ISEP – LEEC - Sistemas Computacionais 2017/2018

Ficha 10 – Implementação de serviços *web* tipo REST

Esta ficha tem uma duração estimada de 2 semanas. A resolução do exercício[1] deverá ser entregue em manuscrito e individualmente.

Extraia o conteúdo do arquivo `ficha-ws-ficheiros.zip` para um diretório à sua escolha.

Introdução

Pretende-se alterar a aplicação web desenvolvida na ficha anterior de forma a permitir a utilização de clientes alternativos, nomeadamente a utilização de uma aplicação cliente para execução em linha de comando.

De forma a cumprir esse objetivo, pretende-se que o servidor disponibilize as suas funcionalidades na forma de serviços *web* do tipo *representational state transfer* (REST). Esta interface de programação será implementada com base no código desenvolvido para o *website*, nomeadamente os ficheiros `vehicle.php` e `vehicles.php`. No entanto, no caso do serviço de envio da lista de registos, pretende-se que a lista seja devolvida ao cliente no formato JSON[2], em vez do formato HTML usado para o *website*.

Adicionalmente, pretende-se criar um programa em C para ambiente Linux que permita replicar, com base numa interface em modo de texto, a funcionalidade oferecida pela página *web* desenvolvida na ficha anterior: inserção e listagem de registos. Ou seja, o programa a desenvolver irá constituir uma alternativa à utilização do *browser* como interface de utilização da aplicação *web*.

O esqueleto do programa a desenvolver é fornecido no ficheiro `veic_clt.c`, devendo ser completado com as chamadas ao servidor. Tal como na ficha anterior, cada registo é constituído por três campos: número de matrícula (6 caracteres), nome do proprietário (máximo de 80 caracteres) e valor comercial (valor numérico com um máximo de duas casas decimais). O programa fornecido inclui a definição do tipo de dados estruturado `vehic_t` que deverá ser usado na declaração das variáveis necessárias para manipular os registos.

Implementação

1) Verifique que o *website* que criou na ficha anterior se encontra operacional, através da consulta do seguinte URL:

   http://localhost/~sistc/subdiretorio_do_aluno/

2) Complete o código da opção de inserção no ficheiro `veic_clt.c`. Tal como na implementação do *website*, o serviço de inserção é acedido através do método POST. A inserção de registos deverá ser feita, tal como no formulário do *website*, através do seguinte URL:

   http://servidor/~sistc/subdiretorio_do_aluno/vehicle.php

[1] Pode omitir do manuscrito todo o código aproveitado dos ficheiros fornecidos, desde que devidamente mencionado.

[2] http://www.json.org/

De forma a usar o método POST, a mensagem HTTP a enviar ao servidor deverá ter a forma apresentada no exemplo seguinte:

```
POST /~sistc/subdiretorio_do_aluno/vehicle.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 44
<linha em branco>
plate=11AA22&owner=John Jones&value=10000.00
```

No exemplo acima, o valor 44, indicado para o campo `Content-Length` do cabeçalho HTTP, é o comprimento da *string* contendo a lista de parâmetros deste exemplo (última linha). Tal no caso do exemplo do método GET, tenha em conta que cada linha das mensagens do protocolo HTTP deverá ser terminada com a sequência "`\r\n`".

3) Em caso de sucesso na inserção do registo, o servidor retorna uma mensagem HTTP com o código 200:

```
HTTP/1.1 200 OK
```

Em caso de erro, dependendo do tipo de erro, a resposta do servidor apresentará um código numérico diferente. Adicionalmente, se o erro for detetado no código PHP, a resposta irá conter uma listagem em HTML com a descrição do erro (ver ficheiros `vehicle.php` e `db_functions.php`). A mensagem com a descrição do erro está embebida entre as anotações `<ws-error></ws-error>`[3].

Altere o código da opção de inserção do cliente de modo a notificar o utilizador em caso de erro. Caso a resposta do servidor inclua a anotação `<ws-error>`, o programa deverá imprimir a mensagem correspondente. Essa mensagem pode ser obtida recorrendo à seguinte sequência de instruções:

```
//assume que a resposta do servidor está armazenada na variável char buffer[]
char *ptr,*ptr2;
//procura a tag <ws-error>
if( (ptr = strstr(buffer, "<ws-error>"))!=NULL)
{
    //ptr fica a apontar para o primeiro carácter da mensagem de erro
    ptr = strstr(ptr,">") + 1;

    //função strsep substitui o '<' de "</ws-error>" pelo fim de string
    ptr2 = ptr;
    ptr = strsep(&ptr2, "<");

    printf("Error message from server: \"%s\"\n\n", ptr);
}
```

4) Altere o ficheiro `vehicles.php`, localizado em `~sistc/public_html/subdiretório_do_aluno/`, de forma a enviar os dados em formato JSON (sem qualquer anotação HTML) no caso do parâmetro "o" tomar o valor "JSON". A codificação para o formato JSON pode ser feita através da função PHP `json_encode`[4].

[3] Estas anotações não fazem parte do HTML padrão, foram definidas especificamente para esta aplicação. A definição de anotações "proprietárias" é uma funcionalidade introduzida no HTML5.

[4] http://php.net/manual/en/function.json-encode.php

5) Complete o código da opção de listagem no ficheiro `veic_clt.c`. Tal como na implementação do *website*, o serviço de listagem é acedido através do método GET. Do ponto de vista do cliente, a implementação do método GET deverá ser feita de forma análoga ao exemplificado no programa `http_get` estudado na ficha de *sockets*. O URL terá a seguinte forma:

   `http://servidor/~sistc/subdiretorio_do_aluno/vehicles.php?o=JSON`

   Tal como indicado acima, o servidor envia os dados no formato JSON. Abaixo, apresenta-se um exemplo de uma listagem em JSON contendo a informação de dois registos:

```
[["IJD007","John Doe","100000.00"],["00JJ77","Jane Doe","50000.00"]]
```

   A biblioteca padrão de C não inclui nenhuma função para manipulação de dados no formato JSON. Existem algumas bibliotecas de domínio público para manipulação do formato JSON em programas C. No entanto, tendo em conta a simplicidade do modelo de dados desta aplicação, iremos optar pela implementação de raiz uma função que permita recuperar os valores de cada um dos campos de cada registo e imprimi-los no ecrã, especificamente para esta aplicação.

   Uma possível implementação para a descodificação e apresentação da resposta do servidor é apresentada na Figura 1. Esse código deverá ser executado para cada carácter recebido.

6) Teste as funcionalidades do cliente criado.

Descrição do código da Figura 1:

A rotina apresentada deverá ser usada para analisar sequencialmente cada um dos caracteres da representação JSON do vetor de registos. Esta rotina lê os registos um a um. Durante a leitura, os dados do registo são armazenados na variável `v` (do tipo `vehic_t`). Cada vez que a rotina verifica que terminou a leitura de um registo, chama a função `vehic_print` para fazer a impressão desse registo.

De forma a identificar os valores de cada um dos campos do registo, a rotina procura o carácter usado pelo JSON para delimitar *strings*, nomeadamente as aspas. A rotina usa a variável `state` para gerir essa pesquisa e para identificar o campo que está a ler em cada momento. A variável `state` pode tomar valores inteiros de 0 a 5, indicando os seguintes estados:

- Procura das aspas (início do campo `plate`), no estado 0.
- Leitura do campo `plate`, no estado 1.
- Procura das aspas (início do campo `owner`*)*, no estado 2.
- Leitura do campo `owner`, no estado 3.
- Procura das aspas (início do campo `value`*)*, no estado 4.
- Leitura do campo `value`, no estado 5.

Como pode ser visto, o código para os estados 0, 2 e 4 é o mesmo, pois em ambos os estados o objetivo é encontrar as aspas.
Nos estados 1, 3 e 5, a variável `nchars` é usada para determinar em que posição da variável deverá ser guardado o carácter recebido, assim como para garantir que o número de caracteres armazenados não excede o máximo previsto.
Em todos os estados, a deteção das aspas (abertura de aspas nos estados pares e fecho de aspas nos estados ímpares) provoca a passagem para o estado seguinte. No caso do estado 5, o estado seguinte é o 0.

```
char c; //should contain received character
vehic_t v;
char buffer2[256];
int nchars, state = 0;

    switch(state) {
    case 0:
    case 2:
    case 4:
      if( c == '"' ) { //found opening "
        state++;
        nchars = 0;
        break;
      }

      if( c == '['  && state != 0 ) { //new record before completing current one
        printf("invalid record\n");
        state = 0;
        break;
      }

      break;

    case 1://reading plate
      if( c == '"' ) {  //found closing "
        state = 2;
        break;
      }

      if( nchars == 6 ) //discard unexpected chars
        break;

      v.plate[nchars++] = c;
      break;

    case 3://reading owner
      if( c == '"' ) { //found closing "
        state = 4;
        v.owner[nchars] = 0;
        break;
      }

      if( nchars == VEHIC_MAXPLEN-1 ) //discard unexpected chars
        break;

      v.owner[nchars++] = c;
      break;

    case 5://reading value
      if( c == '"' ) { //found closing "
        state = 0;

        buffer2[nchars] = 0;
        v.value = atof(buffer2);
        vehic_print(&v);
        break;
      }

      if( nchars == sizeof(buffer2)-1 ) //discard unexpected chars
        break;

      buffer2[nchars++] = c;
      break;
    }
```

Figura 1 – Leitura de vetor de registos em formato JSON