INTERFACE PESSOA-MÁQUINA · Enunciado do teste

Licenciatura em Engenharia Informática · Universidade do Minho

13 de junho de 2024 – Duração: 2h

Regras para a realização do Exame:

- ***Não são permitidos telemóveis*** *na sala. Se trouxe um telemóvel consigo, coloque-o na mochila ou na mesa junto dos docentes. A mochila deverá estar no chão à sua frente.*
- *Coloque o* ***relógio em cima da mesa*** *à sua frente.*
- *Em cima da mesa só deverão ter o enunciado do teste, as folhas para consulta e o material de escrita (eventualmente, o relógio).*
- *O não cumprimento destas regras implica a anulação do exame e comunicação dos factos à presidência do Conselho Pedagógico da Escolha de Engenharia, para eventual instauração de processo disciplinar por tentativa de fraude.*
- *A cotação mínima de cada questão é de zero valores (errar uma questão não desconta nas restantes).*

Leia com atenção todo o enunciado do teste e **responda** às questões de escolha mútipla **na folha de respostas** no fim do enunciado. **Entregue todo o enunciado** no final.

Parte 1 (8 valores)

Questão 1 ♣ Considere o princípio da *Familiarity* (*Como o conhecimento prévio se aplica a um novo sistema*), das seguintes heurísticas de Nielsen indique as duas que considera mais poderem contribuir para este princípio:

A Reconhecer em vez de recordar

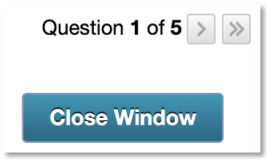
B Correspondência entre o sistema e o mundo real

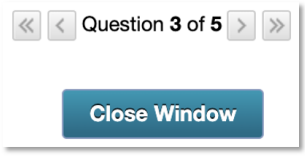
C Prevenção de erros

D Desenho estético e minimalista

E Flexibilidade e eficiência de utilização

Questão 2 ♣ Considere os botões de navegação entre questões, e de fechar a janela, existentes nas páginas dos testes do Blackboard. Os botões são apresentados no canto inferior esquerdo das páginas das questões. As figuras abaixo apresentam os botões da primeira página, de uma página intermédia e da última página de um teste com cinco questões.





Seleccione as opções que tornam a seguinte afirmação válida: *Os botões em causa são um exemplo de...*

A *Consistency* – eliminando a possibilidade de confusões

B *Substitutivity* – sendo possível realizar a mesma acção de vários modos

C *Observability* – em particular de *browsability* (possibilidade de visitar a informação relevante do teste)

D *Familiarity* – existindo utilização de conhecimento anterior ou padrões de interacção conhecidos

E Nenhuma das anteriores

Questão 3 ♣ Na perspectiva do modelo de interacção de Norman, uma das formas de tornar uma interface mais fácil de usar é diminuir o fosso da avaliação. Seleccione as opções que tornam a seguinte afirmação válida: *O fosso da avaliação refere-se*

A à distância física entre o usuário e o dispositivo de interação.

B ao processo pelo qual os usuários interpretam o estado do sistema e avaliam o resultado de suas ações anteriores.

C à quantidade de tempo que um sistema leva para responder a uma ação do utilizador.

D à capacidade do utilizador determinar o estado funcional do sistema.

E ao fosso entre o design do produto e a estética visual desejada pelo utilizador.

Questão 4 ♣ Sabendo que pretende obter uma lista de clientes (nif, nome e lista de compras efetuadas) a partir de `http://xpto.pt/clients`, guardando-a numa propriedade reativa do componente, indique quais das seguintes opções estão correctas:

A
```
this.list = fetch('http://xpto.pt/clients')
                .then(response => response.json());
```

B
```
fetch('http://xpto.pt/clients')
  .then(response => response.json())
  .then(data => this.list=data);
```

C
```
this.list = fetch('http://xpto.pt/clients')
                .then(data => console.log(data));
```

D
```
fetch('http://xpto.pt/clients')
  .then(response => response.json())
  .then(data => console.log(data));
```

E
```
fetch('http://xpto.pt/clients')
  .then(data => this.list=data);
```

Questão 5 ♣ Os protótipos da interface podem ser desenvolvidos a diferentes níveis de fidelidade em relação à interface final. Indique quais dos seguintes fatores considera serem vantagens da utilização de protótipos de baixa fidelidade:

A Menor distância para o produto final

B Menor controlo sobre o nível de abstração

C Baixo custo

D Limitações de funcionalidade

E Facilidade de construção

Questão 6 ♣ Suponha que lhe foi pedido para implementar uma store para guardar os alunos de uma turma. Indique quais das seguintes declarações estariam corretas (assuma que foi feito o **`import`** de `defineStore`):

A
```
export const useStudentStore = defineStore('studentStore', {
    state: () => ({
        listStudent: []
    })
});
function add(newStudent) { this.listStudent.push(newStudent); }
```

B
```
export const useStudentStore = defineStore('studentStore', {
    data: function () {
        listStudent: []
    },
    methods: {
        add(newStudent) { this.listStudent.push(newStudent); }
    }
});
```

C
```
export const useStudentStore = defineStore('studentStore', {
    state: () => ({
        listStudent: ''
    }),
    actions: {
        add(newStudent) { this.listStudent.push(newStudent); }
    }
});
```

D
```
export const useStudentStore = defineStore('studentStore', {
    state: () => ({
        listStudent: []
    }),
    actions: {
        add(newStudent) { this.listStudent.push(newStudent); }
    }
});
```

E
```
export const useStudentStore = defineStore('studentStore', {
    state: () => ({
        listStudent: []
    }),
    getters: {
        add: (newStudent) => { this.listStudent.push(newStudent); }
    }
});
```

Questão 7 ♣ Considere que tem num componente Vue.js uma propriedade reativa `clients` com uma lista de clientes. Sabendo que cada objeto cliente tem um id, um nome e uma lista de compras, indique quais das seguintes declarações permitiriam apresentar corretamente os nomes dos clientes no *template* do componente:

A
```
<ul>
<li v-for="items as item" :key="item.id">{{ item.name }}</li>
</ul>
```

B
```
<ul>
<li v-for="(item, index) in items" :key="index">{{ item.name }}</li>
</ul>
```

C
```
<ul>
<li v-for="item in items" :key="item.id">{{ item.name }}</li>
</ul>
```

D
```
<ul>
<li v-for="item in items">{{ item.name }}</li>
</ul>
```

E
```
<ul>
<li v-if="items.length">{{ items[0].name }}</li>
</ul>
```

Questão 8 ♣ Considere que tem um componente Vue.js com uma lista de clientes e pretende apresentar o tamanho da lista. Indique quais das seguintes soluções garantem que o valor apresentado é atualizado sempre que a lista muda:

A
```
<template>
  ...
  <p v-if="items.length">A lista tem {{ value }} items.</p>
  ...
</template>
```

B
```
<template>
  ...
  <p>A lista tem {{ items.length }} items.</p>
  ...
</template>
```

C
```
<template>
  ...
  <p v-for="n in items.length">A lisata tem {{ n }} items.</p>
  ...
</template>
```

D
```
<template>
  ...
  <p v-model="items.length">A lista tem {{ v-model }} items.</p>
  ...
</template>
```

E
```
<template>
  ...
  <p>A lista tem {{ itemsCount }} items.</p>
  ...
</template>
<script>
...
   computed: {
      itemsCount() {
         return this.items.length;
      }
   }
...
</script>
```

Parte 2 (12 valores)

Questão 9

Considere que se pretende desenvolver um applicação para uma empresa de transporte de passageiros. Como forma de manter o serviço o mais barato possível, a empresa pretende promover a partilha do carro entre clientes. Assim, mesmo quando já está a transportar alguém, o motorista poderá parar para recolher outro passageiro. A recolha de novos passageiros, não deverá, no entanto, causar atrasos exagerados nos serviços (a percentagem de atraso aceitável será configurável centralmente).

A empresa pretende operar num modelo descentralizado. A central faz o *broadcast* dos pedidos de transporte e cada motorista será responsável por tomar a decisão de ir buscar ou não um dado passageiro. Para tal, deverá ter o apoio da aplicação que agora se pretende desenvolver. Assim, quando chega a **notificação de um novo pedido de viagem** (com indicação de origem, destino e número de passageiros a transportar), cada motorista deverá decidir (com o apoio da informação fornecida pela aplicação):

1. **se tem lugar para os novos passageiros** (poderá ter lugares disponíveis no veículo, ou vir a libertar lugares antes de chegar ao local de origem do novo pedido)
2. **se o pedido é compatível com as viagens dos passageiros actuais** (para isso o motorista deverá ter em consideração a localização do novo serviço face aos actuais e o atraso que o desvio gerado pelo novo serviço irá causar).

e, em função disso, aceitar ou rejeitar o serviço O primeiro motorista a sinalizar a aceitação, fica com o serviço.

Sempre que um serviço termina, o sistema deverá ajudar o motorista a determinar qual o valor a pagar e qual o próximo destino. O preço a cobrar tem em consideração os valores previstos e efetivos da distância percorrida e do tempo de viagem e a aplicação tem acesso à informação necessária para apresentar o preço final de um dado serviço. Tem ainda ligação a uma impressora para imprimir os recibos de pagamento, a pedido do motorista. Uma vez que podem existir vários clientes no carro, eventualmente com o mesmo destino, um aspeto a considerar é a necessidade de identificar qual o valor que deverá ser cobrado a cada cliente.

Dando particular relevância às heurísticas *Correspondência entre o sistema e o mundo real* e *Reconhecer em vez de recordar*, desenvolva um protótipo para a interface do motorista, sabendo que tipicamente será usada em tablets e que é importante que possa ser utilizada durante a condução do veículo. Explique de forma sucinta de que modo os princípios e os dois requisitos referidos atrás são tidos em conta.

Note que a interface deverá suportar todo o processo, desde a notificação de pedidos de serviço, até à cobração dos mesmos. Lembre-se que podem estar em execução vários serviços em simultâneo.

Questão 10 Considere que foi decidido implementar a sua proposta de interface como uma aplicação Web utilizando Vue.js e responda ás seguintes questões:

1. Sabendo que foram definidas três rotas principais:
   - `/driver/xxxx` (em que `xxxx` é o ID de um condutor) — permite acesso à interface que propôs na questão anterior (implementada pelo componente `DriverUI`, que recebe como prop o ID do condutor a utilizar).

- `/backoffice` — permite acesso à interface do *backoffice*, em que se faz o gestão de condutores, clientes, etc. (implementada pelo componente `BackOffice`).
- `/backoffice/analysis` — permite acesso a um *dashboard* de estatísticas dentro do *backoffice* (implementada pelo componente `DashBoard`).

Apresente a definição do *router* necessário para estas rotas.

2. Defina o componente `MainUI` que permite escolher entre aceder ao *backoffice* ou à interface de um condutor (que deverá ser especificado na interface). Para isso defina:
   - o *template* em que pode ser lido o ID de um condutor e são apresentados os *links* para escolher entre *backoffice* ou interface do condutor com o ID indicado (só deve ser possível aceder a este último *link* se o ID tiver sido indicado);
   - a *script* em que são declaradas as propriedades reativas necessárias para suportar o *template*;
   - o *style* em que é definido que os *links* são apresentados verticalmente na página (é livre de colocar a leitura do ID onde entender).