**Utilizando Node.js para o controle de hardware pela WEB através do protocolo Firmata**



Neste artigo falarei sobre uma aplicação muito interessante que pode ser feita utilizando-se Node.js: o controle de hardware pela WEB. A aplicação que escolhi foi o controle do estado de diversos leds conectados a um Arduino, com o qual o servidor Node.js se comunicará. É uma aplicação extremamente simples do ponto de vista do hardware, mas que servirá para ilustrar os conceitos necessários para a criação de aplicações mais complexas. O código-fonte completo da aplicação pode ser baixado do [repositório no GItHub](https://github.com/amaurigabriel/node-led-controller).

**A Aplicação**

Primeiramente, descreverei em detalhes a aplicação que será desenvolvida. Ela consistirá de dois leds, que poderão ser acendidos ou apagados pelo usuário através de uma interface WEB. Além disso, o usuário poderá monitorar o estado desses leds a qualquer momento. A mesma aplicação poderá ser utilizada por diversas pessoas, e se um outro usuário alterar o estado de um dos leds através de outra instância da aplicação, os demais usuários serão informados disso em tempo real.

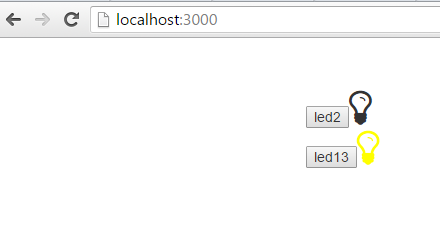
A aplicação é dividida em três partes distintas:

* **Interface do Usuário** – Exibe o estado dos dois leds, e permite que esse estado seja modificado. O estado dos leds é mantido atualizado com o estado físico dos mesmos. Essa camada se comunica diretamente com o servidor Node.js;
* **Servidor Node.js** – Recebe as solicitações de alteração de estado dos leds, e as repassa ao Arduino. Também é responsável por avisar à camada de interface das mudanças ocorridas no estado dos leds em tempo real. Essa camada se comunica tanto com a interface quanto com a camada de Hardware;
* **Hardware**– Consiste dos dois leds mais o Arduino.

A comunicação entre a interface do usuário e o servidor Node.js será feita através da biblioteca [http://socket.io](http://socket.io/), e a comunicação do servidor com o Arduino será feita de maneira serial. Cada uma dessas três partes serão descritas a seguir, de maneira independente.

**Interface do Usuário**

A interface é bastante simples. Cada um dos leds é representado um ícone, que pode estar em preto (led apagado) ou amarelo (led aceso). Além disso, há dois botões de ação. Se um led estiver aceso, e o botão correspondente a ele for acionado, esse led será apagado; caso contrário, ele será aceso. Os nomes dos leds são referentes aos pinos do Arduino aos quais eles estão conectados fisicamente.

Interface do usuário formada por dois leds, cada um deles com um botão de ação associado.

Para a geração do HTML da página, optei por utilizar a linguagem de template jade. É uma linguagem bastante simples, que facilita a geração do HTML através de uma linguagem mais limpa, e que possui uma série de funcionalidades. Essas funcionalidades não serão detalhadas, mas o leitor que sentir necessidade é convidado a ler a [documentação oficial](http://jade-lang.com/reference/). O código abaixo gera todo o HTML da interface.

layout.jade



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | doctype html  html    head      title= title      link(rel='stylesheet', href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css')      link(rel='stylesheet', href='//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css')      link(rel='stylesheet', href='/stylesheets/style.css')    body      div.container          div.content              block content      script(src="https://code.jquery.com/jquery-2.1.4.min.js")    srcipt(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js', type='text/javascript')    script(src="https://cdn.socket.io/socket.io-1.3.5.js", type="text/javascript")    script(src="/javascripts/app.js")      block scripts |

index.jade

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | extends layout    block content    ul.leds      li.led.led2(class=board.pins[2].value == 1? " on" : "off")          button led2          i.fa.fa-lightbulb-o      li.led.led13(class=board.pins[13].value == 1? " on" : "off")          button led13          i.fa.fa-lightbulb-o |

O css presente em style.css é bastante simples, então não vou adicioná-lo ao artigo para não deixá-lo desnecessariamente comprido.

Agora, descreverei o código javascript que será responsável por fazer a comunicação entre o navegador e o servidor Node.js. Essa comunicação será feita através da biblioteca [socket.io](http://socket.io/). Ela permite que uma conexão seja mantida aberta entre as duas partes da aplicação, de modo que qualquer uma delas pode enviar uma mensagem para a outra a qualquer momento.

Essa conexão é estabelecida através do código abaixo.

/javascripts/app.js

JavaScript



|  |  |
| --- | --- |
| 1 | var socket = io('localhost:5555'); |

Nesse código, *localhost* é o endereço do servidor node.js, e *5555* é a porta através da qual as mensagens serão trocadas. Agora, quando um dos botões for pressionado, devemos avisar ao servidor. Isso será feito disparando um evento, que chamarei de *toggle\_led*.

javascripts/app.js

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4 | $('.led button').click(function(){      var led\_number = $(this).closest('.led').is('.led2') ? 'led2' : 'led13';      socket.emit('toggle\_led', led\_number);  }); |

Na linha 3, disparamos o evento *toggle\_led*. Esse evento será “escutado” pelo servidor em tempo real, e terá como parâmetro a variável *led\_number*, que possui o número do led cujo estado deve ser invertido. Por fim, precisamos atualizar o estado dos leds na tela do usuário quando o servidor acender ou apagar algum dos leds.

javascripts/app.js

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | socket.on('changed', function(status){      if (status.led2){          $('.led2').removeClass('on').addClass('off');      }else{          $('.led2').removeClass('off').addClass('on');      }        if (status.led13){          $('.led13').removeClass('on').addClass('off');      }else{          $('.led13').removeClass('off').addClass('on');      }  }); |

Nesse caso, o servidor disparará o evento *changed*, e enviará como parâmetro o estado atual dos dois leds. Assim, cada um dos leds recebe a classe HTML *on* ou *off*, dependendo de ele estar aceso ou apagado. Todos os usuários que estiverem utilizando a aplicação escutarão o evento *changed* quando o servidor modificar o estado de um dos leds. Assim, todos eles terão em sua tela o estado atualizado dos leds.

**Servidor Node.js**

Apesar de se uma aplicação simples, eu optei por utilizar o framework [Express](http://expressjs.com/). Desse modo, se eu quiser fazer algo mais complexo futuramente, não haverá grandes dificuldades. Como eu já expliquei o básico sobre esse framework em outro artigo, vou explicar aqui apenas a parte referente à aplicação deste artigo.

Primeiramente, adicione os pacotes *socket.io* e *johnny-five*à sua aplicação. O pacote *socket.io* já foi explicado anteriormente, e o *johnny-five* é quem fará a comunicação com o Arduino. Abaixo, é feita a  inicialização dos leds e da comunicação com o Arduino .

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | var five = require('johnny-five');    var board = new five.Board({      port : ARDUINO\_PORT  });    var leds = {};    board.on("ready", function() {    leds.led2 = new five.Led(2);    leds.led13 = new five.Led(13);      //escreve 0 nos pinos 2 e 13 do Arduino    //para inicializar os leds    this.digitalWrite(13, 0);    this.digitalWrite(2, 0);      ioChanged();  }); |

Na linha 4, devemos informar a porta serial à qual o Arduino se encontra conectado. Na linha 7, o evento *ready* é disparado pela variável *board* após a comunicação ter sido estabelecida com a placa. Por fim, após a inicialização dos leds, a função *ioChanged()*avisará aos usuários da aplicação que o estado dos placa mudou, disparando o evento *changed* para a camada de interface do usuário. Essa função será apresentada a seguir.

Para permitir a comunicação em tempo real entre o servidor e a camada de interface, devemos inicializar o pacote socket.io, como a seguir.

JavaScript



|  |  |
| --- | --- |
| 1 | var io = require('socket.io')(5555); |

Assim, quando um usuário se conectar ao servidor através da porta 5555, a função a seguir será executada.

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | io.on('connection', function(socket){      if (board.isReady)          socket.emit('changed', getBoardStatus());        socket.on('toggle\_led', function(led\_number){          leds[led\_number].toggle();          ioChanged();      });  }); |

Primeiramente, verificamos se a comunicação entre o servidor e o Arduino já foi estabelecida. Em caso afirmativo, o estado dos leds é enviado para o usuário que acabou de se conectar ao servidor através do evento *changed*. Na linha 5, sempre que esse usuário gerar o evento *toggle\_led* para o servidor, o estado físico do led correspondente é alterado (linha 6), e, então, todos os usuários que estejam usando a aplicação no momento são avisados disso em tempo real (linha 7).

Por fim, segue a implementação da função ioChanged().

JavaScript

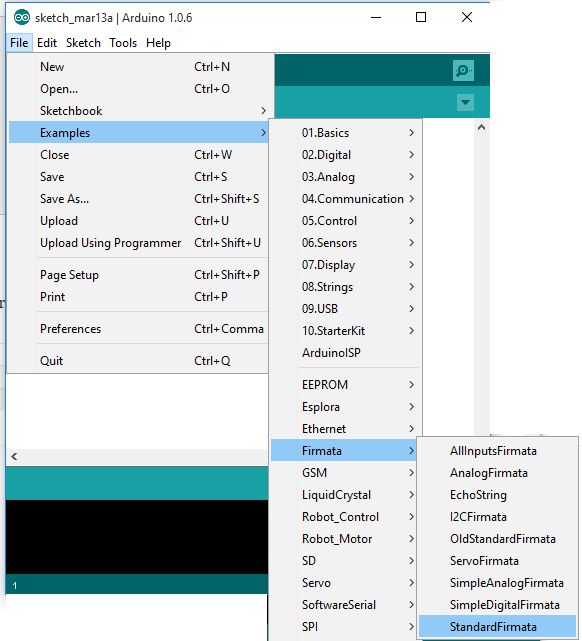


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | function getBoardStatus(){      return {          led2 : leds.led2.isOn? 1 : 0,          led13 : leds.led13.isOn? 1 : 0,      };  }    function ioChanged(){      io.sockets.emit('changed', getBoardStatus());  } |

**Camada de Hardware**

Muito pouco precisa ser feito na camada de hardware para que a comunicação com o servidor seja feita. O pacote *johnny-five*, responsável por fazer essa comunicação, utiliza o protocolo *Firmata*. Sendo assim, precisamos gravar um programa no Arduino que seja implemente esse protocolo. Felizmente, a IDE do Arduino já vem com um exemplo que implementa o protocolo *Firmata*. Assim, só precisamos gravá-lo!

Para isso, vá em *File* -> *Examples* -> *Firmata* -> *StardardFirmata*, como mostra a Figura abaixo.



Feito isso, basta gravar o programa no Arduino, e a comunicação estará funcionando! Agora, basta conectar os leds aos pinos 2 e 13 do Arduino, para que a aplicação esteja finalizada.