Monotorização de Parâmetros Ambientais

André Pereira (pg38923)
Carlos Lemos (pg38410)
João Barreira (a73831)
Rafael Costa (a61799)

Abril 2019

Universidade do Minho
Escola de Engenharia

Trabalho Prático 2
Sistemas Inteligentes – Sistemas Autónomos
MEI / MIEI – Universidade do Minho

1 Introdução

Este trabalho prático explora e incorpora o desenvolvimento de um ambiente inteligente tirando partido da integração de diferentes sensores virtuais num domínio emergente como a *Internet of Things* ou *Smart Cities*.

Para tal foi implementado um sistema inteligente capaz de monitorizar e recolher leituras de sensores virtuais, possibilitando assim uma análise do meio envolvente, através da avaliação de métricas referentes à captura de dados como a temperatura, humidade, entre outros, de uma região.

Para além disso, este sistema é capaz de emitir notificações caso a região que o sistema está a analisar possa tornar-se perigosa para o utilizador, como por exemplo, caso a qualidade do ar atinja um nível demasiado tóxico.

Foi explorada uma arquitetura Cliente-Servidor e incorporado um sistema de notificações na concessão deste sistema. Assim, este ambiente está dividido em duas partes, a parte de *backend* que faz a gestão da recolha de dados através do uso de *APIs*, procede à análise da mesma, e onde é feita toda a gestão do ambiente inteligente de notificações. De modo a que toda a informação recolhida pelos sensores seja facilmente interpretada pelo utilizador, foi desenvolvido uma componente *frontend* para a visualização dos dados capturados de uma determinada região.

2 *APIs* usadas

Como descrito anteriormente, este sistema inteligente procede à recolha de dados através de uma análise a diversos sensores virtuais fornecidos por diversas *APIs*.

Assim, de modo a poder recolher os estados observados pelos diversos sensores virtuais que procedem a uma monitorização de parâmetros ambientais, foram adicionados na componente *backend* três *web APIs*.

Tradicionalmente, a *OpenWeatherMap*, encarregar-se-ia de recolher os dados relativos às condições meteorológicas, tais como a temperatura de uma dada região, mas dado que a *API AirVisual* é capaz de fornecer a mesma informação e ainda dados adicionais, esta *API* foi selecionada para gerir tal recolha dos dados. O único inconveniente desta *API* e da *OpenUV*, é que é necessário introduzir as coordenadas de uma determinada zona para obter os seus dados, ao contrário da *OpenWeatherMap*, que é capaz de facultar a informação introduzindo apenas o nome da cidade.

Portanto, foi utilizada a *OpenWeatherMap* para extrair as coordenadas de uma determinada região. Esta *API* está associada ao motor de pesquisa da aplicação desenvolvida. Isto permitiu uma melhor interação com um utilizador, visto que este apenas necessita de introduzir o nome da cidade que pretende receber informações. A interação com o utilizador é realizada através de uma barra de pesquisa, tal como é ilustrado na seguinte figura:

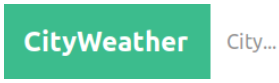

Figura 1: Barra de procura da aplicação *web*

Assim, após o utilizador inserir uma cidade e clicar na tecla *ENTER*, a *OpenWeatherMap* é invocada recebendo como argumento esta cidade de forma a extrair a latitude e a longitude desta região. Assim, passa a ser possível usar as outras duas *APIs* através destes dois parâmetros. O código desenvolvido para este aspeto é o seguinte:

```
const airVisualURL = `http://api.airvisual.com/v2/
    nearest_city?lat=${LATITUDE}&lon=${LONGITUDE}`;

const uvOptions = {
     method: 'GET',
     url: 'https://api.openuv.io/api/v1/uv',
     qs: {
       LATITUDE,
       LONGITUDE
     }
```

A *API OpenWeatherMap* foi utilizada de forma a recolher a maior parte das métricas relativas à região introduzida pelo utilizador na barra de pesquisa. Assim, através desta *API*, o *backend* foi capaz de agregar os seguintes parâmetros ambientais: hora (relativa à informação meteorológica do presente dia), temperaturas atual, mínima e máxima, humidade e uma pequena descrição da condição meteorológica atual. Já a *API AirVisual* foi utilizada para receber informações sobre a qualidade do ar de uma determinada cidade, dando bastante relevância ao atributo *aqi* que apresenta o *Air Quality Index* de uma região. Um resultado deste processo é o seguinte:

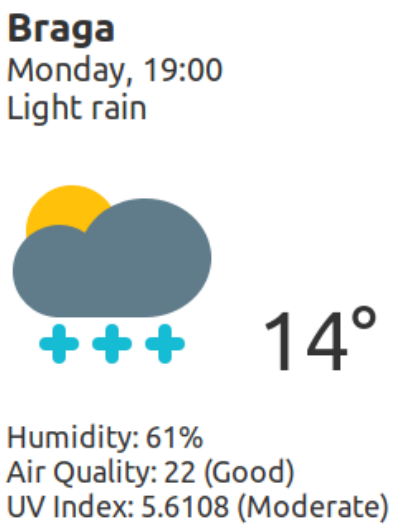

Figura 2: Informação meteorológica

Por último, a *interface* utilizada como apoio de recolha de informação relativa à medição da intensidade dos raios ultravioleta num determinado local foi a *OpenUV*. Com esta *API* o sistema inteligente desenvolvido encontra-se conectado a todos os pontos de recolha de informação útil, sendo assim possível a sustentabilidade de todo este sistema de monitorização ambiental.

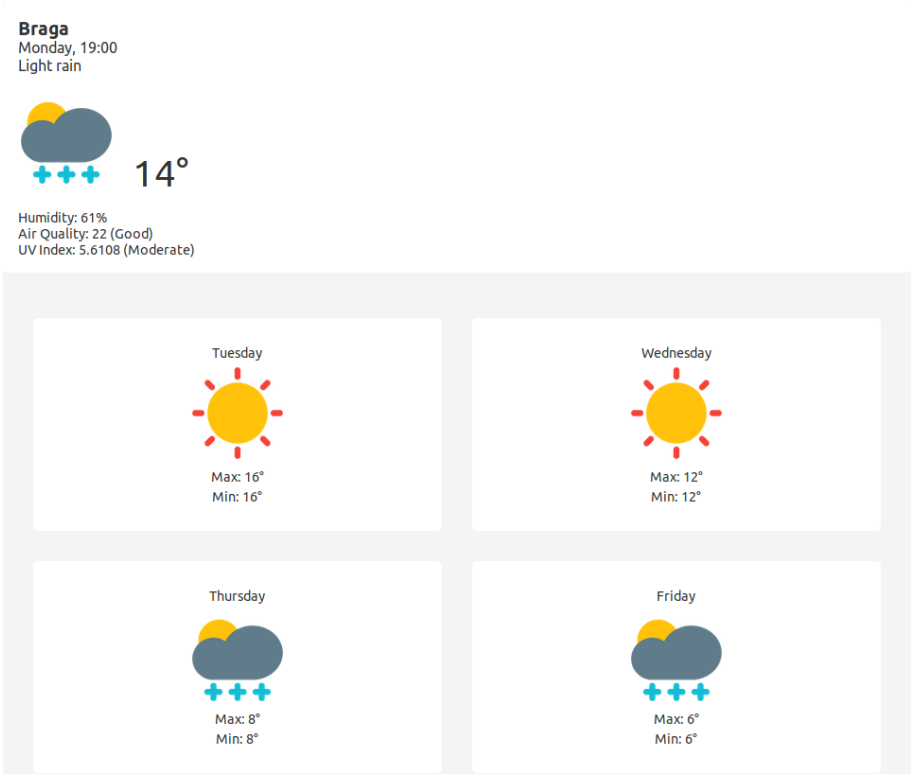

Figura 3: Informação meteorológica com 5 dias

Foi possível, ainda, agregar na aplicação desenvolvida, informação meteorológica relativa à região para cinco dias e ainda adicionar pequenos ícones que simbolizam graficamente o estado do ambiente.

3 Arquitetura do Sistema

A aplicação *web* desenvolvida segue uma arquitetura Cliente-Servidor. A componente do *backend* corresponde a um servidor global que recebe pedidos de todos os possíveis clientes que possam interagir com a aplicação desenvolvida, processando esses mesmos pedidos e enviando as respostas adequadas aos clientes. É no *backend* que se efetuam todas as chamadas às *APIs* descritas na secção anterior. Posteriormente, efetua-se um processamento aos dados recebidos como resposta dessas *APIs*, removendo-se todos os atributos desnecessários, terminando-se com um cruzamento entre todos os dados recebidos para devolver ao cliente. Finalmente, a última função do *backend* consiste em enviar notificações aos clientes sempre que se detetar valores de índices ultravioleta ou de qualidade do ar que sejam considerados como prejudiciais à saúde humana. O servidor do lado do *frontend* é apenas responsável por efetuar a renderização da componente visual da aplicação a um conjunto de clientes. Sempre que um cliente efetua uma pesquisa por uma determinada cidade, este servidor *frontend* reencaminha esse pedido para o servido *backend*, aguardando pela sua resposta. Optou-se por construir esta arquitetura com dois servidores distintos de modo a tornar esta aplicação mais profissional e menos suscetível a ataques já que contém varias camadas de abstração. Este tipo de arquitetura permite esconder ao cliente funções chave como as credencias do servidor do *backend*, assim como todas as chamadas às diversas *APIs* envolvidas. Além disso, permite remover um excesso de carga ao servidor global.

A componente do *backend* foi desenvolvida em *NodeJS*. Foi escolhida esta plataforma de desenvolvimento por ser bastante leve e intuitiva e por ser cada vez mais usada para o desenvolvimento de aplicações *Server Side*, já que é puramente assíncrona. Este assincronismo possui uma grande vantagem, visto não ser necessária a criação explícita de uma nova *thread* para servir um determinado cliente. Finalmente, a plataforma *NodeJS* é bastante utilizada para aplicações *Real Time*, que é o tipo de aplicação desenvolvida já que esta deve ser capaz de enviar ativamente notificações aos clientes. Para facilitar o desenvolvimento da aplicação, foram adicionadas algumas dependências ao servidor *backend*. Uma dependência adicionada foi a extensão *express* de *NodeJS*. Esta extensão permite que se efetue um *routing* de pedidos *REST* de um modo bastante simples e intuitivo. A extensão *request* permitiu concretizar as chamadas necessárias às *APIs* descritas na secção anterior, fornecendo mecanismos para o encapsulamento de todas as credenciais que são requeridas por essas *APIs*. Finalmente, foi adicionada a extensão *web-push*. Esta extensão permite implementar o sistema de notificações do lado do servidor, efetuando o *push* e a manutenção da fila de notificações e a garantindo a entrega de um conjunto de notificações a um determinado cliente.

O *frontend* foi maioritariamente desenvolvido em *VueJS*. Esta *framework* desenvolvida em *javascript* fornece um conjunto de diretivas que tornam a atualização do estado da aplicação bastante mais facilitado. Esta *framework* foi escolhida por ser bastante leve, de fácil uso e por já possuirmos algum conhecimento sobre a mesma. Além disso, tem ganho cada vez mais popularidade

no desenvolvimento de aplicações *web* e permite flexibilidade na sua utilização, por oposição a outras *frameworks* como *react* que dificultam o uso de *scripts* externos. Como extensão a esta *framework* foi utilizado o ecossistema *VueCLi* que incorpora componentes como o *webpack* e permite tirar partido de funções *JSX*, gerando bastante *boilerplate code*. Para a criação de pedidos *http* ao servidor da componente *backend* tirou-se partido da extensão *Vue Resource* que permite a criação deste tipo de pedidos e fornece métodos para aguardar a resposta proveniente do *backend*. Finalmente, desenvolveu-se em *javascript* nativo uma componente chamada *Service Worker*. Esta componente permite que um cliente efetua em registo no serviço de notificações e permite ficar à escuta de notificações provenientes do servidor global. Sempre que recebe uma notificação, é acionado um evento que permite mostrar essa mesma notificação ao cliente recorrendo ao sistema operativo em que este se encontra.

4 Aplicação Desenvolvida

O resultado final consistiu numa aplicação *web* de fácil uso e que ilustra, para uma determinada cidade pesquisada, a meteorologia atual com detalhes acerca da humidade, índice ultravioleta e qualidade do ar, e para os próximos quatro dias ilustra as temperatura máxima e mínima, acompanhadas por uma figura que indica o tipo de clima que se irá presenciar nesses dias. Um exemplo de uma interação com a aplicação desenvolvida é a seguinte:

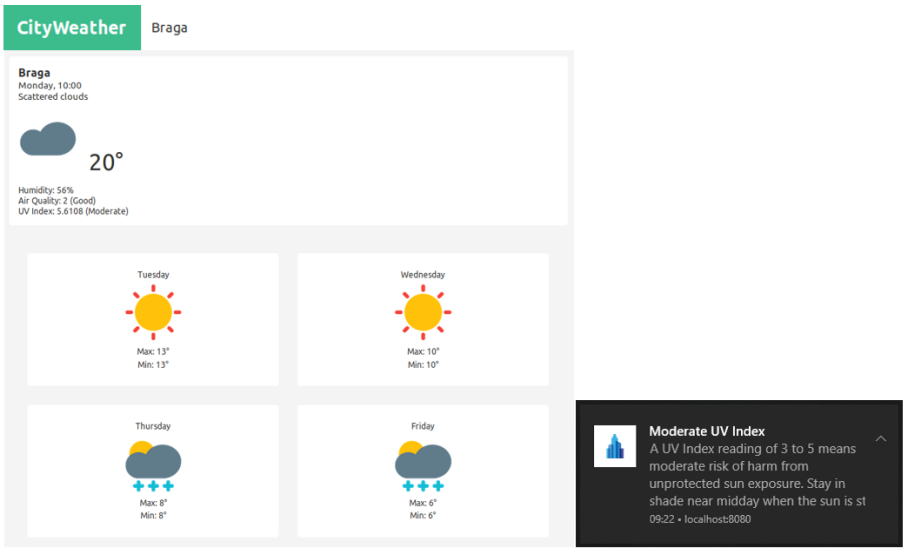

Figura 4: Resultado da pesquisa na aplicação para a cidade de Braga

A figura acima ilustra, também, a receção de uma notificação que indica

o índice ultravioleta é moderado e que aconselha-se às pessoas com problemas de pele a evitar a exposição prolongada ao sol. Os conteúdos fornecidos às notificações foram retirados dos *websites* `https://en.wikipedia.org/wiki/Ultraviolet_index` e `https://airnow.gov/index.cfm?action=aqibasics.aqi`.

Os dados desta aplicação são atualizados de uma em uma hora através da função *setInterval* de *Javascript*. Assim, em cada hora um utilizador pode receber novas notificações a indicar níveis perigosos de qualidade do ar ou de radiação ultravioleta.

5 Conclusões e Trabalho Futuro

A realização deste trabalho permitiu explorar a monotorização de dados a partir de sensores, sendo que neste caso tirou-se partido de sensores virtuais com recurso a três *APIs* distintas. Além disso, permitiu melhorar as nossas capacidades de construção de aplicações *web* baseadas numa arquitetura Cliente-Servidor. Finalmente, exploraram-se inúmeros mecanismos de desenvolvimento de sistemas de notificações, sendo que dentro destes optou-se por desenvolver um sistema de notificações à custa de um *Service Worker*.

Como trabalho futuro, gostaríamos de melhorar principalmente o aspeto visual da componente *frontend*, fornecendo uma interface com mais funcionalidades e mais intuitiva. Além disto, seria útil a criação de mais tipos de notificações e posterior armazenagem numa base de dados para ser consultada pelo utilizador. Tentou-se tirar partido de mais uma *API* que informasse o trânsito para a cidade pesquisada. Esta informação poderia ser ilustrada ao utilizador para que este conseguisse detetar percursos alternativos para escapar ao trânsito numa determinada hora do dia e, poderia ser cruzada com os dados fornecidos pela *API Air Visual* para indicar certas zonas da cidade que possam estar mais poluídas a uma certa hora, cruzando o nível de tráfego existente nessa zona, podendo assim verificar uma possível correlação entre o aumento do tráfego e uma pior qualidade do ar, aí existente. No entanto tivemos que descartar o uso deste tipo de *APIs* por serem bastante complexas e por uma grande parte destas serem pagas.