Introdução

Este artigo tem por objetivo abordar uma estratégia para acionar um relé de 5v via internet, usando o poder da Web em *Real-time* implementando o socket.io, utilizando para isso uma arquitetura com dois servidores rodando Node.js, sem a necessidade do uso de VPN ou No-ip. A estratégia visa fugir um pouco dos exemplos de chat básico, tudo isso através do Node.js.

Para isto, vamos trabalhar com o VS Code, editor de texto fantástico da Microsoft, explorando suas ferramentas de desenvolvimento.

Abaixo seguem algumas definições dos termos e tecnologias explorados neste artigo.

Nota: O presente artigo não contempla a implementação de mecanismos de segurança e/ou performance ficando esse à cargo do leitor, ou à outro fórum.

Node.js  
(<http://nodebr.com/o-que-e-node-js/>)

[Node.js](http://nodejs.org/) é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis. Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

Na JSConf 2009 Européia, um programador jovem chamado Ryan Dahl, apresentou um projeto em que estava trabalhando. Este projeto era uma plataforma que combinava a máquina virtual [JavaScript V8](http://code.google.com/p/v8/) da Google e um laço de eventos. O projeto apontava para uma direção diferente das outras plataformas em JavaScript que rodam no servidor: todos I/O primitivos são orientado a evento. Aproveitando o poder e a simplicidade do Javascript, isso tornou tarefas difíceis de escrever aplicações assíncronas em tarefas fáceis. Desde quando foi aplaudido de pé no final do seu discurso, o projeto de Dahl tem recebido uma popularidade e uma aprovação sem precedentes.

Socket.io

(<http://udgwebdev.com/real-time-com-socket-io-no-nodejs>)

Na prática o **Socket.IO** abre uma conexão com servidor pelo qual possibilita realizar troca de mensagens entre ambos sem a necessidade de dar um refresh na página, ou seja, sem a necessidade de solicitar uma nova request para o servidor. Com isso é possível explorar o conceito de **real-time** em uma aplicação, pelo qual basicamente o cliente envia uma mensagem e o servidor processa e responde via **broadcast** (para todos os clientes de conexão aberta com servidor) através de um **long pooling de conexão**.

**Arduino**

(<https://projeto39.wordpress.com/o-arduino/>)

Arduino é uma ferramenta para criar computadores que podem sentir e controlar mais o mundo que seu PC. Ele é uma plataforma física de computação de código aberto baseado numa simples placa microcontroladora, e um ambiente de desenvolvimento para escrever o código para a placa.

**Heroku PaaS**

(<http://www.devmedia.com.br/primeiros-passos-em-paas-com-heroku/29465>)

O Heroku se enquadra na categoria de serviços da computação em nuvem conhecida como Plataforma como Serviço (Platform as a Service, ou PaaS), no qual o fornecedor entrega para o cliente um ambiente pronto para receber a aplicação. Diferente do IaaS (Infraestrutura como Serviço), no qual cliente contrata máquinas (reais ou virtuais) e é responsável pela instalação de bibliotecas, montagem das estruturas do sistema de arquivos, entre outros recursos, o PaaS é uma solução de alto nível que abstrai este tipo de preocupação.

Até a data da publicação deste artigo o Heroku fornecia um palno free, limitado, mas suficiente para atender nossa proposta.

Aqui tem um ótimo artigo sobre deploy no heroku:

<http://www.pinceladasdaweb.com.br/blog/2013/03/01/como-fazer-deploy-de-apps-para-o-heroku/>

O que vamos precisar:

Hardware:

* Arduino Uno
* Notebook \*
* Componentes como jumpers, led, protoboard.

\*Neste artigo vou utilizar Arduino com um Notebook como Server, na próxima versão pretendo abordar a implementação no Raspberry Pi 2 eliminando a necessidade de um notebook.

Software:

* Node.js (<https://nodejs.org/en/>) – Para rodar localmente.
* Git (<https://git-scm.com/>) – Para fazer o clone do projeto e deploy no heroku
* Drives do Arduino UNO – (<https://www.arduino.cc/en/Main/Software>)

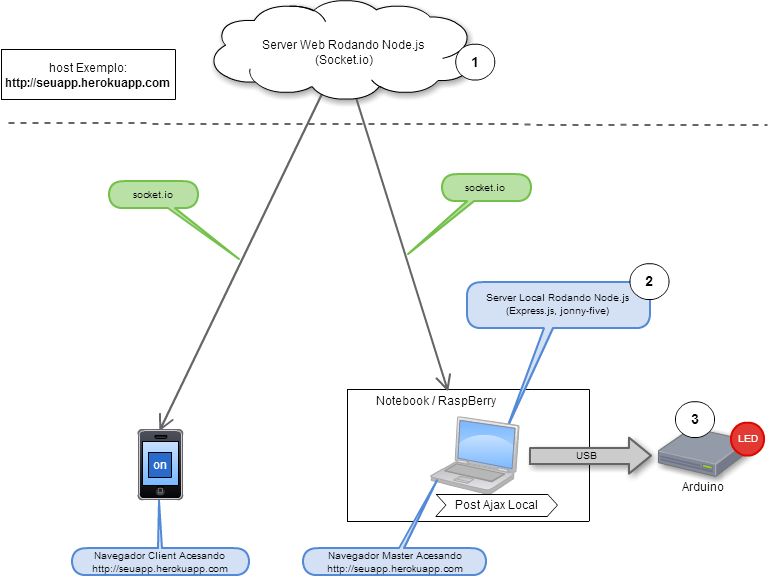
Serviço:

* Uma conta no heroku ou em outro PaaS que tenha compatibilidade com Node.js

Tags Tecnologias:

#JavaScript, #JQuery, #TwitterBoostrap, #Node.js, #Express.js, #Socket.io, #PaaS, #Jonny-five, #Arduino, #Git.

Diagrama da proposta:



1 - ServerWeb:

Instale o Git.

Faça um clone do projeto no github:

> git clone socket-bridge-iot

Com a conta heroku criada iremos montar uma nova app:

> heroku create socket-bridge-iot

Adicione a referência ao repositório do Heroku

> git add remote heroku https://git.heroku.com/socket-bridge-iot

Na constante *adressWeb*. Arquivo (serverWeb\server.js)

Deixe: const adressWeb = 'http://socket-bridge-iot .herokuapp.com/';

2 - ServerLocal:

Na parte local vamos subir a app que fará a comunicação com o embarcado:

> node serverLocal.js

Configurei para escutar a porta 3012, note que essa porta é configurada na app serverWeb

Na constante *adressLocal*. Arquivo (serverWeb\server.js)

3 - Arduino:

Instale os drivers do arduino no notebook, neste projeto em especial, acenderemos um LED de 3,5v e um relé de 5v, que acionará uma lâmpada. Conecte o Arduino no notebook.

Monte o esquema conforme imagem abaixo:

Considerações finais e comunidade:

Neste artigo vimos que é possível implementar comandos remotos utilizando esta fantástica ferramenta que é o socket.io.

O projeto está disponível no github e toda colaboração será bem-vinda.