, pois a expressão não inclui a extensão `.html`.

Em resumo, a opção iii é a correta porque reflete a ação de atualizar o caminho da URL para `/projects`, o que corresponderia a um pedido de método GET para esse caminho específico em uma single page application.

Source: Conversation with Bing, 1/7/2024

- (1) Blank page error due to location pathname - Stack Overflow. <https://stackoverflow.com/questions/46153405/blank-page-error-due-to-location-pathname>.
- (2) Deciding on a Page Address with Window.location | Udacity. <https://www.udacity.com/blog/2021/05/javascript-window-location.html>.
- (3) How-to get the Application path using javascript - Stack Overflow. <https://stackoverflow.com/questions/3677338/how-to-get-the-application-path-using-javascript>.
- (4) undefined. <http://projects.jga.me/routie/%29>.
- (5) undefined. <https://example.com>.

1.6

A avaliação da seguinte expressão JSX

```
<Example a="b"/>
```

- i. E equivalente à avaliação da expressão `React.createElement(Example)({a:"b"})`.
- ii. E equivalente à avaliação da expressão `React.createElement(Example, {a:"b"})`.
- iii. E equivalente à avaliação da expressão `React.createElement(Example({a:"b"}))`.
- iv. Nenhuma das anteriores

ii

1. **Equivalente a `React.createElement(Example)({a:"b"})`:** Isso não é correto. A expressão `React.createElement(Example)` cria um elemento React para o componente `Example`, mas não aceita argumentos adicionais. Portanto, `{a:"b"}` não pode ser passado dessa maneira.

2. **Equivalente a `React.createElement(Example, {a: "b"})`**: Essa é a resposta correta. A função `React.createElement` aceita dois argumentos: o tipo de elemento (neste caso, o componente `Example`) e um objeto de propriedades (no formato `{a: "b"}`).
3. **Equivalente a `React.createElement(Example({a: "b"}))`**: Isso não é correto. Aqui, a expressão está chamando diretamente o componente `Example` com as propriedades `{a: "b"}`, o que não é a sintaxe correta para criar elementos React.
4. **Nenhuma das anteriores**: A opção ii é a correta, como explicado acima.

Em resumo, a opção ii é a mais apropriada para criar um elemento React correspondente ao componente `Example` com as propriedades especificadas.

2

**** No media type Siren, quais as diferenças na estrutura e no propósito dos links e das actions.****

No contexto do media type **Siren**, existem diferenças na estrutura e no propósito entre **links** e **actions**:

1. Links:

- **Estrutura**: Um link em Siren é representado como um objeto JSON com as seguintes propriedades:
 - `rel`: Uma lista de relações (rel) que descrevem o relacionamento entre a entidade atual e o recurso vinculado.
 - `href`: A URL do recurso vinculado.
 - Outras propriedades opcionais, como `class`, `title`, etc.
- **Propósito**: Os links são usados para navegação do cliente. Eles fornecem informações sobre recursos relacionados e permitem que o cliente descubra e acesse outros recursos.

2. Actions (ações):

- **Estrutura**: Uma ação em Siren é representada como um objeto JSON com as seguintes propriedades:
 - `name`: O nome da ação.
 - `method`: O método HTTP usado para executar a ação (por exemplo, POST, PUT, DELETE, etc.).
 - `href`: A URL onde a ação será executada.
 - `type`: O tipo de mídia usado para enviar os dados da ação (por exemplo, `application/x-www-form-urlencoded`, `application/json`, etc.).
 - `fields`: Uma lista de campos (inputs) necessários para a ação, como parâmetros ou dados a serem enviados.
- **Propósito**: As ações representam transições de estado. Elas permitem que o cliente execute operações no recurso atual, como criar, atualizar ou excluir dados.

Em resumo, **links** são usados para navegação e descoberta de recursos relacionados, enquanto **actions** são usadas para executar operações específicas nos recursos.

Source: Conversation with Bing, 1/7/2024

- (1) GitHub - kevinswiber/siren: Structured Interface for Representing <https://github.com/kevinswiber/siren>.
- (2) What's different/better/worse than other JSON hypermedia media types <https://github.com/kevinswiber/siren/issues/15>.
- (3) On choosing a hypermedia type for your API - Sookocheff. <https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/>.
- (4) undefined. <http://x.io/rels/order-items>.
- (5) undefined. <http://api.x.io/orders/42/items>.
- (6) undefined. <http://x.io/rels/customer>.
- (7) undefined. <http://api.x.io/customers/pj123>.
- (8) undefined. <http://api.x.io/orders/42>.
- (9) undefined. <http://api.x.io/orders/41>.
- (10) undefined. <http://api.x.io/orders/43>.

3

No contexto de uma aplicação que usa as bibliotecas React e React Router, qual a diferença entre a utilização dos elementos `<a />` e `<Link />`?

No contexto de uma aplicação que utiliza as bibliotecas **React** e **React Router**, a diferença entre os elementos `<a />` e `<Link />` é significativa e tem a ver com a navegação e o roteamento dentro da aplicação:

1. `<a />` (**âncora**):
 - **Estrutura:** O elemento `<a />` é uma tag HTML padrão que cria um link para uma URL externa ou interna. Ele não está diretamente relacionado ao React Router.
 - **Propósito:**
 - O `<a />` é usado para criar links tradicionais em páginas HTML.
 - Quando usado em uma aplicação React, ele recarrega a página inteira quando o link é clicado, resultando em uma navegação completa.
 - Não mantém o estado da aplicação entre as transições.
2. `<Link />`:
 - **Estrutura:** O `<Link />` é um componente fornecido pela biblioteca **React Router** (especificamente pelo pacote `react-router-dom`). Ele é projetado para ser usado com o roteamento em aplicações React.
 - **Propósito:**
 - O `<Link />` é usado para criar links internos dentro da aplicação, sem recarregar a página inteira.
 - Quando um `<Link />` é clicado, o React Router atualiza a URL e renderiza o componente associado à nova rota, mantendo o estado da aplicação.
 - É uma alternativa mais eficiente e amigável para Single Page Applications (SPAs), pois evita recarregamentos completos da página.

Em resumo:

- Use `<a />` para links tradicionais que recarregam a página inteira.

- Use `<Link />` para navegação interna em SPAs, mantendo o estado da aplicação e evitando recarregamentos desnecessários.

Para mais informações sobre o React Router, você pode consultar a [documentação oficial](#). 😊

Source: Conversation with Bing, 1/7/2024

- (1) How does React router works and what is the difference between `<link ...` <https://stackoverflow.com/questions/50807929/how-does-react-router-works-and-what-is-the-difference-between-link-and-route>.
- (2) React Router - W3Schools. https://www.w3schools.com/react/react_router.asp.
- (3) React Router DOM: The Differences Between NavLink, Link, and. <https://plainenglish.io/blog/react-router-dom-the-differences-between-navlink-link-and-a>.
- (4) react-router vs react-router-dom, when to use one or the other?. <https://stackoverflow.com/questions/42684809/react-router-vs-react-router-dom-when-to-use-one-or-the-other>.
- (5) The difference between react-router and react-router-dom. <https://www.ilovereactjs.com/blog/react/the-difference-between-react-router-and-react-router-dom>.
- (6) undefined. <https://reacttraining.com/react-router/>.
- (7) undefined. <https://reacttraining.com/react-router/web/guides/philosophy>.
- (8) undefined. <https://reactjs.org/docs/hello-world.html>.

4

Descreva os elementos do processo de construção de aplicações usado nesta unidade curricular que permitem a utilização de bibliotecas fornecidas pelo NPM em aplicações para execução no browser.

Segundo explicação do professor em aula de dúvidas:

Os elementos do processo são:

- junção de todos os módulos num único, feito pelo bundle
- no código de cada módulo a usar, é necessário que o mesmo use a função `import/require` e `export`
- todos os módulos são colocados num único ficheiro

Resposta do Bing copilot:

Para construir aplicações que utilizam bibliotecas fornecidas pelo **NPM** (Node Package Manager) e executam no navegador, é importante entender os seguintes elementos do processo:

1. Node.js e NPM:

- **Node.js:** É uma plataforma de tempo de execução JavaScript que permite executar código JavaScript fora do navegador. Ele inclui o NPM como parte de sua instalação.
- **NPM:** É o gerenciador de pacotes do Node.js. Ele permite instalar, gerenciar e compartilhar bibliotecas e módulos JavaScript.

2. Estrutura do Projeto:

- Crie uma estrutura de diretórios para o seu projeto. Geralmente, isso inclui pastas como `src` (para o código-fonte), `public` (para arquivos estáticos) e `node_modules` (onde as bibliotecas instaladas pelo NPM ficam armazenadas).

3. package.json:

- O arquivo `package.json` é essencial para o gerenciamento de dependências.
- Use o comando `npm init` para criar um novo arquivo `package.json`. Ele contém informações sobre o projeto, suas dependências, scripts de build e outras configurações.

4. Instalação de Dependências:

- Use o comando `npm install <package-name>` para instalar bibliotecas do NPM.
- As dependências são listadas no `package.json` sob a chave `"dependencies"`.
- O diretório `node_modules` conterá os códigos-fonte das bibliotecas instaladas.

5. Bundler (Empacotador):

- Para usar bibliotecas do NPM no navegador, você precisa empacotá-las em um único arquivo.
- Ferramentas como **Webpack**, **Browserify** ou **Parcel** podem ser usadas para criar um único arquivo JavaScript (bundle) que inclui todas as dependências.

6. Importação no Código:

- No seu código-fonte (geralmente em arquivos `.js`), importe as bibliotecas usando `import` ou `require`.
- Por exemplo:

```
// Usando ES6 Modules
import axios from 'axios';

// Usando CommonJS
const express = require('express');
```

7. Transpilação e Build:

- Se você estiver usando recursos modernos do JavaScript (como ES6+), é necessário transpilá-los para uma versão compatível com navegadores mais antigos.
- Use ferramentas como **Babel** para transpilar seu código.
- Configure scripts de build no `package.json` para automatizar o processo de transpilação e criação do bundle.

8. Integração com HTML:

- No seu arquivo HTML, inclua o bundle gerado (geralmente chamado de `bundle.js`) usando uma tag `<script>`:

```
<script src="bundle.js"></script>
```

9. Servidor Local:

- Para testar sua aplicação localmente, configure um servidor local (por exemplo, usando **Express.js**).
- Isso permitirá que você acesse sua aplicação no navegador.

10. Execução no Navegador:

- Abra seu aplicativo no navegador e verifique se tudo está funcionando conforme o esperado.

Em resumo, o processo envolve instalar bibliotecas via NPM, empacotá-las usando um bundler, importá-las no código-fonte, transpilar se necessário e executar a aplicação no navegador. Essa abordagem permite aproveitar a rica variedade de bibliotecas disponíveis no ecossistema do NPM para construir aplicações web modernas. 😊

Source: Conversation with Bing, 1/7/2024

(1) Let's Build a Web App with NPM and Express - This Dot Labs. <https://www.thisdot.co/blog/lets-build-a-web-app-with-npm-and-express>.

(2) A Simple Way to Use NPM Library in Browser. <https://bartsolutions.github.io/2019/12/27/use-npm-in-browser-js/>.

(3) npm - How to use a node.js library in html? - Stack Overflow. <https://stackoverflow.com/questions/38454729/how-to-use-a-node-js-library-in-html>.

(4) How to Use NPM Modules in Your Browser in Three Steps. <https://webtips.dev/use-npm-modules-in-your-browser>.

(5) Creating a simple npm library to use in and out of the browser. <https://www.intricatecloud.io/2020/02/creating-a-simple-npm-library-to-use-in-and-out-of-the-browser/>.

5

Realize um ou mais componentes para a plataforma Spring MVC, de forma a que um recurso seja exposto no caminho `/pending`. Um pedido de método GET para esse recurso deve retornar uma mensagem com uma representação contendo um objeto JSON. Esse objeto deve representar o número de pedidos atualmente em processamento (i.e. iniciados e não concluídos), agrupados por método HTTP.

Componente Spring MVC em Kotlin para expor o recurso no caminho `/pending`. O objetivo é retornar um objeto JSON que represente o número de pedidos atualmente em processamento, agrupados por método HTTP.

1. Crie um Controller:

- Vamos criar um controlador chamado `PendingController` que será responsável por lidar com as solicitações para o recurso `/pending`.

2. Defina o Endpoint:

- O endpoint `/pending` responderá a solicitações de método GET.

3. Contagem de Pedidos:

- Para este exemplo, vou simular uma contagem de pedidos em processamento. Você pode adaptar isso para sua lógica real.
- Vou criar um objeto JSON com as contagens para cada método HTTP (GET, POST, PUT, DELETE).

Exemplo:

```
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("/pending")
```

```
class PendingController {

    // Simulando contagem de pedidos em processamento
    private val pendingRequests = mapOf(
        "GET" to 10,
        "POST" to 5,
        "PUT" to 3,
        "DELETE" to 2
    )

    @GetMapping
    fun getPendingRequests(): Map<String, Int> {
        return pendingRequests
    }
}
```

Neste exemplo:

- O endpoint `/pending` responde a solicitações de método GET.
- A contagem de pedidos em processamento é simulada no mapa `pendingRequests`.

Quando você acessar `http://localhost:8080/pending`, receberá uma resposta JSON como esta:

```
{
  "GET": 10,
  "POST": 5,
  "PUT": 3,
  "DELETE": 2
}
```

Source: Conversation with Bing, 1/7/2024

- (1) Spring MVC Setup with Kotlin | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/spring-mvc-kotlin>.
- (2) Resources :: Spring Framework. <https://docs.spring.io/spring-framework/reference/core/resources.html>.
- (3) The State of Kotlin Support in Spring | The Kotlin Blog. <https://blog.jetbrains.com/kotlin/2020/08/the-state-of-kotlin-support-in-spring/>.
- (4) Guide to the Kotlin Exposed Framework | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/exposed-persistence>.
- (5) Serve Static Resources with Spring | Baeldung. <https://www.baeldung.com/spring-mvc-static-resources>.

6

Realize um componente React com as seguintes propriedades: um array de strings messages, um número period, e um componente component.

Este componente deve apresentar cada mensagem do array messages durante period milissegundos, avançando automaticamente para a próxima mensagem depois de decorrido esse tempo. A seguir á última mensagem, o processo deve recomeçar na primeira mensagem

A apresentação das mensagens deve ser feita com recurso ao componente component, nomeadamente através da sua propriedade text com tipo string

O componente realizado deve reagir correctamente a mudanças nas suas propriedades.

Componente React que atenda aos requisitos especificados. Vamos chamá-lo de

`MessageCarousel`.

Exemplo:

```
import React, { useState, useEffect } from 'react';

const MessageCarousel = ({ messages, period, component: Component }) => {
  const [currentMessageIndex, setCurrentMessageIndex] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setCurrentMessageIndex((prevIndex) => (prevIndex + 1) % messages.length);
    }, period);

    return () => {
      clearInterval(intervalId);
    };
  }, [messages, period]);

  return (
    <Component text={messages[currentMessageIndex]} />
  );
};

export default MessageCarousel;
```

Neste exemplo:

- O componente `MessageCarousel` recebe as propriedades `messages` (um array de strings), `period` (o tempo em milissegundos para cada mensagem) e `component` (o componente que renderizará a mensagem).
- Usamos o `useState` para controlar o índice da mensagem atual.
- O `useEffect` é usado para avançar automaticamente para a próxima mensagem após o período especificado.
- O componente renderiza a mensagem atual usando a propriedade `text` do componente fornecido.

Poderemos usar o `MessageCarousel` assim:

```
// Exemplo de uso
import React from 'react';
import MessageCarousel from './MessageCarousel'; // Importe o componente

const App = () => {
  const messages = ['Olá!', 'Bem-vindo!', 'React é incrível!'];
  const period = 2000; // 2 segundos

  // Componente que renderiza a mensagem
  const MessageComponent = ({ text }) => <div>{text}</div>;

  return (
```

```

    <div>
      <h1>Message Carousel</h1>
      <MessageCarousel messages={messages} period={period} component=
{MessageComponent} />
    </div>
  );
};

export default App;

```

O componente `MessageCarousel` reagirá automaticamente às mudanças nas propriedades `messages` e `period`. 😊

Época de Recurso, Semestre de Verão, 21/22

1.1

****No protocolo HTTP, um pedido de método PUT para o recurso <https://example.com/projects>:**

- i. Solicita a criação ou atualização do recurso com URI <https://example.com/projects>.
- ii. Solicita a atualização do recurso com URI <https://example.com/projects>, devendo resultar em erro caso o recurso não exista.
- iii. Solicita a criação dum recurso com URI <https://example.com/projects/{id}>, para um valor de id controlado pelo servidor.
- iv. Solicita a representação do recurso com URI <https://example.com/projects>

i

Explicação:

- O método PUT é usado para criar ou atualizar recursos em um servidor.
- Quando um cliente envia um pedido PUT para uma URI específica, ele está indicando que deseja criar ou atualizar o recurso representado por essa URI.
- Se o recurso ainda não existir, ele será criado. Se já existir, ele será atualizado com os dados fornecidos no corpo da requisição.

Em resumo, o método PUT é usado para manipular recursos existentes ou criar novos recursos, dependendo da existência prévia da URI especificada. 😊

1.2

No protocolo HTTP, uma mensagem de resposta com status code 401:

- i. Deve originar o reenvio da mensagem de pedido associada, sem alterações, após decorrido um intervalo de tempo.
- ii. Deve conter o header de resposta WWW-Authenticate.
- iii. Deve conter o header de resposta Authorization.
- iv. Deve conter um payload com o media type application/json.

ii

Explicação:

- O status code 401 (Unauthorized) indica que o cliente não está autorizado a acessar o recurso solicitado.

- Quando um servidor envia uma resposta com status 401, ele deve incluir o header `WWW-Authenticate` para indicar ao cliente como ele pode se autenticar corretamente.
- O header `WWW-Authenticate` fornece informações sobre o esquema de autenticação esperado (por exemplo, Basic, Bearer, etc.).

Em resumo, o header `WWW-Authenticate` é essencial para informar ao cliente como proceder com a autenticação para acessar o recurso protegido. 😊

1.3

Uma mensagem de pedido com método GET para o recurso

<https://example.com/projects/123?accept=application/json>:

- Tem de resultar numa mensagem de resposta com Content-Type igual a `application/json`.
- Tem de resultar numa mensagem de resposta com Content-Type igual a `text/json`.
- Tem de resultar numa mensagem de resposta com Content-Type igual a `application/json` ou a `application/problem+json`.
- Pode resultar numa mensagem de resposta com qualquer valor de Content-Type.

iv

Pode resultar em qualquer media-type definido pela API, logo qualquer Content-type

1.4

** No contexto da plataforma Spring MVC

- Vai existir uma instância implementando a interface `HttpServletRequest` partilhada por todos os pedidos.
- Vai existir uma instância implementando a interface `HttpServletRequest` partilhada por todos os pedidos para o mesmo controlador.
- Vai existir uma instância implementando a interface `HttpServletRequest` partilhada por todos os pedidos para o mesmo handler.
- Nenhuma das anteriores.**

iv

Explicação:

- No contexto da plataforma Spring MVC, o **`HttpServletRequest`** é um objeto que representa a requisição HTTP feita pelo cliente (navegador) ao servidor.
- Cada pedido HTTP cria uma nova instância de **`HttpServletRequest`**.
- Essa instância contém informações sobre o pedido, como parâmetros, cabeçalhos, método HTTP, URL, etc.
- Não há compartilhamento de instâncias entre diferentes pedidos. Cada pedido tem sua própria instância.

Em resumo, cada pedido HTTP cria uma nova instância de **`HttpServletRequest`**, e não há compartilhamento entre pedidos. 😊

1.5

No contexto de uma single page application usando a biblioteca React Router, a ativação pelo utilizador do elemento DOM produzido em resultado de* `<Link to="/path"/> *`, realiza:

- i. Um pedido de método GET ao URI com caminho `/path`.
- ii. Um pedido de método GET ao URI com caminho `/index.html`.
- iii. Um pedido de método GET ao URI com caminho `/bundle.js`.
- iv. Nenhuma das anteriores.

iv

Explicação:

- Quando um usuário ativa o elemento DOM produzido pelo componente `<Link to="/path"/>` do React Router, não resulta em nenhum pedido HTTP.
- O React Router manipula a navegação internamente, atualizando a URL e renderizando o componente associado à nova rota, tudo sem recarregar a página inteira.
- Portanto, não há solicitações de método GET para `/path`, `/index.html` ou `/bundle.js`.

Em resumo, o React Router lida com a navegação internamente, sem fazer solicitações HTTP adicionais. ☺

Source: Conversation with Bing, 1/7/2024

- (1) React Router v4 - How to get current route? - Stack Overflow. <https://stackoverflow.com/questions/42253277/react-router-v4-how-to-get-current-route>.
- (2) javascript - Get path params in react-router v4 - Stack Overflow. <https://stackoverflow.com/questions/45468837/get-path-params-in-react-router-v4>.
- (3) Get the current URL and Route in React and React Router. <https://bobbyhadz.com/blog/react-get-current-url>.
- (4) how to get a URL parameters in react router V6? - Stack Overflow. <https://stackoverflow.com/questions/75054449/how-to-get-a-url-parameters-in-react-router-v6>.

1.6

A avaliação da seguinte expressão JSX

```
<Example render={()=><Inside />} />
```

- i. E equivalente à avaliação da expressão `React.createElement(Example, null, Inside)`.
- ii. E equivalente à avaliação da expressão `React.createElement(Example, {render: () => createElement(Inside)})`.
- iii. E equivalente à avaliação da expressão `React.createElement(Example(Inside()))`.
- iv. Nenhuma das anteriores.

ii

Explicação:

- A expressão JSX está passando uma propriedade chamada `render` para o componente `Example`.
- Essa propriedade é uma função de seta que retorna o componente `Inside`.
- A função `React.createElement` cria um elemento React com as propriedades especificadas.

Em resumo, a opção ii é a correta porque representa corretamente a criação de um elemento React com a propriedade `render` que contém o componente `Inside`. 😊

2

No contexto do protocolo HTTP, indique as vantagens do conceito de interface uniforme, fornecido por este protocolo

O conceito de **interface uniforme** no contexto do protocolo HTTP (Hypertext Transfer Protocol) traz várias vantagens importantes para o design e a evolução de serviços web.

1. Simplificação e Desacoplamento:

- A interface uniforme simplifica e desacopla a arquitetura.
- Cada parte do sistema pode evoluir independentemente, sem afetar outras partes.
- Isso é fundamental para a escalabilidade e a manutenção de sistemas distribuídos.

2. Princípios da Interface Uniforme:

- A interface uniforme é baseada em quatro princípios:
 - **Identificação de Recursos:**
 - Os recursos (objetos ou informações) são identificados por URIs (Uniform Resource Identifiers).
 - As representações dos recursos podem variar (HTML, XML, JSON), mas o recurso em si é o mesmo.
 - **Manipulação de Recursos por Representações:**
 - Quando um cliente possui uma representação de um recurso, ele tem informações suficientes para modificá-lo ou excluí-lo.
 - **Mensagens Autoexplicativas:**
 - Cada mensagem inclui informações suficientes para descrever como processá-la.
 - Os tipos de mídia (MIME types) especificam como interpretar os dados.
 - As respostas indicam explicitamente se são cacheáveis.
 - **HATEOAS (Hypermedia as the Engine of Application State):**
 - Os clientes navegam entre estados de aplicação apenas através de ações identificadas dinamicamente no conteúdo (por exemplo, hiperlinks em hipertexto).
 - Os clientes não assumem a disponibilidade de ações específicas para recursos específicos.

3. Portabilidade e Evolução:

- A interface uniforme permite que os clientes se adaptem facilmente a diferentes serviços.
- Os serviços podem evoluir sem impactar os clientes existentes.
- Isso é crucial para a manutenção e a escalabilidade de sistemas distribuídos.

Em resumo, a interface uniforme no protocolo HTTP simplifica a comunicação, permite a evolução independente de partes do sistema e promove a interoperabilidade entre diferentes serviços. 😊

Source: Conversation with Bing, 1/7/2024

- (1) REST - What exactly is meant by Uniform Interface?. <https://stackoverflow.com/questions/25172600/rest-what-exactly-is-meant-by-uniform-interface>.
- (2) Difference Between REST and HTTP | Baeldung on Computer Science. <https://www.baeldung.com/cs/rest-vs-http>.
- (3) RESTful Statelessness - GeeksforGeeks. <https://www.geeksforgeeks.org/restful-statelessness/>.
- (4) What is REST?. <https://www.restapitutorial.com/lessons/whatisrest.html>.

3

Qual o propósito e estrutura do header Link, definido na especificação RFC 8288 - Web Linking.

Segundo o professor na aula de dúvidas

- Estrutura:
 - URi (obrigatório)
 - rel, identificador da relação (ex: next, self) (obrigatório)
 - atributos adicionais (opcional) (ex: title)
 - media-type (opcional) - para informar como representar o recurso alvo
- Propósito
 - fornecer ao cliente informação sobre um recurso relacionado, colocando a informação no header em vez de no body
- Exemplo de caso de utilização, quando a representação (body) não permite a representação de links

Resposta do Bing copilot:

O **header Link**, definido na especificação **RFC 8288 - Web Linking**, tem como objetivo fornecer uma maneira padronizada de expressar relacionamentos entre recursos na Web. Ele permite que os servidores comuniquem informações sobre links (relações) associados a um recurso específico. Aqui estão os detalhes sobre o objetivo e a estrutura desse header:

1. Objetivo:

- O objetivo do header Link é definir um modelo para as relações entre recursos na Web (chamados de "links") e os tipos dessas relações (chamados de "link relation types").
- Ele fornece uma maneira de serializar esses links em cabeçalhos HTTP usando o campo de cabeçalho `Link`.

2. Estrutura:

- O header Link é composto por uma ou mais entradas, cada uma representando um link.
- Cada entrada consiste em:
 - **URI do recurso relacionado:** Representa o recurso vinculado.
 - **Relação (link relation type):** Descreve o tipo de relação entre o recurso atual e o recurso vinculado.

- **Outras informações opcionais:** Pode incluir atributos como `title`, `media`, `hreflang`, etc.

3. Exemplo:

- Um exemplo de uso do header Link:

```
Link: </TheBook/chapter2>; rel="previous"; title*=UTF-8'de'letztes%20Kapitel, </TheBook/chapter4>; rel="next"; title*=UTF-8'de'n%c3%a4chstes%20Kapitel
```

- Neste exemplo, temos dois links:
 - O primeiro link aponta para o capítulo anterior com título em alemão.
 - O segundo link aponta para o próximo capítulo com título em alemão.

Em resumo, o header Link é uma maneira padronizada de expressar relações entre recursos na Web, permitindo que os clientes descubram e naveguem entre recursos relacionados. 😊

Source: Conversation with Bing, 1/7/2024

(1) RFC 8288: Web Linking - RFC Editor. <https://www.rfc-editor.org/rfc/rfc8288>.

(2) Information on RFC 8288 » RFC Editor. <https://www.rfc-editor.org/info/rfc8288>.

(3) Link - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link>.

(4) GitHub - PulsarBlow/WebLinking: □ A .NET implementation of the Web <https://github.com/PulsarBlow/WebLinking>.

(5) undefined. <https://trustee.ietf.org/license-info%29>.

4

No contexto da biblioteca React, o recurso identificado por <https://reactjs.org/docs/hooks-rules.html> tem presente a seguinte afirmação na sua representação

Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns.

Indique qual a razão para esta limitação.

A razão para esta limitação é que os Hooks dependem da ordem em que são chamados para funcionar corretamente. Se os Hooks forem chamados dentro de loops, condições ou funções aninhadas, a ordem pode variar entre diferentes renderizações, o que pode causar bugs e inconsistências. Ao usar os Hooks no nível superior da função React, garante-se que eles são chamados na mesma ordem a cada renderização.

5

Realize um ou mais componentes para a plataforma Spring MVC, de forma a que um conjunto de recursos seja exposto no caminho `/status/{method}`. Um pedido de método GET para um desses recursos deve retornar uma mensagem com uma representação ao contendo um objeto JSON. Esse objeto deve representar o tempo máximo e mínimo do processamento de pedidos com método `method`. Caso o servidor ainda não tenha recebido pedidos com o método `method`, a resposta ao pedido deve ter status code 404

Componente Spring MVC em Kotlin para expor o conjunto de recursos no caminho `/status/{method}`. O objetivo é retornar um objeto JSON que represente o tempo máximo e mínimo de processamento de pedidos com o método especificado. Se o servidor ainda não tiver recebido pedidos com o método `method`, a resposta ao pedido deve ter o status code 404.

código de exemplo:

```
import org.springframework.http.HttpStatus
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("/status")
class StatusController {

    // Simulando tempos de processamento (em milissegundos)
    private val processingTimes = mapOf(
        "GET" to listOf(100, 200, 300), // Exemplo: [100, 200, 300]
        // Adicione outros métodos e tempos de processamento aqui
    )

    @GetMapping("/{method}")
    fun getStatus(@PathVariable method: String): ResponseEntity<Map<String, Int>>
    {
        val times = processingTimes[method.toUpperCase()] ?: return
        ResponseEntity.notFound().build()
        val minTime = times.minOrNull() ?: 0
        val maxTime = times.maxOrNull() ?: 0
        val result = mapOf("minTime" to minTime, "maxTime" to maxTime)
        return ResponseEntity(result, HttpStatus.OK)
    }
}
```

Neste exemplo:

- O endpoint `/status/{method}` responde a solicitações de método GET com o método especificado.
- Os tempos de processamento são simulados no mapa `processingTimes`.
- Se o método não estiver presente no mapa, a resposta terá o status code 404 (Not Found).

Quando você acessar `http://localhost:8080/status/GET`, receberá uma resposta JSON como esta:

```
{
  "minTime": 100,
  "maxTime": 300
}
```

Source: Conversation with Bing, 1/7/2024

- (1) Serve Static Resources with Spring | Baeldung. <https://www.baeldung.com/spring-mvc-static-resources>.
- (2) MockMvc Kotlin DSL | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/mockmvc-kotlin-dsl>.
- (3) Building web applications with Spring Boot and Kotlin. <https://spring.io/guides/tutorials/spring-boot-kotlin/>.
- (4) The State of Kotlin Support in Spring | The Kotlin Blog. <https://blog.jetbrains.com/kotlin/2020/08/the-state-of-kotlin-support-in-spring/>.
- (5) Spring Dependency Injection With Kotlin | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/spring-dependency-injection>.
- (6) github.com. <https://github.com/cx-cloud-101/speedtest-workshop/tree/d1e3e26d98114576bc913f618b251a0287f83742/gcp%2Flove-the-smell-of-http.md>.

6

Realize um componente React com as seguintes propriedades: uri do tipo string, e period do tipo number. O componente deve realizar um pedido HTTP de método GET ao URI definido por uri, com a periodicidade de period milisegundos. Após a conclusão de cada pedido, o componente deve apresentar o status code da resposta e o tempo que a resposta demorou a ser recebida. Em caso de erro, deve apresentar a descrição do erro. Caso o pedido não esteja concluído em period milisegundos, deve ser apresentada a mensagem "timeout" e o resultado do pedido deve ser ignorado. O componente realizado deve reagir correctamente a mudanças nas suas propriedades.

```
import React, { useState, useEffect } from 'react';

const HttpComponent = ({ uri, period }) => {
  const [response, setResponse] = useState(null);
  const [error, setError] = useState(null);

  useEffect(() => {
    let isMounted = true;

    const fetchData = async () => {
      try {
        const startTime = Date.now();
        const response = await fetch(uri);
        const endTime = Date.now();

        if (isMounted) {
          setResponse({
            status: response.status,
            time: endTime - startTime,
          });
        }
      } catch (err) {
        if (isMounted) {
          setError(err.message);
        }
      }
    }

    if (period > 0) {
      fetchData();
      setTimeout(fetchData, period);
    }
  }, [uri, period]);

  return (
    <div>
      {response ? `Status: ${response.status} | Time: ${response.time}ms` : ''}
      {error ? `Error: ${error}` : ''}
    </div>
  );
};
```

```

    const intervalId = setInterval(fetchData, period);

    return () => {
      clearInterval(intervalId);
      isMounted = false;
    };
  }, [uri, period]);

  if (error) {
    return <div>Error: {error}</div>;
  }

  if (!response) {
    return <div>Loading...</div>;
  }

  return (
    <div>
      <div>Status Code: {response.status}</div>
      <div>Response Time: {response.time} ms</div>
    </div>
  );
};

export default HttpComponent;

```

Usage example:

```

import React from 'react';
import HttpComponent from './HttpComponent'; // Import your component

const App = () => {
  return (
    <div>
      <h1>HTTP Component</h1>
      <HttpComponent uri="https://api.example.com/data" period={5000} />
    </div>
  );
};

export default App;

```

In this example:

- The `HttpComponent` makes periodic GET requests to the specified URI (`uri`) every `period` milliseconds.
- It displays the status code and response time for each request.
- If there's an error, it shows the error message.
- If a request doesn't complete within the specified period, it displays "timeout."

Remember to adjust the URI and customize error handling according to your specific use case. 😊

Source: Conversation with Bing, 1/7/2024

- (1) Where to make http request on react component - Stack Overflow. <https://stackoverflow.com/questions/39092004/where-to-make-http-request-on-react-component>.
- (2) How to send HTTP Requests in React | malcoded. <https://malcoded.com/posts/react-http-requests-axios/>.
- (3) node.js - How do I get http headers in React.js - Stack Overflow. <https://stackoverflow.com/questions/44354763/how-do-i-get-http-headers-in-react-js>.
- (4) React + Fetch - HTTP POST Request Examples | Jason Watmore's Blog. <https://jasonwatmore.com/post/2020/02/01/react-fetch-http-post-request-examples>.
- (5) GET HTTP Request in React - Stack Abuse. <https://stackabuse.com/get-http-request-in-react/>.
- (6) undefined. <https://reqres.in/api/posts>.

Época Especial, Semestre de Verão, 21/22

1.1

** No protocolo HTTP, um pedido de método GET para o recurso

<https://example.com/projects/123/delete>:

- i. Solicita a remoção do recurso com URI <https://example.com/projects/123>.
- ii. Solicita uma representação do recurso com URI <https://example.com/projects/123>.
- iii. Solicita uma representação do recurso com URI <https://example.com/projects/123/delete>.
- iv. Deve resultar sempre numa resposta com status code 400 (bad request)

iii

1.2

No protocolo HTTP, uma mensagem de resposta de status code 500:

- i. Não pode ter payload.**
- ii. Pode ter um payload usando um qualquer media type.**
- iii. Tem de ter um payload usando o media type application/problem+json.**
- iv. Tem de ter um payload usando o media type application/json**

ii

Explicação:

- O status code 500 (Internal Server Error) indica que ocorreu um erro no servidor ao processar a solicitação.
- A especificação HTTP não impõe restrições específicas sobre o payload (corpo) da resposta com status 500.
- Portanto, a resposta pode incluir um payload com qualquer tipo de mídia (media type), como `application/json`, `text/plain`, etc.

Em resumo, a opção ii é a correta porque não há restrições específicas para o payload em uma resposta com status 500. 😊

1.3

Na norma Web Linking (RFC 8288), o atributo rel de um link:

- i. Contém o URI do recurso destino do link.
- ii. Contém o URI do recurso contexto do link.
- iii. Contém o media-type usado na representação do recurso destino do link.
- iv. Contém o tipo da relação entre dois recursos, representado eventualmente na forma de um URI.

iv

1.4

** Na biblioteca Spring MVC e na configuração por omissão:

- i. E sempre criada uma nova instância dum interceptor por cada pedido.
- ii. Não são criadas instâncias de interceptors porque todos os métodos têm de ser estáticos.
- iii. Apenas são criadas novas instâncias de interceptors quando todas as instâncias existentes estiverem a ser usadas.
- iv. Nenhuma das anteriores.**

iii

Explicação:

- Na biblioteca Spring MVC, os interceptors são criados como beans gerenciados pelo contêiner Spring.
- Por padrão, o Spring cria uma única instância de cada interceptor e a reutiliza para todos os pedidos.
- Essa abordagem é eficiente e evita a criação desnecessária de instâncias.

Em resumo, a opção iii é a correta porque o Spring reutiliza as instâncias de interceptors existentes, criando novas apenas quando necessário. 😊

Source: Conversation with Bing, 1/7/2024

(1) Interceptors :: Spring Framework. <https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-config/interceptors.html>.

(2) Using Spring Interceptors in your MVC Webapp - DZone. <https://dzone.com/articles/using-spring-interceptors-your>.

(3) Using the Spring RestTemplate Interceptor | Baeldung. <https://www.baeldung.com/spring-rest-template-interceptor>.

(4) Introduction to Spring MVC HandlerInterceptor | Baeldung. <https://www.baeldung.com/spring-mvc-handlerinterceptor>.

1.5

A utilização de módulos NPM usando o sistema CommonJS em ficheiros Javascript destinados à execução em browsers requer:

- i. A instalação de extensões no browser.
- ii. A utilização da ferramenta webpack ou similar.
- iii. A utilização da biblioteca React ou similar.
- iv. A utilização da sintaxe JSX

ii

Explicação:

- O sistema CommonJS é uma especificação para módulos em JavaScript que permite a organização e reutilização de código.
- Para usar módulos NPM (que seguem o padrão CommonJS) em navegadores, é necessário empacotá-los em um único arquivo.
- Ferramentas como **Webpack**, **Browserify** ou **Parcel** podem ser usadas para criar um único arquivo JavaScript (bundle) que inclui todas as dependências.
- Essas ferramentas resolvem questões como carregamento assíncrono, dependências e transpilação de código moderno para navegadores mais antigos.

Em resumo, o uso de módulos NPM no sistema CommonJS em navegadores requer uma ferramenta de empacotamento como o Webpack para criar bundles que podem ser executados no navegador. 😊

Source: Conversation with Bing, 1/7/2024

(1) How can I 'require' CommonJS modules in the browser?. <https://stackoverflow.com/questions/7576001/how-can-i-require-commonjs-modules-in-the-browser>.

(2) Use CommonJS npm packages in the browser with Browserify. <https://tsmx.net/npm-packages-browser/>.

(3) node.js - Use npm modules in browser js - Stack Overflow. <https://stackoverflow.com/questions/76989552/use-npm-modules-in-browser-js>.

(4) common-js - npm. <https://www.npmjs.com/package/common-js>.

1.6

** O primeiro parâmetro da função createElement da biblioteca React é:

- Sempre uma string.
- Sempre um componente.
- Sempre uma instância de um componente.
- Nenhuma das anteriores**

iv

O primeiro parâmetro da função `createElement` na biblioteca **React** pode ser uma **string** ou um **componente**. Vamos entender por quê:

1. **String:** Quando você passa uma string como o primeiro argumento para `createElement`, está criando um elemento HTML ou um componente funcional. Por exemplo:

```
const myElement = React.createElement('div', { className: 'my-class' },  
  'Hello, World!');
```

Nesse caso, `'div'` é uma string que representa a tag HTML a ser criada.

2. **Componente:** Você também pode passar um componente como o primeiro argumento. Isso é comum quando você cria seus próprios componentes personalizados. Por exemplo:

```
function MyComponent() {  
  return <h1>Hello from MyComponent!</h1>;  
}  
  
const myElement = React.createElement(MyComponent, { prop1: 'value1' });
```

Aqui, `MyComponent` é um componente funcional que será renderizado.

3. **Instância de um componente:** A opção (iii) não é estritamente verdadeira. O primeiro argumento não precisa ser uma instância já criada de um componente. Em vez disso, ele representa o tipo de componente que você deseja criar. A instância real do componente é criada internamente pelo React.

Portanto, a resposta correta é **iv. Nenhuma das anteriores**. O primeiro parâmetro pode ser uma string ou um componente, mas não precisa ser uma instância específica. O React cuida da criação da instância do componente com base no tipo fornecido.

1: [Referência da API React - createElement](#)

2: [MDN Web Docs - Document.createElement\(\)](#)

Source: Conversation with Bing, 1/7/2024

(1) createElement – React - GitHub Pages. <https://react.dev/reference/react/createElement>.

(2) Document.createElement() - APIs da Web | MDN - MDN Web Docs. <https://bing.com/search?q=O+primeiro+parametro+da+funcao+createElement+da+biblioteca+React+e>.

(3) Document.createElement() - APIs da Web | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement>.

2

No desenho de APIs HTTP, quais as vantagens da utilização de métodos idempotentes?

Em APIs HTTP, **métodos idempotentes** oferecem várias vantagens importantes:

1. **Tolerância a falhas:** Quando os consumidores de API cometem erros e enviam solicitações duplicadas, métodos idempotentes garantem que essas solicitações não deixem o sistema instável. Isso é crucial para manter a integridade e confiabilidade do sistema².
2. **Consistência de dados:** Métodos idempotentes, como **GET** e **PUT**, podem ser executados várias vezes com segurança sem alterar o resultado. Isso é especialmente útil para operações de leitura (como **GET**) e atualizações (como **PUT**) em recursos, pois garante que os dados permaneçam consistentes³.
3. **Segurança:** A idempotência ajuda a evitar efeitos colaterais indesejados. Quando um método é idempotente, ele não deve ter nenhum efeito colateral, exceto para manter estatísticas. Isso simplifica o design e a previsibilidade das operações da API¹.

Portanto, a utilização de métodos idempotentes é uma prática recomendada para projetar APIs HTTP confiáveis e robustas.

Source: Conversation with Bing, 1/7/2024

(1) What Are Idempotent HTTP Methods? - DEV Community. <https://blog.savetchuk.com/what-are-idempotent-http-methods>.

(2) Idempotence – Definirtec. <https://definirtec.com/idempotence/>.

(3) Idempotente - Glossário do MDN Web Docs: Definições de termos <https://developer.mozilla.org/pt-BR/docs/Glossary/Idempotent>.

(4) O que é ser idempotente em REST? O debate continua. <https://www.infoq.com/br/news/2013/05/idempotent/>.

3

No contexto do desenvolvimento de Single Page Applications, qual o propósito do uso do método `preventDefault`, presente em objectos evento?

Segundo o professor na aula de dúvidas as razões são:

- queremos manter o mesmo documento e não trocar por outro
- não queremos submeter (POST), queremos apenas fazer um fetch à API

Resposta do Bing copilot:

Em **Single Page Applications (SPAs)**, o método `preventDefault()` é frequentemente usado para **cancelar o comportamento padrão** associado a um evento. Vamos explorar por quê:

1. Eventos em React:

- O React possui um sistema interno de eventos que é acionado sempre que uma ação específica é realizada pelo usuário. Por exemplo, um evento pode ser acionado quando você clica em um botão usando a propriedade `onClick`:

```
function LogButton() {  
  const handleClick = (event) => {  
    console.log("Olá, Mundo!");  
    console.log(event);  
  };  
  
  return <button onClick={handleClick}>Clique em mim</button>;  
}
```

- Quando você clica no botão acima, a variável `event` será registrada como um objeto `SyntheticBaseEvent` no console. Esses eventos sintéticos do React são essencialmente wrappers em torno dos eventos nativos do DOM, garantindo propriedades consistentes em diferentes navegadores.

2. Comportamento Padrão do DOM:

- Os eventos nativos do DOM têm comportamentos padrão associados a eles. Por exemplo, ao clicar em um botão de envio de formulário (`<input type="submit">`), o navegador normalmente atualiza a página para enviar os dados ao servidor.
- No entanto, em SPAs modernas, a validação de formulários muitas vezes ocorre no lado do cliente para economizar tempo e largura de banda. Nesse cenário, queremos executar nosso próprio comportamento personalizado.

3. Cancelando o Comportamento Padrão:

- Para evitar que o comportamento padrão do DOM seja acionado, usamos o método `preventDefault()`:

```
export default function App() {  
  const [name, setName] = useState("");  
  
  const handleSubmit = (event) => {
```

```

    event.preventDefault(); // Cancela o comportamento padrão do envio do
    formulário
    console.log(name);
    console.log("Obrigado por enviar o formulário!");
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Nome:
        <input
          type="text"
          value={name}
          onChange={(event) => setName(event.target.value)}
        />
      </label>
      <input type="submit" value="Enviar" />
    </form>
  );
}

```

- Agora, o comportamento padrão de atualização da página após o envio do formulário é cancelado, e o código dentro de `handleSubmit` será executado.

Portanto, o uso de `preventDefault()` permite que você controle o comportamento do evento e execute suas próprias ações personalizadas em SPAs.

Source: Conversation with Bing, 1/7/2024

(1) Understanding React's `preventDefault()` | sebastian. <https://sebastian.com/react-preventdefault/>.

(2) Event: `preventDefault()` method - Web APIs | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault>.

(3) `preventDefault()` Event Method - W3Schools. https://www.w3schools.com/jsref/event_preventdefault.asp.

(4) How to prevent default behavior in an event callback in React JS. <https://www.geeksforgeeks.org/how-to-prevent-default-behavior-in-an-event-callback-in-reactjs/>.

4

No contexto da biblioteca React, o recurso identificado por <https://reactjs.org/docs/hooks-rules.html> (acedido a 2022-09-09) tem presente a seguinte afirmação na sua representação

Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns.

Indique qual a razão para esta limitação.

A razão para esta limitação é que os Hooks dependem da ordem em que são chamados para funcionar corretamente. Se os Hooks forem chamados dentro de loops, condições ou funções aninhadas, a ordem pode variar entre diferentes renderizações, o que pode causar bugs e inconsistências. Ao usar os Hooks no nível superior da função React, garante-se que eles são chamados na mesma ordem a cada renderização.

5

Realize um ou mais componentes para a plataforma Spring MVC, de forma a que um recurso seja exposto no caminho `/failures`. Um pedido de método GET para esse recurso deve retornar uma mensagem com uma representação contendo um objeto JSON. Esse objecto deve representar os últimos 10 pedidos que resultaram numa resposta com status code maior ou igual a 500. A representação de cada pedido deve incluir: o método do pedido, o URI do pedido, o status code da resposta, e o nome do controlador e do método responsável pelo processamento desse pedido (caso o processamento tenha sido realizado por um controlador).

Para realizar essa tarefa, podemos usar os seguintes componentes:

- Um **filtro** que intercepta todas as requisições e registra as informações relevantes em um **repositório**. O filtro também verifica se a resposta tem um status code maior ou igual a 500 e, nesse caso, adiciona o nome do controlador e do método responsável pelo processamento ao registro.
- Um **controlador** que expõe o recurso `/failures` e retorna um objeto JSON com os últimos 10 registros do repositório.

A seguir, apresentamos um possível código em Kotlin para esses componentes, usando a biblioteca **Spring Boot**¹ e o formato **JSON**²:

```
// Um data class que representa um registro de uma requisição e uma resposta
data class Failure(
    val method: String, // o método do pedido
    val uri: String, // o URI do pedido
    val status: Int, // o status code da resposta
    var controller: String? = null, // o nome do controlador responsável pelo
processamento
    var handler: String? = null // o nome do método responsável pelo
processamento
)

// Um repositório que armazena os registros em uma lista
@Component
class FailureRepository {
    private val failures = mutableListOf<Failure>()

    // Adiciona um novo registro à lista
    fun addFailure(failure: Failure) {
        failures.add(failure)
    }

    // Retorna os últimos 10 registros da lista
    fun getLastFailures(): List<Failure> {
        return failures.takeLast(10)
    }
}

// Um filtro que intercepta todas as requisições e registra as informações
relevantes
@Component
class FailureFilter(
```

```

    @Autowired val failureRepository: FailureRepository // injeta o repositório
) : Filter {
    override fun doFilter(
        request: ServletRequest,
        response: ServletResponse,
        chain: FilterChain
    ) {
        // Converte os objetos de requisição e resposta para HttpServletRequest e
        // HttpServletResponse
        val httpRequest = request as HttpServletRequest
        val httpResponse = response as HttpServletResponse

        // Cria um novo registro com as informações da requisição e da resposta
        val failure = Failure(
            method = httpRequest.method,
            uri = httpRequest.requestURI,
            status = httpResponse.status
        )

        // Adiciona o registro ao repositório
        failureRepository.addFailure(failure)

        // Verifica se a resposta tem um status code maior ou igual a 500
        if (httpResponse.status >= 500) {
            // Obtém o objeto HandlerExecutionChain que contém o controlador e o
            // método responsável pelo processamento
            val handler = (request.getAttribute(HandlerMapping.HANDLER_ATTRIBUTE)
                as HandlerExecutionChain).handler

            // Verifica se o handler é um HandlerMethod
            if (handler is HandlerMethod) {
                // Adiciona o nome do controlador e do método ao registro
                failure.controller = handler.beanType.simpleName
                failure.handler = handler.method.name
            }
        }

        // Continua o processamento da requisição
        chain.doFilter(request, response)
    }
}

// Um controlador que expõe o recurso /failures
@RestController
class FailureController(
    @Autowired val failureRepository: FailureRepository // injeta o repositório
) {
    // Define um método GET para o caminho /failures
    @GetMapping("/failures")
    fun getFailures(): ResponseEntity<List<Failure>> {
        // Obtém os últimos 10 registros do repositório
        val failures = failureRepository.getLastFailures()

        // Retorna uma resposta com o objeto JSON dos registros
        return ResponseEntity.ok(failures)
    }
}

```



```
}
```

Esse código é apenas um exemplo e pode ser adaptado conforme as necessidades do projeto. Espero que tenha sido útil. 😊

Source: Conversation with Bing, 1/7/2024

- (1) Spring MVC Setup with Kotlin | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/spring-mvc-kotlin>.
- (2) Kotlin :: Spring Framework. <https://docs.spring.io/spring-framework/reference/languages/kotlin.html>.
- (3) spring-mvc · GitHub Topics · GitHub. <https://github.com/topics/spring-mvc?l=kotlin>.
- (4) GitHub - cdimascio/kotlin-spring-mvc-template: 12-factor compliant <https://github.com/cdimascio/kotlin-spring-mvc-template>.
- (5) Kotlin + Spring Boot MVC Starter - GitHub. <https://github.com/waterlink/kotlin-spring-boot-mvc-starter>.
- (6) undefined. <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-kotlin.html>.

6

Realize um componente para uso com a biblioteca React que receba uma lista de URLs através das suas propriedades e apresente uma lista de items. Cada item está associado a um URL da lista e deve conter:

- O URL e o texto "...", enquanto o pedido ao URL estiver a ser realizado
- O URL e o código de estado da resposta, ou o erro ocorrido, depois do pedido ter sido concluído, com ou sem sucesso.

A ordem dos items deve respeitar a ordem dos URLs. A sua solução pode criar e usar componentes auxiliares. O componente realizado deve reagir correctamente a mudanças nas suas propriedades.

Para realizar esse componente, podemos seguir os seguintes passos:

1. Criar um componente auxiliar chamado `URLItem` que recebe um URL como propriedade e faz uma requisição HTTP usando a API `fetch`¹. Esse componente deve renderizar o URL e o texto "..." enquanto a requisição estiver em andamento, e depois mostrar o código de status da resposta ou o erro ocorrido.
2. Criar um componente principal chamado `URLList` que recebe uma lista de URLs como propriedade e usa o método `map` para renderizar cada URL como um `URLItem`. Esse componente deve usar o hook `useEffect` para detectar mudanças na lista de URLs e atualizar a renderização.
3. Usar o componente `URLList` em um componente pai, como o `App`, e passar a lista de URLs desejada como propriedade.

A seguir, apresentamos um possível código em JavaScript para esses componentes, usando a biblioteca **React**² e o formato **JSX**³:

```
// Importa o React e o useState e useEffect hooks
import React, { useState, useEffect } from "react";

// Cria o componente URLItem que recebe um URL como propriedade
function URLItem({ url }) {
```

```

// Cria um estado para armazenar o status da requisição
const [status, setStatus] = useState("");

// Cria um efeito que é executado quando o componente é montado
useEffect(() => {
  // Faz uma requisição HTTP usando o fetch
  fetch(url)
    .then((response) => {
      // Se a requisição for bem sucedida, atualiza o status com o código da
      resposta
      setStatus(response.status);
    })
    .catch((error) => {
      // Se a requisição falhar, atualiza o status com o erro
      setStatus(error.message);
    });
}, [url]); // Adiciona o url como dependência do efeito

// Retorna o elemento JSX que renderiza o url e o status
return (
  <li>
    {url}: {status}
  </li>
);
}

// Cria o componente URLList que recebe uma lista de URLs como propriedade
function URLList({ urls }) {
  // Retorna o elemento JSX que renderiza uma lista de URLItems
  return (
    <ul>
      {urls.map((url) => (
        <URLItem key={url} url={url} />
      ))}
    </ul>
  );
}

// Cria o componente App que usa o URLList
function App() {
  // Cria uma lista de URLs de exemplo
  const urls = [
    "https://reactjs.org/",
    "https://pt.wikipedia.org/wiki/React_(JavaScript)",
    "https://invalid.url/",
  ];

  // Retorna o elemento JSX que renderiza o URLList com a lista de URLs
  return <URLList urls={urls} />;
}

// Exporta o componente App
export default App;

```

Esse código é apenas um exemplo e pode ser adaptado conforme as necessidades do projeto. Espero que tenha sido útil. 😊

1: [MDN Web Docs - fetch\(\)](#)

2: [React - Uma biblioteca JavaScript para criar interfaces de usuário](#)

3: [MDN Web Docs - JSX](#)

Source: Conversation with Bing, 1/7/2024

(1) React List component - Material UI. <https://mui.com/material-ui/react-list/>.

(2) How to Use URLs for State Management in React - freeCodeCamp.org. <https://www.freecodecamp.org/news/how-to-use-urls-for-state-management-in-react/>.

(3) React Native List Views – Introdução e Exemplo - DEV Community. <https://dev.to/alexandreifeire/react-native-list-views-introducao-e-exemplo-14d7>.

Teste Final Época Normal, Sermestre de Inverno, 22/23

1.1

No protocolo HTTP, o método DELETE é:

- i. Não safe e não idempotente.
- ii. Safe e não idempotente.
- iii. Não Safe e idempotente.
- iv. Safe e idempotente.

A resposta correta é **iii. Não Safe e idempotente**. Vamos explicar o que isso significa:

- **Não Safe:** Um método de requisição HTTP é considerado **safe** se ele não altera o estado do servidor¹. Ou seja, ele apenas lê dados, mas não os modifica. O método **DELETE**, por outro lado, é usado para **remover** um recurso específico do servidor². Portanto, ele altera o estado do servidor e não é safe.
- **Idempotente:** Um método de requisição HTTP é considerado **idempotente** se o resultado de aplicá-lo várias vezes é o mesmo que aplicá-lo apenas uma vez¹. Ou seja, ele não causa efeitos colaterais adicionais. O método **DELETE** é idempotente, pois remover um recurso uma ou mais vezes tem o mesmo efeito: o recurso é removido. Se o recurso já não existir, o servidor deve retornar um código de status **404 (Not Found)**².

Portanto, o método **DELETE** é não safe e idempotente, pois ele remove um recurso do servidor e pode ser aplicado várias vezes sem causar efeitos colaterais adicionais.

1: [MDN Web Docs - Métodos de requisição HTTP](#)

2: [MDN Web Docs - DELETE](#)

Source: Conversation with Bing, 1/7/2024

(1) DELETE - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/DELETE>.

(2) Métodos de requisição HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>.

(3) Métodos de requisição HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>.

(4) DELETE - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/DELETE>.

(5) Método Delete HTTP - Informatica. https://docs.informatica.com/pt_pt/data-engineering/s

hared-content-for-data-engineering/10-2-2-service-pack-1/guia-de-servicos-da-web/transfor-macao-de-consumidor-de-servico-da-web-rest/metodos-http/metodo-delete-http.html.

(6) DELETE - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTTP/Methods/DELETE>.

(7) Idempotente - Glossário do MDN Web Docs: Definições de termos <https://developer.mozilla.org/pt-BR/docs/Glossary/Idempotent>.

1.2

No protocolo HTTP, o conceito de interface uniforme significa que:

- i. Os URIs tem de ter sempre a mesma estrutura, tal como usar nomes pluralizados (e.g. /games/123 em vez de /game/123) e ter sempre os identificadores da base de dados no caminho e não na query string (e.g. /games/123 em vez de /games?id=123).
- ii. O significado do status codes nas mensagens de resposta não depende do recurso acedido.
- iii. Devem ser usados media types baseados em JSON em todas as representações (e.g. application/json e application/problem+json).
- iv. Não podem ser usados métodos não definidos no RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1):

Semantics and Content

A resposta correta é **ii. O significado do status codes nas mensagens de resposta não depende do recurso acedido**. Vamos explicar o que isso significa e por que as outras opções estão erradas:

- **Interface uniforme:** O conceito de interface uniforme é um dos princípios do estilo arquitetural REST (Representational State Transfer), que é usado para projetar serviços web baseados em HTTP¹. Uma interface uniforme significa que os clientes e servidores se comunicam usando um conjunto padrão de regras e formatos, independentemente dos recursos específicos envolvidos. Isso simplifica e desacopla a arquitetura, permitindo que cada parte evolua independentemente².
- **i. Os URIs tem de ter sempre a mesma estrutura...:** Essa opção está errada, pois o formato dos URIs (Uniform Resource Identifiers) não é uma parte essencial da interface uniforme. Embora seja uma boa prática usar URIs consistentes e intuitivos, eles não precisam seguir uma estrutura rígida, desde que identifiquem os recursos de forma única e clara³.
- **ii. O significado do status codes nas mensagens de resposta não depende do recurso acedido:** Essa opção está correta, pois faz parte da interface uniforme garantir que as mensagens sejam auto-descritivas. Isso significa que cada mensagem deve incluir informações suficientes para descrever como processá-la. Por exemplo, o código de status da resposta deve indicar o resultado da requisição, independentemente do recurso solicitado. Os códigos de status são padronizados pelo HTTP e têm o mesmo significado em qualquer contexto.
- **iii. Devem ser usados media types baseados em JSON em todas as representações...:** Essa opção está errada, pois a interface uniforme não impõe o uso de um media type específico para as representações dos recursos. O media type é uma informação que descreve o formato e o tipo de dados de um recurso, como por exemplo `application/json`

ou `text/html`. A interface uniforme permite que os clientes e servidores negociem o media type mais adequado para cada recurso, usando os cabeçalhos `Accept` e `Content-Type`.

- **iv. Não podem ser usados métodos não definidos no RFC 7231....** Essa opção está errada, pois a interface uniforme não proíbe o uso de métodos não padronizados pelo HTTP. Os métodos são verbos que indicam a ação a ser realizada sobre um recurso, como por exemplo `GET` ou `POST`. Embora seja recomendado usar os métodos definidos pelo HTTP, é possível definir métodos personalizados para casos específicos, desde que eles sejam documentados e compatíveis com as propriedades dos métodos HTTP, como segurança e idempotência.

Portanto, a interface uniforme significa que os clientes e servidores usam um conjunto padrão de regras e formatos para se comunicar, independentemente dos recursos específicos envolvidos. Isso implica que o significado dos códigos de status nas mensagens de resposta não depende do recurso acedido, mas não implica nas outras opções apresentadas.

1: [MDN Web Docs - REST](#)

2: [MDN Web Docs - Uma visão geral do HTTP](#)

3: [MDN Web Docs - Identificando recursos na web](#)

: [MDN Web Docs - Códigos de status de respostas HTTP](#)

: [MDN Web Docs - Tipos de mídia HTTP](#)

: [MDN Web Docs - Métodos de requisição HTTP](#)

Source: Conversation with Bing, 1/7/2024

(1) REST - What exactly is meant by Uniform Interface?. <https://stackoverflow.com/questions/25172600/rest-what-exactly-is-meant-by-uniform-interface>.

(2) Uma visão geral do HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>.

(3) Identificando recursos na web - HTTP | MDN - MDN Web Docs. https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web.

(4) Uma visão geral do HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>.

(5) REST - What exactly is meant by Uniform Interface?. <https://stackoverflow.com/questions/25172600/rest-what-exactly-is-meant-by-uniform-interface>.

(6) Identificando recursos na web - HTTP | MDN - MDN Web Docs. https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web.

1.3

No contexto de APIs HTTP, qual a forma que sugere para comunicar informação de erro específica da aplicação, em resposta a pedidos HTTP.

- i. Definindo e usando um novo status code nas mensagens de resposta.**
- ii. Colocando essa informação no payload da mensagem de resposta.**
- iii. Definindo e usando um novo header nas mensagens de resposta.**
- iv. Retornando uma exceção, criada especificamente para representar esse tipo de informação.**

A forma que eu sugiro para comunicar informação de erro específica da aplicação, em resposta a pedidos HTTP, é a opção **ii. Colocando essa informação no payload da mensagem de resposta**. Vamos explicar o porquê e por que as outras opções estão erradas:

- **Payload da mensagem de resposta:** O payload é o corpo da mensagem de resposta, que contém os dados do recurso solicitado ou uma representação do resultado da operação. Ao colocar a informação de erro específica da aplicação no payload, podemos usar um formato padronizado e estruturado, como por exemplo o JSON¹, para descrever o tipo, a causa e os detalhes do erro. Isso facilita o entendimento e o tratamento do erro pelo cliente, bem como a depuração e a correção do problema pelo desenvolvedor. Um exemplo de payload de erro em JSON é o seguinte:

```
{
  "type": "https://example.com/probs/out-of-credit",
  "title": "You do not have enough credit.",
  "detail": "Your current balance is 30, but that costs 50.",
  "instance": "/account/12345/msgs/abc",
  "balance": 30,
  "accounts": ["/account/12345", "/account/67890"]
}
```

- **i. Definindo e usando um novo status code nas mensagens de resposta:** Essa opção está errada, pois não é recomendado definir e usar novos códigos de status para comunicar erros específicos da aplicação. Os códigos de status são padronizados pelo HTTP e têm o mesmo significado em qualquer contexto². Eles indicam o resultado geral da requisição, mas não os detalhes do erro. Além disso, definir novos códigos de status pode causar incompatibilidade e confusão com os clientes e intermediários que não reconhecem esses códigos.
- **iii. Definindo e usando um novo header nas mensagens de resposta:** Essa opção está errada, pois não é aconselhável definir e usar novos cabeçalhos para comunicar erros específicos da aplicação. Os cabeçalhos são usados para transmitir informações adicionais sobre a requisição ou a resposta, mas não sobre o recurso ou o erro em si. Além disso, definir novos cabeçalhos pode causar problemas de interoperabilidade e segurança com os clientes e intermediários que não esperam esses cabeçalhos.
- **iv. Retornando uma exceção, criada especificamente para representar esse tipo de informação:** Essa opção está errada, pois não é possível retornar uma exceção em uma resposta HTTP. As exceções são objetos que representam erros ou condições anormais que ocorrem durante a execução de um programa. Elas são usadas para interromper o fluxo normal do programa e transferir o controle para um manipulador de exceções. As exceções são específicas da linguagem de programação e do ambiente de execução, e não podem ser transmitidas em uma resposta HTTP, que é um protocolo de comunicação independente de plataforma.

Portanto, a forma que eu sugiro para comunicar informação de erro específica da aplicação, em resposta a pedidos HTTP, é colocar essa informação no payload da mensagem de resposta, usando um formato padronizado e estruturado, como o JSON.

¹: [MDN Web Docs - JSON](#)

²: [MDN Web Docs - Códigos de status de respostas HTTP](#)

³: [MDN Web Docs - JSON](#)

Source: Conversation with Bing, 1/7/2024

(1) Como melhorar a comunicação de erros em APIs HTTP — Parte 1. <https://medium.com/gbtech/como-melhorar-a-comunica%C3%A7%C3%A3o-de-erros-em-apis-http-parte-1-4323fb694e41>.

(2) Como lidar com erros em APIs Web do ASP.NET Core. <https://learn.microsoft.com/pt-br/as>

pnet/core/web-api/handle-errors?view=aspnetcore-8.0.

(3) Códigos de status de respostas HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>.

(4) Como lidar com erros em APIs Web do ASP.NET Core. <https://learn.microsoft.com/pt-br/aspnet/core/web-api/handle-errors?view=aspnetcore-8.0>.

(5) Códigos de status de respostas HTTP - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>.

(6) undefined. <http://www.w3.org/1999/xhtml>.

(7) github.com. <https://github.com/zhiyi7/zhiyi7.github.io/tree/801f754c10069c8d0d1a5b30f09935682fc4a0b3/posts%2F2019-01-11-rfc7807-api-problem-detail.md>.

(8) github.com. <https://github.com/PDMLab/http-problem-details/tree/ed79e6911b37f62e42b3acd1c9e4ec1bfaf7606b/README.md>.

1.4

Para se suportar deep-linking no contexto de uma single page application é necessário configurar o servidor

que serve essa aplicação da seguinte forma.

- i. Caso o caminho presente num pedido GET não esteja associado a um ficheiro, então é retornada uma resposta de sucesso com o conteúdo de index.html em vez de uma resposta com status code 404.**
- ii. Caso o caminho presente num pedido GET não esteja associado a um ficheiro, então é retornada uma resposta de redireção para o caminho /index.html.**
- iii. O servidor usar a API de história para navegar para o deep-link.**
- iv. O servidor usar React Router para navegar para o deep-link.**

A resposta correta é a opção **i. Caso o caminho presente num pedido GET não esteja associado a um ficheiro, então é retornada uma resposta de sucesso com o conteúdo de index.html em vez de uma resposta com status code 404.** Vamos explicar o que isso significa e por que as outras opções estão erradas:

- **Deep-linking:** O conceito de deep-linking se refere à capacidade de acessar uma parte específica de uma aplicação web usando um URL¹. Por exemplo, se uma aplicação web tem uma página que mostra os detalhes de um produto, um deep-link seria um URL que leva diretamente a essa página, como por exemplo `[https://www.example.com/products/123]` (¹).
- **Single Page Application (SPA):** Uma SPA é uma aplicação web que funciona dentro de uma única página, sem recarregar a página inteira quando o usuário navega entre diferentes seções². Uma SPA usa JavaScript para manipular o conteúdo da página dinamicamente, criando uma experiência mais fluida e interativa para o usuário.
- **Servidor que serve a SPA:** Para suportar deep-linking no contexto de uma SPA, é necessário configurar o servidor que serve a aplicação de uma forma especial. Isso porque, quando o usuário digita um URL no navegador ou clica em um link externo, o navegador faz uma requisição HTTP ao servidor para obter o conteúdo da página correspondente ao URL. No entanto, em uma SPA, o conteúdo da página é gerado dinamicamente pelo JavaScript, e não pelo servidor. Portanto, se o servidor não encontrar um ficheiro associado ao caminho do URL, ele retornaria uma resposta com status code 404 (Not Found), o que não é o que queremos. Em vez disso, queremos que o servidor retorne o conteúdo de index.html, que é o

ficheiro que contém o código JavaScript da SPA. Assim, o JavaScript pode analisar o URL e renderizar o conteúdo apropriado na página. Essa é a opção i, que é a correta.

- **ii. Caso o caminho presente num pedido GET não esteja associado a um ficheiro, então é retornada uma resposta de redireção para o caminho /index.html:** Essa opção está errada, pois uma resposta de redireção mudaria o URL no navegador para /index.html, perdendo o caminho original que continha a informação do deep-link. Além disso, uma resposta de redireção causaria um recarregamento desnecessário da página, prejudicando a performance e a experiência do usuário.
- **iii. O servidor usar a API de história para navegar para o deep-link:** Essa opção está errada, pois a API de história é uma funcionalidade do navegador, e não do servidor. A API de história permite que o JavaScript manipule o histórico de navegação do navegador, adicionando, removendo ou modificando entradas. Isso é útil para uma SPA, pois permite que o JavaScript mude o URL no navegador sem recarregar a página, criando uma ilusão de navegação entre diferentes páginas³. No entanto, isso não resolve o problema de como o servidor deve responder a uma requisição HTTP de um URL com um deep-link.
- **iv. O servidor usar React Router para navegar para o deep-link:** Essa opção está errada, pois o React Router é uma biblioteca JavaScript, e não uma funcionalidade do servidor. O React Router é uma biblioteca que facilita a criação de rotas em uma SPA feita com React, que é uma biblioteca JavaScript para construir interfaces de usuário. O React Router permite que o JavaScript renderize diferentes componentes React na página de acordo com o URL no navegador, usando a API de história para mudar o URL sem recarregar a página⁴. No entanto, isso não resolve o problema de como o servidor deve responder a uma requisição HTTP de um URL com um deep-link.

Portanto, a forma de suportar deep-linking no contexto de uma SPA é configurar o servidor para retornar o conteúdo de index.html em vez de uma resposta com status code 404, caso o caminho presente num pedido GET não esteja associado a um ficheiro. Isso permite que o JavaScript da SPA analise o URL e renderize o conteúdo apropriado na página.

1: [MDN Web Docs - Deep linking](#)

2: [MDN Web Docs - Single-page applications](#)

3: [MDN Web Docs - Manipulando o histórico do navegador](#)

4: [React Router - Declarative Routing for React.js](#)

Source: Conversation with Bing, 1/7/2024

(1) single page application deep linking with login page. <https://stackoverflow.com/questions/17356919/single-page-application-deep-linking-with-login-page>.

(2) single page application deep linking with login page. <https://stackoverflow.com/questions/17356919/single-page-application-deep-linking-with-login-page>.

(3) Single Page Applications. <https://web.stanford.edu/class/cs142/lectures/SPA.pdf>.

(4) What is deep linking? How to create a deep link? - Storyly. <https://www.storyly.io/glossary/deep-linking>.

(5) Respecting Browser Navigation in Single Page Applications. <https://rakhman.info/blog/respecting-browser-navigation-spa/>.

(6) What is deep linking? How to create a deep link? - Storyly. <https://www.storyly.io/glossary/deep-linking>.

(7) Single Page Applications. <https://web.stanford.edu/class/cs142/lectures/SPA.pdf>.

(8) What is Deep linking? - Singular. <https://www.singular.net/glossary/deep-linking/>.

(9) undefined. <http://example.com>.

(10) undefined. <http://example.com?id=3535>.

(11) undefined. <http://www.example.org/dirmod?sid=789AB8&type=gen&mod=Core+Pag>.

(12) undefined. <http://www.example.org/show/A6CD4967199>.

(13) undefined. <https://www.flickr.com/photos/jarnasen/24593000826/in/explore-2016-01-26/>.

(14) undefined. <https://v5.reactrouter.com/>.

(15) undefined. <https://www.myapp.com/path/to/page?pageid=1>.

1.5

A avaliação da expressão JSX `<A><p>Hello</p>` é equivalente qual das seguintes expressões:

- i. `A({children=[document.createElement("p"), "hello"]})`
- ii. `React.createElement("A", null, React.createElement("p", null, "Hello"))`
- iii. `React.createElement(A, null, document.createElement("p", null, "Hello"))`
- iv. `React.createElement(A, null, React.createElement("p", null, "Hello"))`

A resposta correta é a opção **iv. `React.createElement(A, null, React.createElement("p", null, "Hello"))`**. Vamos explicar o que isso significa e por que as outras opções estão erradas:

- **JSX:** JSX é uma sintaxe de extensão JavaScript usada no React para escrever facilmente HTML e JavaScript juntos¹. Por exemplo, o código JSX a seguir cria um elemento `<h1>` com o texto "Olá, Mundo!":

```
const element = <h1>Olá, Mundo!</h1>;
```

- **React.createElement:** Para que o navegador entenda o código JSX, ele precisa ser transformado em JavaScript válido. O React faz isso usando uma função chamada `React.createElement`, que recebe como argumentos o tipo do elemento, as propriedades do elemento e os filhos do elemento. O código JSX acima é equivalente ao seguinte código JavaScript:

```
const element = React.createElement(  
  "h1",  
  null,  
  "Olá, Mundo!"  
);
```

- **Avaliação da expressão JSX:** A expressão JSX dada no enunciado é a seguinte:

```
<A><p>Hello</p></A>
```

Essa expressão cria um elemento do tipo `A`, que pode ser um componente personalizado ou uma tag HTML, com um filho do tipo `p`, que é uma tag HTML, com o texto "Hello". Para transformar essa expressão em JavaScript, usamos a função `React.createElement` duas vezes: uma para o elemento `A` e outra para o elemento `p`. O resultado é o seguinte código JavaScript:

```
React.createElement(  
  A,  
  null,  
  React.createElement(  
    "p",  
    null,  
    "Hello"  
  )  
);
```

Esse é o código da opção iv, que é a correta.

- **i. `A({children:[document.createElement("p"), "hello"]})`:** Essa opção está errada, pois não usa a função `React.createElement` para criar os elementos. Além disso, ela passa um array com dois elementos como filhos de `A`: um elemento `p` criado pelo método `document.createElement`, que é diferente do método `React.createElement`, e uma string "hello", que deveria estar dentro do elemento `p`.
- **ii. `React.createElement("A", null, React.createElement("p", null, "Hello"))`:** Essa opção está errada, pois usa uma string "A" como o tipo do elemento, em vez de uma variável `A`. Isso implica que o elemento é uma tag HTML, mas não existe uma tag HTML chamada `A`. Se `A` for um componente personalizado, ele deve ser passado sem aspas, como na opção iv.
- **iii. `React.createElement(A, null, document.createElement("p", null, "Hello"))`:** Essa opção está errada, pois usa o método `document.createElement` para criar o elemento `p`, em vez do método `React.createElement`. Isso cria um elemento do DOM nativo, e não um elemento do React. Além disso, o método `document.createElement` não aceita um segundo argumento com o texto do elemento, como na opção iv.

Portanto, a avaliação da expressão JSX `<A><p>Hello</p>` é equivalente à expressão JavaScript `React.createElement(A, null, React.createElement("p", null, "Hello"))`, que é a opção iv.

1: [React - O que é JSX](#)

Source: Conversation with Bing, 1/7/2024

(1) javascript - ReactJS - JS vs JSX - Stack Overflow. <https://stackoverflow.com/questions/46169472/reactjs-js-vs-jsx>.

(2) React - O que é JSX - DEV Community. <https://dev.to/nascimento/react-o-que-e-jsx-3032>.

(3) TypeScript: Documentation - JSX. <https://www.typescriptlang.org/pt/docs/handbook/jsx.html>.

1.6

No contexto da biblioteca React, uma função hook.

- i. Pode ser chamada dentro de um callback associado a um evento.
- ii. Pode ser chamada dentro da função passada ao `useEffect`.
- iii. Pode ser chamada dentro de outra função hook.
- iv. Nenhuma das anteriores.

A resposta correta é a opção **iii. Pode ser chamada dentro de outra função hook**. Vamos explicar o que isso significa e por que as outras opções estão erradas:

- **Função hook:** Uma função hook é uma função especial que permite que você use o estado e outros recursos do React em componentes funcionais¹. Por exemplo, a função `useState` é um hook que permite que você adicione e atualize o estado de um componente funcional. Outros exemplos de hooks são `useEffect`, `useContext`, `useRef` e `useReducer`.
- **Regras dos hooks:** Para usar os hooks de forma correta e consistente, é preciso seguir duas regras básicas²:
 - Apenas chame hooks no nível mais alto. Não chame hooks dentro de loops, condições ou funções aninhadas.
 - Apenas chame hooks de componentes funcionais ou de outros hooks personalizados.
- **i. Pode ser chamada dentro de um callback associado a um evento:** Essa opção está errada, pois viola a primeira regra dos hooks. Um callback associado a um evento é uma função aninhada que é executada quando um evento específico ocorre, como por exemplo um clique em um botão. Se você chamar um hook dentro de um callback, você estará quebrando a ordem em que os hooks são chamados, o que pode causar bugs e inconsistências.
- **ii. Pode ser chamada dentro da função passada ao useEffect:** Essa opção está errada, pois também viola a primeira regra dos hooks. A função passada ao `useEffect` é uma função aninhada que é executada após a renderização do componente, ou quando alguma dependência mudar. Se você chamar um hook dentro dessa função, você estará quebrando a ordem em que os hooks são chamados, o que pode causar bugs e inconsistências.
- **iii. Pode ser chamada dentro de outra função hook:** Essa opção está correta, pois respeita a segunda regra dos hooks. Você pode chamar um hook dentro de outro hook, desde que esse outro hook seja um componente funcional ou um hook personalizado. Um hook personalizado é uma função que usa outros hooks para criar um comportamento reutilizável. Por exemplo, o código a seguir cria um hook personalizado chamado `useCounter`, que usa o hook `useState` para criar um contador simples:

```
function useCounter(initialValue) {  
  const [count, setCount] = useState(initialValue);  
  
  function increment() {  
    setCount(count + 1);  
  }  
  
  function decrement() {  
    setCount(count - 1);  
  }  
  
  function reset() {  
    setCount(initialValue);  
  }  
  
  return [count, increment, decrement, reset];  
}
```

Esse hook personalizado pode ser usado em qualquer componente funcional que precise de um contador, como por exemplo:

```
function Counter() {
  const [count, increment, decrement, reset] = useCounter(0);

  return (
    <div>
      <p>Contagem: {count}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Resetar</button>
    </div>
  );
}
```

- **iv. Nenhuma das anteriores:** Essa opção está errada, pois existe pelo menos uma situação em que você pode chamar um hook, que é dentro de outra função hook, como na opção iii.

Portanto, a resposta correta é a opção iii. Pode ser chamada dentro de outra função hook, pois isso respeita as regras dos hooks e permite criar comportamentos reutilizáveis.

1: [React - Hooks](#)

2: [React - Regras dos Hooks](#)

Source: Conversation with Bing, 1/7/2024

(1) React Hooks: o que é e como funcionam? | Alura. <https://www.alura.com.br/artigos/react-hooks>.

(2) Introdução aos Hooks – React. <https://pt-br.legacy.reactjs.org/docs/hooks-intro.html>.

(3) React Hooks: o que é e como funcionam? | Alura. <https://www.alura.com.br/artigos/react-hooks>.

(4) Introdução aos Hooks – React. <https://pt-br.legacy.reactjs.org/docs/hooks-intro.html>.

(5) Iniciando Com o Hook useState no React - Kinsta®. <https://bing.com/search?q=fun%c3%a7%c3%a3o+hook+React>.

(6) Iniciando Com o Hook useState no React - Kinsta®. <https://kinsta.com/pt/base-de-conhecimento/usestate-react/>.

(7) ReactJS Hooks - GeeksforGeeks. <https://www.geeksforgeeks.org/reactjs-hooks/>.

(8) React Hooks - W3Schools. https://www.w3schools.com/react/react_hooks.asp.

(9) undefined. <https://api.github.com/users/julio-cesar96/repos>.

2

No âmbito da biblioteca Spring, indique qual a consequência de anotar classes com a anotação @Component.

A anotação @Component é uma forma de indicar ao Spring que uma classe é um bean gerenciado pelo container de inversão de controle (IoC) do framework. Isso significa que, sem ter que escrever nenhum código explícito, o Spring irá:

- Escanear a aplicação em busca de classes anotadas com @Component
- Instanciar essas classes e injetar as dependências especificadas nelas
- Injetar essas classes onde forem necessárias

A anotação @Component é uma das chamadas anotações estereotipadas do Spring, que servem para marcar as classes como candidatas à detecção automática e configuração baseada em anotações. Outras anotações estereotipadas são @Repository, @Service e @Controller, que são especializações de @Component com usos e significados específicos fora do contexto de auto-

detecção ou injeção de dependência¹².

Source: Conversation with Bing, 1/7/2024

(1) Spring @Component Annotation | Baeldung. <https://www.baeldung.com/spring-component-annotation>.

(2) O que faz a anotação @Component? - Stack Overflow em Português. <https://pt.stackoverflow.com/questions/117109/o-que-faz-a-anota%C3%A7%C3%A3o-component>.

(3) O que faz a anotação @Component? - Stack Overflow em Português. <https://bing.com/search?q=anota%C3%A7%C3%A3o+%40Component+Spring>.

(4) Spring @Component Annotation Example - Java Guides. <https://www.javaguides.net/2018/11/spring-component-annotation-example.html>.

(5) undefined. <https://bing.com/search?q=>.

3

Tendo em conta que os browsers modernos já suportam o sistema de módulos ECMAScript Modules (ESM), qual a relevância de se ainda usar uma ferramenta como o webpack?

Segundo aula de dúvidas com o professor:

- browsers modernos já suportam ESM, não é necessário fazer bandling, mas...
- o uso do webpack é usado para evitar enviar 1 pedido por módulo
- o uso do webpack é usado para não fornecer o código fonte do módulo
 - minimização do código (inlegibilidade e menor tamanho - poupa banda)
 - também ajuda na compilação caso esteja em typescript - esta é feita através da chamada de um loader, pelo webpack, que entende typescript

Resposta do Bing copilot:

Uma ferramenta como o webpack ainda tem relevância para se usar com o sistema de módulos ECMAScript Modules (ESM), pois ela oferece várias vantagens que os browsers modernos não oferecem, tais como:

- **Otimização do código:** O webpack pode processar os módulos ESM para otimizar o código, eliminando os módulos não usados, reduzindo o tamanho dos arquivos, agrupando os módulos em pacotes, etc. Isso melhora a performance e a eficiência da aplicação web¹.
- **Compatibilidade com versões anteriores:** O webpack pode transpilar os módulos ESM para outros formatos de módulos, como CommonJS ou AMD, que são suportados por browsers mais antigos ou por ambientes como o Node.js. Isso permite que o código escrito com ESM seja portátil e interoperável com diferentes plataformas².
- **Suporte a outros recursos:** O webpack pode integrar outros recursos que não são suportados nativamente pelos browsers, como pré-processadores, carregadores, plugins, variáveis de ambiente, etc. Isso permite que o desenvolvedor use ferramentas e bibliotecas que facilitam e enriquecem o desenvolvimento web³.

Portanto, uma ferramenta como o webpack ainda tem relevância para se usar com o sistema de módulos ESM, pois ela oferece otimização, compatibilidade e suporte a outros recursos que os browsers modernos não oferecem.

- 1: [ECMAScript Modules | webpack](#)
- 2: [How to transpile ES modules with webpack and Node.js](#)
- 3: [webpack - A bundler for javascript and friends](#)

Source: Conversation with Bing, 1/7/2024

- (1) ECMAScript Modules | webpack. <https://webpack.js.org/guides/ecma-script-modules/>.
- (2) How to transpile ES modules with webpack and Node.js. <https://blog.logrocket.com/transpile-es-modules-with-webpack-node-js/>.
- (3) javascript - Webpack 5 and ESM - Stack Overflow. <https://stackoverflow.com/questions/64944796/webpack-5-and-esm>.
- (4) ECMAScript Modules | webpack. <https://webpack.js.org/guides/ecma-script-modules/>.
- (5) How to transpile ES modules with webpack and Node.js. <https://blog.logrocket.com/transpile-es-modules-with-webpack-node-js/>.

4

Realize um ou mais componentes para uso com a biblioteca Spring MVC de forma a expor recursos nos caminhos `/handlers` e `/handlers/{handler-id}`. Um pedido de método GET a `/handlers` retorna uma representação JSON contendo uma lista de links para caminhos com a forma `/handlers/{handler-id}`, um para cada handler que tenha sido executado até ao momento. Um pedido para `/handlers/{handler-id}` retorna uma representação com o número de vezes que o handler foi executado, caso `handler-id` seja o identificador de um handler executado pelo menos uma vez. Assuma que todos os handlers são do tipo `HandlerMethod` e use o método `getShortLogMessage` para obter o identificador dum handler.

Para realizar essa tarefa, podemos usar os seguintes componentes:

- Um **interceptor** que intercepta todas as requisições e registra os handlers executados em um **repositório**. O interceptor também incrementa o número de vezes que cada handler foi executado.
- Um **controlador** que expõe os recursos `/handlers` e `/handlers/{handler-id}` e retorna uma representação JSON com a lista de links ou o número de execuções, respectivamente.

A seguir, apresentamos um possível código em Java para esses componentes, usando a biblioteca **Spring MVC**¹ e o formato **JSON**²:

```
// Um data class que representa um registro de um handler e o número de execuções
public class HandlerRecord {
    private String id; // o identificador do handler
    private int count; // o número de execuções

    // Construtor, getters e setters omitidos
}

// Um repositório que armazena os registros em um mapa
@Component
public class HandlerRepository {
    private Map<String, HandlerRecord> records = new HashMap<>();
}
```

```

// Adiciona ou atualiza um registro no mapa
public void addOrUpdateRecord(HandlerMethod handler) {
    // Obtém o identificador do handler usando o método getShortLogMessage
    String id = handler.getShortLogMessage();
    // Verifica se o mapa já contém um registro com esse identificador
    if (records.containsKey(id)) {
        // Se sim, incrementa o número de execuções
        records.get(id).setCount(records.get(id).getCount() + 1);
    } else {
        // Se não, cria um novo registro com uma execução
        records.put(id, new HandlerRecord(id, 1));
    }
}

// Retorna a lista de registros do mapa
public List<HandlerRecord> getRecords() {
    return new ArrayList<>(records.values());
}

// Retorna um registro específico do mapa, ou null se não existir
public HandlerRecord getRecord(String id) {
    return records.get(id);
}
}

// Um interceptor que intercepta todas as requisições e registra os handlers
executados
@Component
public class HandlerInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private HandlerRepository repository; // injeta o repositório

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        // Verifica se o handler é um HandlerMethod
        if (handler instanceof HandlerMethod) {
            // Adiciona ou atualiza o registro no repositório
            repository.addOrUpdateRecord((HandlerMethod) handler);
        }
    }
}

// Um controlador que expõe os recursos /handlers e /handlers/{handler-id}
@RestController
public class HandlerController {
    @Autowired
    private HandlerRepository repository; // injeta o repositório

    // Define um método GET para o caminho /handlers
    @GetMapping("/handlers")
    public ResponseEntity<List<String>> getHandlers() {
        // Obtém a lista de registros do repositório
        List<HandlerRecord> records = repository.getRecords();
        // Cria uma lista de links para os caminhos /handlers/{handler-id}
        List<String> links = new ArrayList<>();
    }
}

```

```

        for (HandlerRecord record : records) {
            links.add("/handlers/" + record.getId());
        }
        // Retorna uma resposta com o objeto JSON da lista de links
        return ResponseEntity.ok(links);
    }

    // Define um método GET para o caminho /handlers/{handler-id}
    @GetMapping("/handlers/{handler-id}")
    public ResponseEntity<Integer> getHandler(@PathVariable("handler-id") String
id) {
        // Obtém o registro específico do repositório
        HandlerRecord record = repository.getRecord(id);
        // Verifica se o registro existe
        if (record != null) {
            // Se sim, retorna uma resposta com o objeto JSON do número de
execuções
            return ResponseEntity.ok(record.getCount());
        } else {
            // Se não, retorna uma resposta com status code 404 (Not Found)
            return ResponseEntity.notFound().build();
        }
    }
}

```

Esse código é apenas um exemplo e pode ser adaptado conforme as necessidades do projeto. Espero que tenha sido útil. 😊

1: [Spring MVC - A biblioteca para criar aplicações web com o Spring Framework](#)

2: [JSON - Um formato leve de troca de dados](#)

Source: Conversation with Bing, 1/7/2024

(1) Guide to Spring Handler Mappings | Baeldung. <https://www.baeldung.com/spring-handler-mappings>.

(2) 17. Web MVC framework - Spring | Home. <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>.

(3) what's difference between Controller and Handler in Spring MVC?. <https://stackoverflow.com/questions/30165625/whats-difference-between-controller-and-handler-in-spring-mvc>.

(4) undefined. <http://docs.spring.io/autorepo/docs/spring/4.0.4.RELEASE/javadoc-api/org/springframework/web/servlet/handler/SimpleUrlHandlerMapping.html>.

5

Realize um componente para uso com a biblioteca React, recebendo um URI e um período temporal em milissegundos. O componente deve realizar periodicamente um pedido de método GET para esse URI e apresentar ou atualizar o conteúdo (payload) textual da resposta numa caixa de texto (textarea), independentemente do status code da resposta. Caso a realização do pedido resulte numa exceção, a caixa de texto deve apresentar o texto associado a essa exceção. O componente deve ser sensível a mudanças no URI ou no período temporal.

Para realizar esse componente, podemos seguir os seguintes passos:

1. Criar um componente funcional chamado `FetchText` que recebe um `uri` e um `interval` como propriedades. Essas propriedades representam o URI para o pedido GET e o período temporal em milissegundos, respectivamente.
2. Usar o hook `useState` para criar um estado para armazenar o texto da resposta ou da exceção. Inicializar esse estado com uma string vazia.
3. Usar o hook `useEffect` para criar um efeito que é executado quando o componente é montado ou quando o `uri` ou o `interval` mudam. Dentro desse efeito, fazer o seguinte:
 - Criar uma variável para armazenar o identificador do temporizador que será usado para realizar o pedido GET periodicamente.
 - Definir uma função assíncrona que faz o pedido GET usando a API `fetch`¹ e atualiza o estado com o texto da resposta ou da exceção.
 - Chamar essa função imediatamente para fazer o primeiro pedido GET.
 - Usar a função `setInterval` para chamar essa função a cada `interval` milissegundos e armazenar o identificador do temporizador na variável criada anteriormente.
 - Retornar uma função de limpeza que cancela o temporizador usando a função `clearInterval`.
4. Retornar o elemento JSX que renderiza uma caixa de texto (textarea) com o texto do estado.

A seguir, apresentamos um possível código em JavaScript para esse componente, usando a biblioteca **React**² e o formato **JSX**³:

```
// Importa o React e o useState e useEffect hooks
import React, { useState, useEffect } from "react";

// Cria o componente FetchText que recebe um uri e um interval como propriedades
function FetchText({ uri, interval }) {
  // Cria um estado para armazenar o texto da resposta ou da exceção
  const [text, setText] = useState("");

  // Cria um efeito que é executado quando o componente é montado ou quando o uri
  // ou o interval mudam
  useEffect(() => {
    // Cria uma variável para armazenar o identificador do temporizador
    let timerId;

    // Define uma função assíncrona que faz o pedido GET e atualiza o estado
    async function fetchText() {
      try {
        // Faz o pedido GET usando o fetch
        const response = await fetch(uri);
        // Obtém o texto da resposta
        const text = await response.text();
        // Atualiza o estado com o texto da resposta
        setText(text);
      } catch (error) {
        // Se ocorrer uma exceção, atualiza o estado com o texto da exceção
        setText(error.message);
      }
    }
  }, [uri, interval]);
}
```

```

    // Chama a função imediatamente para fazer o primeiro pedido GET
    fetchText();

    // Usa o setInterval para chamar a função a cada interval milisegundos e
    armazena o identificador do temporizador
    timerId = setInterval(fetchText, interval);

    // Retorna uma função de limpeza que cancela o temporizador
    return () => {
        clearInterval(timerId);
    };
}, [uri, interval]); // Adiciona o uri e o interval como dependências do efeito

// Retorna o elemento JSX que renderiza uma caixa de texto com o texto do
estado
return <textarea value={text} readOnly />;
}

// Exporta o componente FetchText
export default FetchText;

```

Esse código é apenas um exemplo e pode ser adaptado conforme as necessidades do projeto. Espero que tenha sido útil. 😊

- 1: [MDN Web Docs - fetch\(\)](#)
- 2: [React - Uma biblioteca JavaScript para criar interfaces de usuário](#)
- 3: [MDN Web Docs - JSX](#)

Source: Conversation with Bing, 1/7/2024

- (1) Bibliotecas de Componentes React UI: Nossas Melhores Escolhas para 2024. <https://kinsta.com/pt/blog/bibliotecas-de-componentes-react/>.
- (2) frontend react para consumo da API de pedidos - GitHub. https://github.com/lipilemos/TEST_SITE_Pedidos.
- (3) DanielSoaresRocha/Sistema-de-Pedidos-Frontend - GitHub. <https://github.com/DanielSoaresRocha/Sistema-de-Pedidos-Frontend>.
- (4) Bibliotecas de Componentes React UI: Nossas Melhores Escolhas para 2024. <https://kinsta.com/pt/blog/bibliotecas-de-componentes-react/>.
- (5) frontend react para consumo da API de pedidos - GitHub. https://github.com/lipilemos/TEST_SITE_Pedidos.
- (6) DanielSoaresRocha/Sistema-de-Pedidos-Frontend - GitHub. <https://github.com/DanielSoaresRocha/Sistema-de-Pedidos-Frontend>.
- (7) undefined. https://github.com/lipilemos/TEST_SITE_Pedidos.git.
- (8) undefined. <https://facebook.github.io/create-react-app/docs/code-splitting>.
- (9) undefined. <https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size>.
- (10) undefined. <https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app>.
- (11) undefined. <https://facebook.github.io/create-react-app/docs/advanced-configuration>.

6

Realize a função

```
useCounter(initial: number): [observed: number, inc: () => void, dec: () => void]
```

para ser usado como hook em componentes para a biblioteca React. Este hook serve para gerir um contador, retornando a função um array com três elementos: o valor atual do contador, uma função para incrementar o contador, e uma função para decrementar o contador.

Para realizar essa função, podemos seguir os seguintes passos:

1. Importar o React e o hook `useState`, que permite usar o estado em componentes funcionais.
2. Definir a função `useCounter` que recebe um parâmetro `initial`, que representa o valor inicial do contador.
3. Usar o hook `useState` para criar um estado para armazenar o valor atual do contador, e uma função para atualizar esse estado. Inicializar o estado com o valor de `initial`.
4. Definir três funções: `inc`, `dec` e `reset`, que incrementam, decrementam e resetam o contador, respectivamente. Essas funções usam a função de atualização do estado para modificar o valor do contador.
5. Retornar um array com o valor atual do contador e as três funções definidas.

A seguir, apresentamos um possível código em JavaScript para essa função, usando a biblioteca **React**:

```
// Importa o React e o useState hook
import React, { useState } from "react";

// Define a função useCounter que recebe um valor inicial
function useCounter(initial) {
  // Cria um estado para armazenar o valor atual do contador e uma função para
  // atualizar esse estado
  const [count, setCount] = useState(initial);

  // Define uma função que incrementa o contador
  function inc() {
    setCount(count + 1);
  }

  // Define uma função que decrementa o contador
  function dec() {
    setCount(count - 1);
  }

  // Define uma função que reseta o contador para o valor inicial
  function reset() {
    setCount(initial);
  }

  // Retorna um array com o valor atual do contador e as três funções
  return [count, inc, dec, reset];
}
```

```
// Exporta a função useCounter
export default useCounter;
```

Época de Recuso, Semestre de Inverno 22/23

1.1

A realização de um pedido de método GET para <https://example.com/games/create>, deve ser interpretado por um intermediário como sendo:

- i. Um pedido não safe e não idempotente.
- ii. Um pedido safe e não idempotente.
- iii. Um pedido não safe e idempotente.
- iv. Um pedido safe e idempotente.

iv

1.2

Uma mensagem de resposta HTTP com status code igual a 200 e Content-Type igual a `application/problem+json` deve ser interpretada por um intermediário como sendo

- i. Uma resposta de sucesso.
- ii. Uma resposta de não sucesso.
- iii. Uma resposta de sucesso ou de não sucesso, dependendo do valor do campo `type` presente na representação.
- iv. Nenhuma das anteriores.

i

Se tem 200 OK deve ser interpretada por um intermediário como sendo uma resposta de sucesso

1.3

O campo `rel` presente num header `Link` representa:

- i. O media-type potencialmente recebido na resposta a um pedido ao recurso alvo do link.
- ii. Um valor booleano que indica se o URI para o destino é absoluto ou relativo.
- iii. O identificador do recurso alvo do link.
- iv. Nenhuma das anteriores.

iv

O campo **rel** presente num header **Link** é usado para indicar **relações entre o recurso atual e outros recursos** ¹. O campo **rel** especifica o tipo de relação que o link tem com o recurso atual ¹. Por exemplo, o valor **stylesheet** indica que o link é um estilo para o documento atual ¹. O campo **href** especifica o URI do recurso alvo do link ¹.

Source: Conversation with Bing, 1/7/2024

(1) HTML link rel Attribute - W3Schools. https://www.w3schools.com/tags/att_link_rel.asp.

(2) LinkHeader - W3C Wiki - World Wide Web Consortium (W3C). <https://www.w3.org/wiki/LinkHeader>.

(3) HTML a rel Attribute - W3Schools. https://www.w3schools.com/TAGS/att_a_rel.asp.

(4) Canonical URL specification via an HTTP header - Stack Overflow. <https://stackoverflow.com/questions/12571326/canonical-url-specification-via-an-http-header>.

(5) Link - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link>.

(6) undefined. <https://www.w3schools.com/>.

(7) undefined. <http://www.link-to-page-here.com/>.

1.4

No contexto da utilização da biblioteca Spring MVC, a execução da função `doFilter` pertencente à interface `HttpFilter`:

- i. Ocorre sempre no contexto da mesma thread.
- ii. Ocorre sempre no contexto da thread associada à instância sobre a qual é chamado o método.
- iii. Ocorre sempre no contexto da thread associada ao pedido HTTP que resultou nesta chamada.
- iv. Ocorre sempre no contexto da thread associada ao handler que vai processar o pedido HTTP.

iii

No contexto da utilização da biblioteca **Spring MVC**, a execução da função **doFilter** pertencente à interface **HttpFilter** ocorre sempre no contexto da **thread associada ao pedido HTTP que resultou nesta chamada** ². O método **doFilter** é usado para interceptar solicitações HTTP e manipular a resposta antes de enviá-la de volta ao cliente ¹. O método **doFilter** é chamado pelo contêiner do servlet para permitir que um filtro processe a solicitação e a resposta ¹. O contêiner do servlet cria uma nova thread para cada solicitação HTTP recebida e chama o método **doFilter** nessa thread ².

Source: Conversation with Bing, 1/7/2024

(1) RequestContextFilter (Spring Framework 6.1.2 API). <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/RequestContextFilter.html>.

(2) spring mvc - Logging MDC with @Async and TaskDecorator - Stack Overflow. <https://stackoverflow.com/questions/45890181/logging-mdc-with-async-and-taskdecorator>.

(3) java - How to use Multithreading concept in Spring MVC for subsequent <https://stackoverflow.com/questions/39200587/how-to-use-multithreading-concept-in-spring-mvc-for-subsequent-operations>.

(4) Asynchronous Requests :: Spring Framework. <https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-ann-async.html>.

1.5

Considere o seguinte componente para a biblioteca React

```
function Counter() {
  const [value, setValue] = useState(0)
  useEffect(() => {
    const tid = setInterval(() => setValue(value + 1), 1000)
    return () => {clearInterval(tid)}
  }, [])
  return (
    <div>{value}</div>
  );
}
```

A colocação deste componente resulta:

- i. Na apresentação constante do valor 0.
- ii. Na apresentação do valor 0, seguida do valor 1 após 1000 milissegundos.
- iii. Na apresentação de um valor numérico, incrementado a cada 1000 milissegundos.
- iv. Nenhuma das anteriores

iii

A colocação do componente **Counter** resulta na apresentação de um valor numérico, incrementado a cada 1000 milissegundos ¹. O componente **Counter** é implementado usando os hooks **useState** e **useEffect** da biblioteca React ¹. O hook **useState** é usado para declarar uma variável de estado **value** e seu valor inicial é definido como 0 ¹. O hook **useEffect** é usado para executar um efeito colateral, que é a atualização do valor do estado **value** a cada 1000 milissegundos ¹. O valor do estado **value** é atualizado usando a função **setValue** ¹. A função **setValue** é chamada com o valor atualizado do estado **value** a cada 1000 milissegundos ¹. O valor atualizado do estado **value** é então renderizado na tela ¹.

Source: Conversation with Bing, 1/7/2024

- (1) Use React hook to implement a self-increment counter. <https://stackoverflow.com/questions/53395147/use-react-hook-to-implement-a-self-increment-counter>.
- (2) React Increment Counter using Hooks – Contact Mentor. <https://contactmentor.com/react-increment-counter-hooks/>.
- (3) How to increase values of different counters when i click different <https://altcademy.com/blog/how-to-increase-values-of-different-counters-when-i-click-different-buttons-in-reactjs/>.
- (4) How to create a simple counter Using ReactJS? - GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-create-a-simple-counter-using-reactjs/>.
- (5) undefined. <https://codesandbox.io/s/nw4jym4n0>.
- (6) undefined. <https://unpkg.com/react@16.7.0-alpha.0/umd/react.development.js>.
- (7) undefined. <https://unpkg.com/react-dom@16.7.0-alpha.0/umd/react-dom.development.js>.

1.6

Quando uma single page application suporta deep linking e o utilizador introduz directamente o URL <https://example.com/games?id=123> (e.g. activando um bookmark), o browser faz sempre um pedido HTTP

- i. de método GET, usando o URL <https://example.com>.
- ii. de método GET, usando o URL <https://example.com/index.html>.
- iii. de método GET, usando o URL <https://example.com/games?id=123>.
- iv. o browser não realiza nenhum pedido HTTP.

iii

Quando uma **single page application** suporta **deep linking** e o utilizador introduz directamente o URL <https://example.com/games?id=123> (e.g. activando um bookmark), o browser faz sempre um pedido HTTP **de método GET, usando o URL <https://example.com/games?id=123>** ¹. Quando o utilizador introduz um URL no navegador, o navegador envia um pedido HTTP para o servidor para obter a página correspondente ². O método HTTP padrão para solicitar recursos é o método **GET** ². O URL que o navegador envia no pedido HTTP é o URL completo que o utilizador introduziu, incluindo quaisquer parâmetros de consulta ². No caso do exemplo fornecido, o URL completo é <https://example.com/games?id=123>, portanto, o navegador envia um pedido HTTP **de método GET, usando o URL <https://example.com/games?id=123>** ¹.

Source: Conversation with Bing, 1/7/2024

- (1) Single Page Applications. <https://web.stanford.edu/class/cs142/lectures/SPA.pdf>.
- (2) Using Hashed vs. Non-Hashed URL Paths in Single Page Apps. <https://blog.bitsrc.io/using-hashed-vs-nonhashed-url-paths-in-single-page-apps-a66234cefc96>.
- (3) How to redirect non supported Deep-link URL to Browser. <https://stackoverflow.com/questions/37428206/how-to-redirect-non-supported-deep-link-url-to-browser>.
- (4) single page application deep linking with login page. <https://stackoverflow.com/questions/17356919/single-page-application-deep-linking-with-login-page>.
- (5) undefined. <http://example.com>.
- (6) undefined. <http://example.com?id=3535>.
- (7) undefined. <http://www.example.org/dirmod?sid=789AB8&type=gen&mod=Core+Pag>.
- (8) undefined. <http://www.example.org/show/A6CD4967199>.
- (9) undefined. <https://www.flickr.com/photos/jarnasen/24593000826/in/explore-2016-01-26/>.
- (10) undefined. <https://v5.reactrouter.com/>.

2

No contexto da utilização de hypermedia no âmbito de APIs HTTP, em que situações deve ser usada o link relation self?

O campo **rel** presente num header **Link** é usado para indicar relações entre o recurso atual e outros recursos ¹. O campo **rel** especifica o tipo de relação que o link tem com o recurso atual ¹. Por exemplo, o valor **stylesheet** indica que o link é um estilo para o documento atual ¹. O campo **href** especifica o URI do recurso alvo do link ¹.

O valor **self** é um tipo de relação que indica que o link aponta para o próprio recurso ¹. O uso do link relation **self** é importante em APIs HTTP que usam **hypermedia** para permitir que os clientes descubram e naveguem pelos recursos disponíveis ¹. O link relation **self** fornece um URI que identifica exclusivamente o recurso atual ¹. Isso permite que os clientes recuperem o recurso atual, atualizem o recurso atual ou excluam o recurso atual usando o método HTTP apropriado ¹.

Source: Conversation with Bing, 1/7/2024

- (1) What is the importance of the self link in hypermedia APIs?. <https://stackoverflow.com/questions/14185011/what-is-the-importance-of-the-self-link-in-hypermedia-apis>.
- (2) How to reference a resource in a update-request of a hypermedia api?. <https://stackoverflow.com/questions/63365589/how-to-reference-a-resource-in-a-update-request-of-a-hypermedia-api>.
- (3) Hypermedia Compliant API. https://docs.oracle.com/cd/E93130_01/service_layer/service%20layer%20API/Content/Introduction/Hypermedia%20Compliant%20API.htm.
- (4) REST with Hypermedia - Hot or Not? - Reflectoring. <https://reflectoring.io/rest-hypermedia/>.
- (5) How to Build Hypermedia API with Spring HATEOAS - Grape Up. <https://grapeup.com/blog/how-to-build-hypermedia-api-with-spring-hateoas/>.

3

**** No desenho de APIs HTTP, quais as vantagens da utilização de métodos idempotentes? ****

A utilização de métodos idempotentes em APIs HTTP tem várias vantagens. Alguns exemplos incluem:

- **Redução de efeitos colaterais:** Métodos idempotentes não devem ter efeitos colaterais, o que significa que eles não devem modificar o estado do servidor ou do cliente¹. Isso ajuda a reduzir o risco de comportamento inesperado e torna as APIs mais previsíveis e fáceis de usar.
- **Tolerância a falhas:** Como os métodos idempotentes podem ser chamados várias vezes sem efeitos colaterais, eles são mais tolerantes a falhas de rede e outras interrupções². Isso ajuda a garantir que as solicitações sejam processadas com segurança e confiabilidade.
- **Cacheabilidade:** Métodos idempotentes são cacheáveis, o que significa que as respostas podem ser armazenadas em cache para melhorar o desempenho e reduzir a carga no servidor³. Isso é particularmente útil em cenários de alto tráfego, onde a redução da carga no servidor pode ajudar a melhorar a escalabilidade e a disponibilidade.
- **Facilidade de uso:** Como os métodos idempotentes são previsíveis e não têm efeitos colaterais, eles são mais fáceis de usar e entender⁴. Isso pode ajudar a reduzir a curva de aprendizado para novos desenvolvedores e melhorar a produtividade geral.

Source: Conversation with Bing, 1/7/2024

(1) Idempotente - Glossário do MDN Web Docs: Definições de termos <https://developer.mozilla.org/pt-BR/docs/Glossary/Idempotent>.

(2) Idempotency - What is an Idempotent REST API?. <https://restfulapi.net/idempotent-rest-apis/>.

(3) What Are Idempotent HTTP Methods? - DEV Community. <https://blog.savetchuk.com/what-are-idempotent-http-methods>.

(4) O que é ser idempotente em REST? O debate continua. <https://www.infoq.com/br/news/2013/05/idempotent/>.

4

Realize um ou mais componentes para uso com a biblioteca Spring MVC de forma a que, para cada pedido HTTP, seja emitida uma mensagem de log com: método HTTP; URI do recurso acessado; status code da resposta; tempo de processamento; identificador do handler que processou o pedido, caso o handler seja do tipo HandlerMethod. Valorizam-se soluções onde o cálculo do tempo de processamento tem em conta mais etapas desse processamento. Use o método getShortLogMessage para obter o identificador dum HandlerMethod

Para registrar uma mensagem de log com informações sobre cada pedido HTTP em Kotlin, podemos criar um **interceptor** personalizado que capture as informações relevantes e as registre em um arquivo de log. O interceptor pode ser implementado usando a interface **HandlerInterceptor** fornecida pela biblioteca Spring MVC ¹.

Aqui está um exemplo de como implementar um interceptor personalizado para registrar informações de log para cada pedido HTTP:

```
class LoggingInterceptor : HandlerInterceptor {

    companion object {
        private val logger =
            LoggerFactory.getLogger(LoggingInterceptor::class.java)
    }

    override fun preHandle(request: HttpServletRequest, response:
        HttpServletResponse, handler: Any): Boolean {
```



```

        request.setAttribute("startTime", System.currentTimeMillis())
        return true
    }

    override fun postHandle(request: HttpServletRequest, response:
    HttpServletResponse, handler: Any, modelAndView: ModelAndView?) {
        val startTime = request.getAttribute("startTime") as Long
        val endTime = System.currentTimeMillis()
        val processingTime = endTime - startTime
        val handlerId = getShortLogMessage(handler)
        val requestUri = request.requestURI
        val statusCode = response.status
        logger.info("HTTP Request: {} {} {} {}ms {}", request.method, requestUri,
        statusCode, processingTime, handlerId)
    }

    private fun getShortLogMessage(handler: Any): String {
        return if (handler is HandlerMethod) {
            "${handler.beanType.simpleName}.${handler.method.name}"
        } else {
            handler.toString()
        }
    }
}

```

Neste exemplo, o interceptor personalizado é implementado como uma classe Kotlin que implementa a interface **HandlerInterceptor** ¹. O método **preHandle** é chamado antes do processamento do pedido HTTP e é usado para registrar o tempo de início do processamento do pedido ¹. O método **postHandle** é chamado após o processamento do pedido HTTP e é usado para registrar as informações relevantes de log, incluindo o método HTTP, URI do recurso acessado, status code da resposta, tempo de processamento e identificador do handler que processou o pedido ¹. O método **getShortLogMessage** é usado para obter o identificador do handler que processou o pedido ¹.

Para usar o interceptor personalizado, podemos registrá-lo no arquivo de configuração do Spring MVC, como mostrado abaixo:

```

@Configuration
class AppConfig : WebMvcConfigurer {

    override fun addInterceptors(registry: InterceptorRegistry) {
        registry.addInterceptor(LoggingInterceptor())
    }
}

```

Neste exemplo, o interceptor personalizado é registrado no método **addInterceptors** da classe **AppConfig** ¹. Isso garante que o interceptor seja aplicado a todos os pedidos HTTP recebidos pelo aplicativo Spring MVC ¹.

Source: Conversation with Bing, 1/7/2024

(1) How to log properly http requests with Spring MVC. <https://stackoverflow.com/questions/6631257/how-to-log-properly-http-requests-with-spring-mvc>.

(2) Spring - Log Incoming Requests | Baeldung. <https://www.baeldung.com/spring-http-logging>.

- (3) Logging all request and responses using Spring Boot Actuator. <https://stackoverflow.com/questions/52926974/logging-all-request-and-responses-using-spring-boot-actuator>.
- (4) Logging Spring WebClient Calls | Baeldung. <https://www.baeldung.com/spring-log-webclient-calls>.

5

Realize um componente para a biblioteca React que recebe a propriedade `f` do tipo `()=>Promise` e que apresenta o fulfillment value ou a rejection reason da promise resultante da avaliação de `f`. Enquanto esta promise estiver pendente, deve ser apresentado um contador incrementado a cada 100 milissegundos. O componente deve ser sensível a mudanças na propriedade `f`.

Aqui está um exemplo de como implementar um componente React que recebe uma propriedade `f` do tipo `()=>Promise` e apresenta o fulfillment value ou a rejection reason da promise resultante da avaliação de `f`. Enquanto esta promise estiver pendente, o componente apresenta um contador incrementado a cada 100 milissegundos:

```
import React, { useState, useEffect } from 'react';

function PromiseComponent({ f }) {
  const [result, setResult] = useState(null);
  const [error, setError] = useState(null);
  const [count, setCount] = useState(0);

  useEffect(() => {
    let isMounted = true;
    f()
      .then((value) => {
        if (isMounted) {
          setResult(value);
        }
      })
      .catch((reason) => {
        if (isMounted) {
          setError(reason);
        }
      });
    return () => {
      isMounted = false;
    };
  }, [f]);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setCount((count) => count + 1);
    }, 100);
    return () => {
      clearInterval(intervalId);
    };
  }, []);

  if (result !== null) {
    return <div>{result}</div>;
  }
}
```

```

    } else if (error !== null) {
      return <div>Error: {error.toString()}</div>;
    } else {
      return <div>Loading: {count}</div>;
    }
  }
}

```

Neste exemplo, o componente **PromiseComponent** é implementado usando os hooks **useState** e **useEffect** da biblioteca React ¹. O componente recebe uma propriedade **f** do tipo **()=>Promise** que é avaliada para obter o fulfillment value ou a rejection reason da promise ¹. Enquanto a promise estiver pendente, o componente apresenta um contador incrementado a cada 100 milissegundos ¹.

Para usar o componente **PromiseComponent**, podemos passar uma função que retorna uma promise como propriedade **f**, como mostrado abaixo:

```

function App() {
  const [count, setCount] = useState(0);

  const f = () => {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (count % 2 === 0) {
          resolve(`Even count: ${count}`);
        } else {
          reject(`Odd count: ${count}`);
        }
      }, 3000);
    });
  };

  return (
    <div>
      <button onClick={() => setCount((count) => count + 1)}>Increment</button>
      <PromiseComponent f={f} />
    </div>
  );
}

```

Neste exemplo, a função **f** retorna uma promise que é avaliada para obter o fulfillment value ou a rejection reason da promise ¹. O componente **PromiseComponent** é usado para apresentar o resultado da avaliação da promise ¹. Quando o botão é clicado, o contador é incrementado e a função **f** é chamada novamente para avaliar a promise ¹.

Source: Conversation with Bing, 1/7/2024

(1) How to Handle Promises in React - Upmosty. <https://upmosty.com/tutorials/how-to-handle-promises-in-react>.

(2) GitHub - capaj/react-promise: a react.js hook for general promise in <https://github.com/capaj/react-promise>.

(3) How to Render React Components with Promises Inside?. <https://thewebdev.info/2021/09/25/how-to-render-react-components-with-promises-inside/>.

(4) Promise based props in React - JavaScript in Plain English. <https://javascript.plainenglish.io/promise-based-prop-in-react-78a77440f4fc>.

(5) Rendering React components with promises inside the render method. <https://stackoverflow.com/questions/33242378/rendering-react-components-with-promises-inside-the-render-method>.

(6) undefined. <https://cataas.com/cat?json=true>.

(7) undefined. <https://cataas.com>.

6

Realize a função

```
useInput(initial: string): [  
  currentValue: string,  
  changeHandler: React.ChangeEvent<HTMLInputElement> => void  
]
```

para ser usada como hook em componentes para a biblioteca React, tal como ilustrado no seguinte exemplo

```
function Example() {  
  const [value, handler] = useInput("")  
  return (  
    <input type="text" value={value} onChange={handler} />  
  );  
}
```

Implementação da função `useInput`:

```
import { useState } from 'react';  
  
function useInput(initial) {  
  const [currentValue, setCurrentValue] = useState(initial);  
  
  const changeHandler = (event) => {  
    setCurrentValue(event.target.value);  
  };  
  
  return [currentValue, changeHandler];  
}
```

Nesta implementação, a função `useInput` recebe um argumento `initial` que é o valor inicial do estado. A função retorna um array com dois elementos: o valor atual do estado `currentValue` e o manipulador de mudança `changeHandler` que atualiza o valor do estado sempre que o usuário digita algo no campo de entrada.

Para usar a função `useInput` em um componente React, basta chamar a função e passar o valor inicial como argumento, como mostrado abaixo:

```
function Example() {
  const [value, handler] = useInput("");
  return (
    <input type="text" value={value} onChange={handler} />
  );
}
```

Neste exemplo, a função `useInput` é chamada com um valor inicial vazio `""` e os valores de estado retornados são desestruturados em `value` e `handler`. O valor atual do estado é usado como o valor do campo de entrada e o manipulador de mudança é usado para atualizar o valor do estado sempre que o usuário digita algo no campo de entrada.

Source: Conversation with Bing, 1/7/2024

(1) reactjs - React(Typescript) onchange event type - Stack Overflow. <https://stackoverflow.com/questions/62591811/reacttypescript-onchange-event-type>.

(2) Typescript input onchange event.target.value - Stack Overflow. <https://stackoverflow.com/questions/40676343/typescript-input-onchange-event-target-value>.

(3) How to use and specify Generic Hooks in React with Typescript?. <https://stackoverflow.com/questions/65214132/how-to-use-and-specify-generic-hooks-in-react-with-typescript>.

(4) React + TypeScript: Handling input onChange event - KindaCode. <https://www.kindacode.com/article/react-typescript-handling-input-onchange-event/>.

Época Especial, Semestre Inverno, 22/23

1.1

No protocolo HTTP, um pedido de método GET para <https://example.com/games/create>:

- i. Solicita a criação do recurso com URI <https://example.com/games/create>.
- ii. Solicita a criação de um recurso com URI <https://example.com/games/{id}>, onde o valor de id é determinado pelo servidor.
- iii. Solicita uma representação do recurso com URI <https://example.com/games/create>.
- iv. Deve resultar sempre numa resposta com status code 405 (method not allowed).

iii

1.2

Uma mensagem de resposta HTTP com status code igual a 401 e Content-Type igual a `application/problem+json` deve ser interpretada por um intermediário como sendo:

- i. Uma resposta de sucesso.
- ii. Uma resposta de não sucesso.
- iii. Uma resposta de sucesso ou de não sucesso, dependendo do valor do campo `type` presente na representação.
- iv. Nenhuma das anteriores.

ii

Uma mensagem de resposta HTTP com status code igual a **401** e **Content-Type** igual a **application/problem+json** deve ser interpretada por um intermediário como sendo **uma resposta de não sucesso** ³. O código de status **401** indica que a solicitação não foi concluída porque o cliente não possui credenciais de autenticação válidas para o recurso solicitado ³. O **Content-Type** é um cabeçalho HTTP que indica o tipo de mídia da representação de uma resposta HTTP. O valor **application/problem+json** é um tipo de mídia definido pelo grupo de trabalho de

aplicativos HTTP da IETF para representar problemas HTTP ¹. O uso do tipo de mídia **application/problem+json** indica que a resposta contém informações sobre um problema que ocorreu durante o processamento da solicitação ¹.

Source: Conversation with Bing, 1/7/2024

(1) 401 Unauthorized - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401>.

(2) The application/problem+json Content-Type - SymfonyCasts. <https://symfonycasts.com/screeencast/rest/application-problem>.

(3) HTTP status and error codes for JSON - Google Cloud. https://cloud.google.com/storage/docs/json_api/v1/status-codes.

(4) How to Fix the 401 Error (5 Solutions) - Kinsta®. <https://kinsta.com/knowledgebase/401-error/>.

1.3

O campo **rel** presente num header **Link** representa:

- i. O identificador do recurso alvo do link.
- ii. O media-type potencialmente recebido na resposta a um pedido ao recurso alvo do link.
- iii. Um valor booleano que indica se o URI para o destino é absoluto ou relativo.
- iv. Nenhuma das anteriores.

iv

O campo **rel** presente num header **Link** é usado para indicar relações entre o recurso atual e outros recursos ¹. O campo **rel** especifica o tipo de relação que o link tem com o recurso atual ¹. Por exemplo, o valor **stylesheet** indica que o link é um estilo para o documento atual ¹. O campo **href** especifica o URI do recurso alvo do link ¹.

O valor **rel** não representa o identificador do recurso alvo do link ou o media-type potencialmente recebido na resposta a um pedido ao recurso alvo do link ¹. O valor **rel** indica o tipo de relação que o link tem com o recurso atual ¹.

Source: Conversation with Bing, 1/7/2024

(1) Link - HTTP | MDN - MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link>.

(2) LinkHeader - W3C Wiki - World Wide Web Consortium (W3C). <https://www.w3.org/wiki/LinkHeader>.

(3) HTML attribute: rel - HTML: HyperText Markup Language | MDN. <https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel>.

(4) HTML link rel Attribute - W3Schools. https://www.w3schools.com/tags/att_link_rel.asp.

(5) undefined. <https://example.com>.

(6) undefined. <https://bad.example>.

(7) undefined. <https://example.com/%E8%8B%97%E6%9D%A1>.

(8) undefined. <https://example.com/苗条>.

(9) undefined. <https://one.example.com>.

(10) undefined. <https://two.example.com>.

(11) undefined. <https://three.example.com>.

1.4

Na biblioteca Spring MVC e na configuração por omissão:

- i. E usada apenas uma instância por cada tipo de controller.
- ii. E criada sempre uma nova instância dum controller por cada pedido.
- iii. Apenas são criadas novas instâncias de controller quando todas as instâncias existentes estiverem a ser usadas.
- iv. Não são criadas instâncias de controller porque todos os métodos têm de pertencer ao companion object.

i

Na biblioteca Spring MVC e na configuração por omissão, é usada apenas **uma instância por cada tipo de controller** ¹. Isso significa que, por padrão, o contêiner do Spring cria apenas uma instância de cada classe de controlador e a reutiliza para todos os pedidos HTTP recebidos pelo aplicativo ¹. Essa abordagem é conhecida como escopo de singleton e é usada para garantir que os controladores sejam eficientes e escaláveis ¹.

Source: Conversation with Bing, 1/7/2024

- (1) Scope of a Spring-Controller and its instance-variables. <https://stackoverflow.com/questions/11139571/scope-of-a-spring-controller-and-its-instance-variables>.
- (2) how to create a controller based on auto configuration spring boot 1.4. <https://stackoverflow.com/questions/39552060/how-to-create-a-controller-based-on-auto-configuration-spring-boot-1-4-0>.
- (3) How to set default handler method in controller in Spring mvc?. <https://stackoverflow.com/questions/28430665/how-to-set-default-handler-method-in-controller-in-spring-mvc>.
- (4) java - Are Spring MVC Controllers Singletons? - Stack Overflow. <https://stackoverflow.com/questions/11508405/are-spring-mvc-controllers-singletons>.

1.5

Considere o seguinte componente para a biblioteca React

```
function Counter() {  
  const [value, setValue] = useState(0)  
  useEffect(() => {  
    const tid = setInterval(() => setValue((x) => x + 1), 1000)  
    return () => {clearInterval(tid)}  
  }, [setValue])  
  return (  
    <div>{value}</div>  
  );  
}
```

A colocação deste componente resulta:

- i. Na apresentação constante do valor 0.
- ii. Na apresentação do valor 0, seguida do valor 1 após 1000 milissegundos.
- iii. Na apresentação de um valor numérico, incrementado a cada 1000 milissegundos.
- iv. Nenhuma das anteriores

iii

A colocação deste componente resulta na apresentação de um valor numérico, incrementado a cada 1000 milissegundos. O componente é implementado usando os hooks `useState` e `useEffect` da biblioteca React. O hook `useState` é usado para declarar uma variável de estado `value` e seu valor inicial é definido como `0`. O hook `useEffect` é usado para executar um efeito colateral, que é a atualização do valor do estado `value` a cada 1000 milissegundos. O valor do estado `value` é atualizado usando a função `setValue`. A função `setValue` é chamada com o valor atualizado do estado `value` a cada 1000 milissegundos. O valor atualizado do estado `value` é então renderizado na tela.

1.6

****Quando uma single page application suporta deep linking e o utilizador introduz diretamente o URL <https://example.com/games?id=123> (e.g. ativando um bookmark), o browser faz sempre um pedido HTTP**

- i. de método GET, usando o URL <https://example.com/index.html>.
- ii. de método GET, usando o URL <https://example.com/>.
- iii. de método GET, usando o URL <https://example.com/games>.
- iv. de método GET, usando o URL <https://example.com/games?id=123>

iv

Quando uma **single page application** suporta **deep linking** e o utilizador introduz directamente o URL <https://example.com/games?id=123> (e.g. ativando um bookmark), o browser faz sempre um pedido HTTP **de método GET, usando o URL <https://example.com/games?id=123>**¹. Quando o utilizador introduz um URL no navegador, o navegador envia um pedido HTTP para o servidor para obter a página correspondente². O método HTTP padrão para solicitar recursos é o método **GET**². O URL que o navegador envia no pedido HTTP é o URL completo que o utilizador introduziu, incluindo quaisquer parâmetros de consulta². No caso do exemplo fornecido, o URL completo é <https://example.com/games?id=123>, portanto, o navegador envia um pedido HTTP **de método GET, usando o URL <https://example.com/games?id=123>**¹.

Source: Conversation with Bing, 1/7/2024

- (1) Single Page Applications. <https://web.stanford.edu/class/cs142/lectures/SPA.pdf>.
- (2) Using Hashed vs. Non-Hashed URL Paths in Single Page Apps. <https://blog.bitsrc.io/using-hashed-vs-nonhashed-url-paths-in-single-page-apps-a66234cefc96>.
- (3) How to redirect non supported Deep-link URL to Browser. <https://stackoverflow.com/questions/37428206/how-to-redirect-non-supported-deep-link-url-to-browser>.
- (4) single page application deep linking with login page. <https://stackoverflow.com/questions/17356919/single-page-application-deep-linking-with-login-page>.
- (5) undefined. <http://example.com>.
- (6) undefined. <http://example.com?id=3535>.
- (7) undefined. <http://www.example.org/dirmod?sid=789AB8&type=gen&mod=Core+Pag>.
- (8) undefined. <http://www.example.org/show/A6CD4967199>.
- (9) undefined. <https://www.flickr.com/photos/jarnasen/24593000826/in/explore-2016-01-26/>.
- (10) undefined. <https://v5.reactrouter.com/>.

2

**** No desenho de APIs HTTP, quais as vantagens da utilização de métodos idempotentes? ****

A utilização de métodos idempotentes em APIs HTTP tem várias vantagens. Alguns exemplos incluem:

- **Redução de efeitos colaterais:** Métodos idempotentes não devem ter efeitos colaterais, o que significa que eles não devem modificar o estado do servidor ou do cliente¹. Isso ajuda a reduzir o risco de comportamento inesperado e torna as APIs mais previsíveis e fáceis de usar.
- **Tolerância a falhas:** Como os métodos idempotentes podem ser chamados várias vezes sem efeitos colaterais, eles são mais tolerantes a falhas de rede e outras interrupções². Isso ajuda a garantir que as solicitações sejam processadas com segurança e confiabilidade.
- **Cacheabilidade:** Métodos idempotentes são cacheáveis, o que significa que as respostas podem ser armazenadas em cache para melhorar o desempenho e reduzir a carga no servidor³. Isso é particularmente útil em cenários de alto tráfego, onde a redução da carga no servidor pode ajudar a melhorar a escalabilidade e a disponibilidade.
- **Facilidade de uso:** Como os métodos idempotentes são previsíveis e não têm efeitos colaterais, eles são mais fáceis de usar e entender⁴. Isso pode ajudar a reduzir a curva de aprendizado para novos desenvolvedores e melhorar a produtividade geral.

Source: Conversation with Bing, 1/7/2024

(1) Idempotente - Glossário do MDN Web Docs: Definições de termos <https://developer.mozilla.org/pt-BR/docs/Glossary/Idempotent>.

(2) Idempotency - What is an Idempotent REST API?. <https://restfulapi.net/idempotent-rest-api/>.

(3) What Are Idempotent HTTP Methods? - DEV Community. <https://blog.savetchuk.com/what-are-idempotent-http-methods>.

(4) O que é ser idempotente em REST? O debate continua. <https://www.infoq.com/br/news/2013/05/idempotent/>.

3

Tendo em conta que os browsers modernos já suportam o sistema de módulos ECMAScript Modules (ESM), qual a relevância de se ainda usar uma ferramenta como o webpack?

Uma ferramenta como o webpack ainda tem relevância para se usar com o sistema de módulos ECMAScript Modules (ESM), pois ela oferece várias vantagens que os browsers modernos não oferecem, tais como:

- **Otimização do código:** O webpack pode processar os módulos ESM para otimizar o código, eliminando os módulos não usados, reduzindo o tamanho dos arquivos, agrupando os módulos em pacotes, etc. Isso melhora a performance e a eficiência da aplicação web¹.
- **Compatibilidade com versões anteriores:** O webpack pode transpilar os módulos ESM para outros formatos de módulos, como CommonJS ou AMD, que são suportados por browsers mais antigos ou por ambientes como o Node.js. Isso permite que o código escrito com ESM seja portátil e interoperável com diferentes plataformas².
- **Suporte a outros recursos:** O webpack pode integrar outros recursos que não são suportados nativamente pelos browsers, como pré-processadores, carregadores, plugins, variáveis de ambiente, etc. Isso permite que o desenvolvedor use ferramentas e bibliotecas que facilitam e enriquecem o desenvolvimento web³.

Portanto, uma ferramenta como o webpack ainda tem relevância para se usar com o sistema de módulos ESM, pois ela oferece otimização, compatibilidade e suporte a outros recursos que os browsers modernos não oferecem.

- 1: [ECMAScript Modules | webpack](#)
- 2: [How to transpile ES modules with webpack and Node.js](#)
- 3: [webpack - A bundler for javascript and friends](#)

Source: Conversation with Bing, 1/7/2024

- (1) ECMAScript Modules | webpack. <https://webpack.js.org/guides/ecma-script-modules/>.
- (2) How to transpile ES modules with webpack and Node.js. <https://blog.logrocket.com/transpile-es-modules-with-webpack-node-js/>.
- (3) javascript - Webpack 5 and ESM - Stack Overflow. <https://stackoverflow.com/questions/64944796/webpack-5-and-esm>.
- (4) ECMAScript Modules | webpack. <https://webpack.js.org/guides/ecma-script-modules/>.
- (5) How to transpile ES modules with webpack and Node.js. <https://blog.logrocket.com/transpile-es-modules-with-webpack-node-js/>.

4

Realize um ou mais componentes para a plataforma Spring MVC, de forma a que um recurso seja exposto no caminho `/errors`. Um pedido de método GET para esse recurso deve retornar uma mensagem com uma representação contendo um objeto JSON. Esse objeto deve representar os pedidos processados nos últimos 10 minutos e que resultaram numa resposta com status code igual a 500. A representação de cada pedido deve incluir: o momento em que o pedido foi recebido, o método do pedido, o URI do pedido, e o nome do controlador e do método responsável pelo processamento desse pedido (caso o processamento tenha sido realizado por um controlador).

Exemplo de como implementar um componente Spring MVC em Kotlin que expõe um recurso no caminho `/errors` e retorna uma mensagem com uma representação contendo um objeto JSON. Esse objeto representa os pedidos processados nos últimos 10 minutos e que resultaram em uma resposta com status code igual a 500. A representação de cada pedido inclui o momento em que o pedido foi recebido, o método do pedido, o URI do pedido e o nome do controlador e do método responsável pelo processamento desse pedido (caso o processamento tenha sido realizado por um controlador):

```
import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController
import java.time.LocalDateTime

@RestController
@RequestMapping("/errors")
class ErrorController {

    private val errors = mutableListOf<Error>()

    @GetMapping
    fun getErrors(): List<Error> {
        val tenMinutesAgo = LocalDateTime.now().minusMinutes(10)
        return errors.filter { it.timestamp.isAfter(tenMinutesAgo) }
    }

    fun addError(error: Error) {
        errors.add(error)
    }
}
```

```

    }

    data class Error(
        val timestamp: LocalDateTime,
        val method: String,
        val uri: String,
        val controller: String?,
        val handler: String?
    )

    @RestControllerAdvice
    class ErrorControllerAdvice {

        @ExceptionHandler(HttpStatus.INTERNAL_SERVER_ERROR::class)
        fun handleInternalServerError(ex: Exception) {
            val error = Error(
                LocalDateTime.now(),
                "GET",
                "/errors",
                null,
                null
            )
            addError(error)
        }
    }
}

```

Nesta implementação, a classe `ErrorController` é um controlador Spring MVC que expõe um recurso no caminho `/errors`¹. O método `getErrors` retorna uma lista de erros que foram processados nos últimos 10 minutos e que resultaram em uma resposta com status code igual a 500¹. O método `addError` é usado para adicionar um novo erro à lista de erros¹. A classe `Error` é uma classe de dados que representa um erro e inclui o momento em que o pedido foi recebido, o método do pedido, o URI do pedido e o nome do controlador e do método responsável pelo processamento desse pedido (caso o processamento tenha sido realizado por um controlador)¹. A classe `ErrorControllerAdvice` é um controlador de exceções que é usado para capturar exceções do tipo `HttpStatus.INTERNAL_SERVER_ERROR` e adicionar um novo erro à lista de erros¹.

Para usar o componente `ErrorController`, basta injetá-lo em outros componentes Spring MVC e chamar o método `addError` sempre que ocorrer um erro com status code igual a 500.

Source: Conversation with Bing, 1/7/2024

(1) Passing JSON objects in a REST HTTP GET request using Spring MVC. <https://stackoverflow.com/questions/24551412/passing-json-objects-in-a-rest-http-get-request-using-spring-mvc>.

(2) Parsing json into java objects in spring-mvc - Stack Overflow. <https://stackoverflow.com/questions/3921736/parsing-json-into-java-objects-in-spring-mvc>.

(3) Rendering Exceptions in JSON with Spring | Baeldung. <https://www.baeldung.com/spring-exceptions-json>.

(4) JSON Parameters with Spring MVC | Baeldung. <https://www.baeldung.com/spring-mvc-send-json-parameters>.

(5) Error Handling for REST with Spring in Kotlin - Baeldung. <https://www.baeldung.com/kotlin/spring-rest-error-handling>.

(6) Kotlin Get Value from JSON - Stack Overflow. <https://stackoverflow.com/questions/44057957/kotlin-get-value-from-json>.

(7) Make a standard request | Volley. <https://google.github.io/volley/request.html>.

(8) Kotlin and @Valid Spring Annotation | Baeldung on Kotlin. <https://www.baeldung.com/kotlin/valid-spring-annotation>.

5

****** Realize um componente para a biblioteca React que receba as propriedade `f` do tipo `()=>Promise` e que apresenta o fulfillment value ou a rejection reason da promise resultante da avaliação de `f`. O componente deve também apresentar um botão que promove a reavaliação da função `f` sempre que premido. Este botão deve estar desabilitado enquanto a promessa retornada pela última avaliação não se tiver completado. O componente deve ser sensível a mudanças na propriedade `f`.******

Exemplo de como implementar um componente React que recebe uma propriedade `f` do tipo `()=>Promise<string>` e apresenta o fulfillment value ou a rejection reason da promise resultante da avaliação de `f`. O componente também apresenta um botão que promove a reavaliação da função `f` sempre que premido. Este botão está desabilitado enquanto a promessa retornada pela última avaliação não se tiver completado. O componente é sensível a mudanças na propriedade `f`:

```
import React, { useState, useEffect } from 'react';

function PromiseComponent({ f }) {
  const [result, setResult] = useState(null);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(false);

  const handleClick = () => {
    setLoading(true);
    f()
      .then((value) => {
        setResult(value);
        setLoading(false);
      })
      .catch((reason) => {
        setError(reason);
        setLoading(false);
      });
  };

  useEffect(() => {
    setLoading(true);
    f()
      .then((value) => {
        setResult(value);
        setLoading(false);
      })
      .catch((reason) => {
        setError(reason);
        setLoading(false);
      });
  }, [f]);

  return (
```

```

    <div>
      <button onClick={handleClick} disabled={loading}>
        {loading ? 'Loading...' : 'Refresh'}
      </button>
      {result !== null && <div>{result}</div>}
      {error !== null && <div>Error: {error.toString()}</div>}
    </div>
  );
}

```

Nesta implementação, o componente `PromiseComponent` é implementado usando os hooks `useState` e `useEffect` da biblioteca React. O componente recebe uma propriedade `f` do tipo `()=>Promise<string>` que é avaliada para obter o fulfillment value ou a rejection reason da promise. O componente também apresenta um botão que promove a reavaliação da função `f` sempre que premido. Este botão está desabilitado enquanto a promessa retornada pela última avaliação não se tiver completado. O estado `loading` é usado para controlar o estado do botão. O estado `result` é usado para armazenar o fulfillment value da promise. O estado `error` é usado para armazenar a rejection reason da promise.

Para usar o componente `PromiseComponent`, basta chamar a função e passar a função que retorna uma promise como propriedade `f`, como mostrado abaixo:

```

function App() {
  const [count, setCount] = useState(0);

  const f = () => {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (count % 2 === 0) {
          resolve(`Even count: ${count}`);
        } else {
          reject(`Odd count: ${count}`);
        }
      }, 3000);
    });
  };

  return (
    <div>
      <button onClick={() => setCount((count) => count + 1)}>Increment</button>
      <PromiseComponent f={f} />
    </div>
  );
}

```

Neste exemplo, a função `f` retorna uma promise que é avaliada para obter o fulfillment value ou a rejection reason da promise. O componente `PromiseComponent` é usado para apresentar o resultado da avaliação da promise. Quando o botão é clicado, a função `f` é chamada novamente para avaliar a promise.

Source: Conversation with Bing, 1/7/2024

- (1) Pass React Native Promise between activities on Android [Kotlin]. <https://stackoverflow.com/questions/48815234/pass-react-native-promise-between-activities-on-android-kotlin>.
- (2) Promise in Kotlin with Android - Stack Overflow. <https://stackoverflow.com/questions/58319868/promise-in-kotlin-with-android>.
- (3) How to create a Promise from the nested kotlin.js.Promise?. <https://stackoverflow.com/questions/44544125/how-to-create-a-promise-from-the-nested-kotlin-js-promise>.
- (4) Working with promises in a React class component. <https://stackoverflow.com/questions/72237572/working-with-promises-in-a-react-class-component>.
- (5) Creating a custom React hook that deals with promises. <https://dev.to/lukasmoellerch/a-hook-to-use-promise-results-2hfd>.