Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Desenvolvimento de Aplicações Web
Teste Final, Época de Recurso, Semestre de Verão, 20/21
Duração: 2 horas

Divida as respostas em pelo menos duas folhas fisicamente independentes, de forma a permitir a correcção em paralelo dos dois seguintes grupos de questões:

- Questões 1, 3, e 5 (questões ímpares).
- Questões 2, 4, e 6 (questões pares).

Na questão 1, a resposta deve incluir o texto completo da opção e não apenas o seu número.

1. (6) Para cada uma das seguintes questões, seleccione a opção de resposta correcta. A escolha de uma opção errada contribui negativamente para o resultado final com um terço da cotação da questão.

   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.

   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.

   1.3. Na platforma 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.

   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.

   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.

   1.6. O resultado da avaliação da expressão JSX `<div />` 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.

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

3. (2) 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*?

4. (2) Indique o que é necessário realizar para que uma aplicação *single page application* suporte *deep-linking*.

5. (4) 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 anónimos aos recursos da API. Cada elemento da lista é composto pelo URI do recurso e pelo número de acessos anónimos a esse recurso.

6. (4) Realize um componente para a *framework* React que implementa um cronómetro 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 cronómetro, 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 milisegundos desde 1 de janeiro de 1970.

Paulo Pereira,
Pedro Félix,
21 de julho de 2021