**Projeto Interativo 7**

**ReAct**

**Centro Universitário Senac**

**Daniel Gomes dos Santos**

**David Kumagai Pedro**

**Francini de Assis Silva**

**Henrique Lima**

**Nicolas Fidel**

**Thais Lima**

**Engenharia da Computação**

**Prof. Eduardo Heredia**

**São Paulo, 13 de Junho de 2017**

**Introdução**

O projeto tem como base, a separação em quatro módulos, sendo cada um, um projeto interativo, da respectiva turma que está realizando o projeto interativo. Essa documentação demonstra todo o processo desenvolvido no segundo módulo do projeto total, que se dá através da construção de uma documentação de API Restful sólida, podendo a mesma ser exportada para outras linguagens, além de uma pesquisa aprofundada para criar uma base de requerimentos de Hardware, especificamente nos sensores que poderão vir a ser utilizados no projeto, além de algumas aplicações já consumindo dados da API, e a mesma já conectada com um banco de dados relacional, já modelado para implementação.

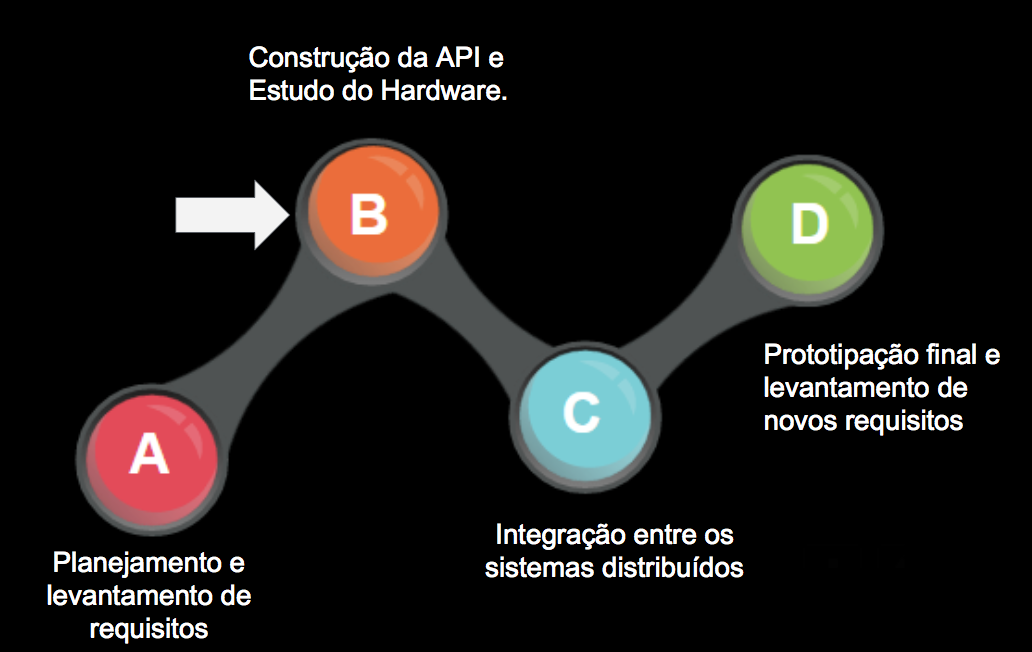
A partir da documentação e do levantamento de requisitos feitos no projeto anterior, iniciamos a construção e documentação da API, juntamente da modelagem do banco de dados. Paralelamente, começamos uma pesquisa mais aprofundada com os sensores, para melhor performance do projeto.

A idéia foi exportar para esse artigo, todo o conteúdo, do segundo módulo do projeto, para facilitar a continuação do projeto.

**Módulos**

O projeto tem como base, no mínimo quatro módulos, sendo eles:

* 1. Planejamento e levantamento de requisitos: A idéia desse módulo era entender a necessidade do projeto, e o levantamento de requisitos básicos para criar uma arquitetura básica do projeto.
* 2. Construção e modelagem da API, e estudo do Hardware: Modelar e documentar a API, a partir dos requisitos necessários, além de um estudo aprofundado dos diferentes tipos de sensores que recebemos.
* 3. Integração entre os sistemas distribuídos: Integrar a base da API, com o sistema distribuído, gerando o fluxo da informação, e os requisitos mínimos de funcionamentos do sistema como um todo.
* 4. Prototipação final e novos levantamentos de requisitos: A partir da integração re-validar o levantamento de requisitos, para imprevistos, e prototipação do sistema, com layout do hardware, e ajustes na integração.

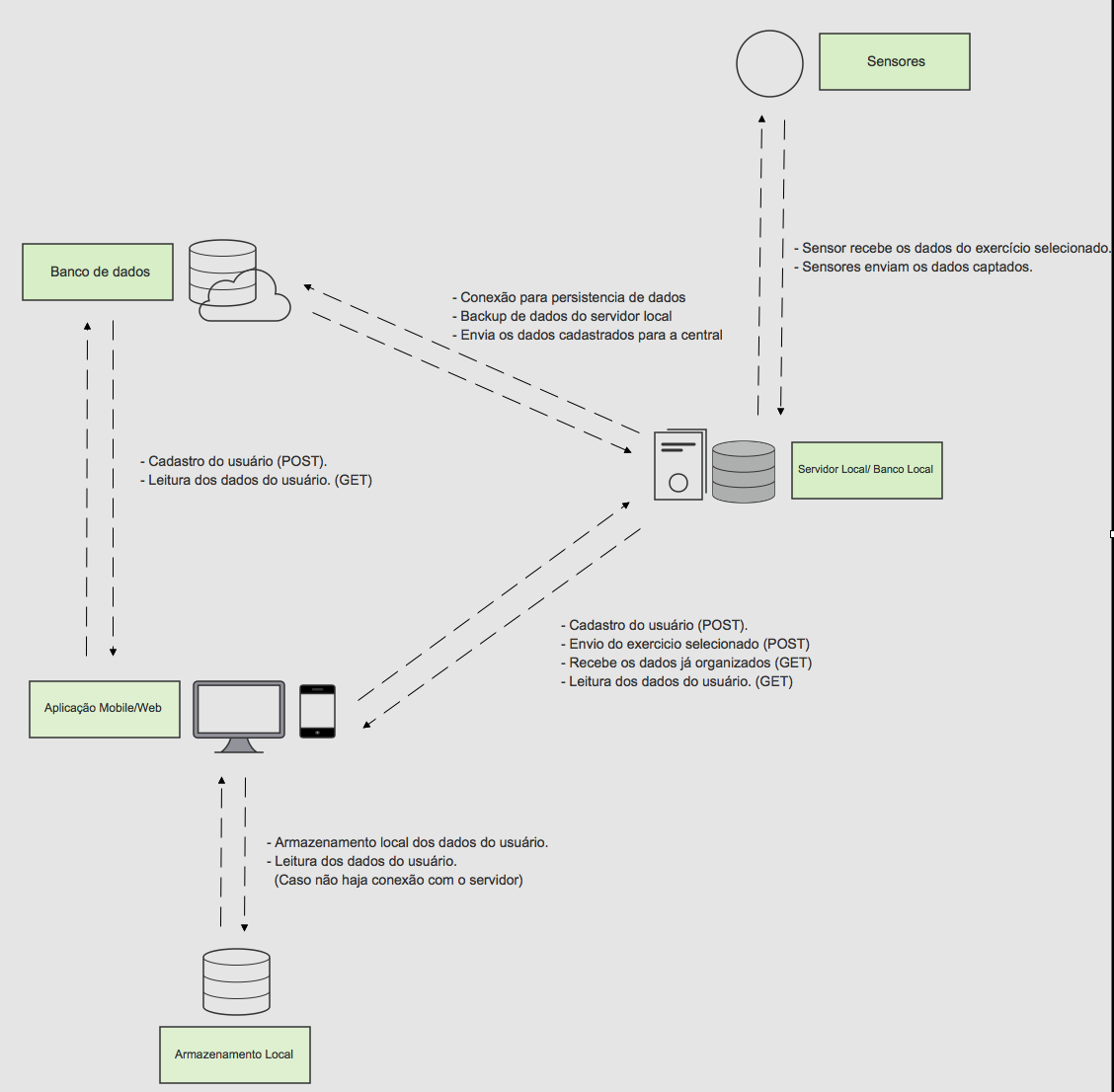


**Arquitetura**

A base de arquitetura demonstrada nesse sistema em seus pilares, demonstra apenas o modo com que a informação percorrerá pelo sistema arquitetado, a idéia não é relatar o método exato, mas o caminho ideal, podendo ser alterado, caso haja alguma mudança de protocolo de conexão, ou de mudanças futuras.

Ela demonstra, em que momento a informação deixa o sistema para uma maior persistência de dados em nuvem, em que momento haverá um sistema de autenticação, em que momento haverá conexão com a API, e toda a base necessária para o entendimento dos pilares do módulo 2 do projeto.

Segue abaixo uma imagem representativa da arquitetura do sistema:



**Funcionalidades do Sistema**

A partir da arquitetura, chegamos a seguinte idéia, para funcionalidades básicas do sistema, sabemos que cada parte da aplicação deve rodar, para que a aplicação seja simples e intuitiva para o usuário, sendo assim, precisamos ser precisos ao simplificar a aplicação e suas requisições como um todo.

Pensando nisso, chegamos a uma idéia de que todo o sistema deveria se basear nos seguintes pilares de desenvolvimento; Análise de exercício, criação de exercício, autenticação e banco de dados, sendo esses os requisitos mínimos para o funcionamento da base de informação de forma satisfatória.



**API**

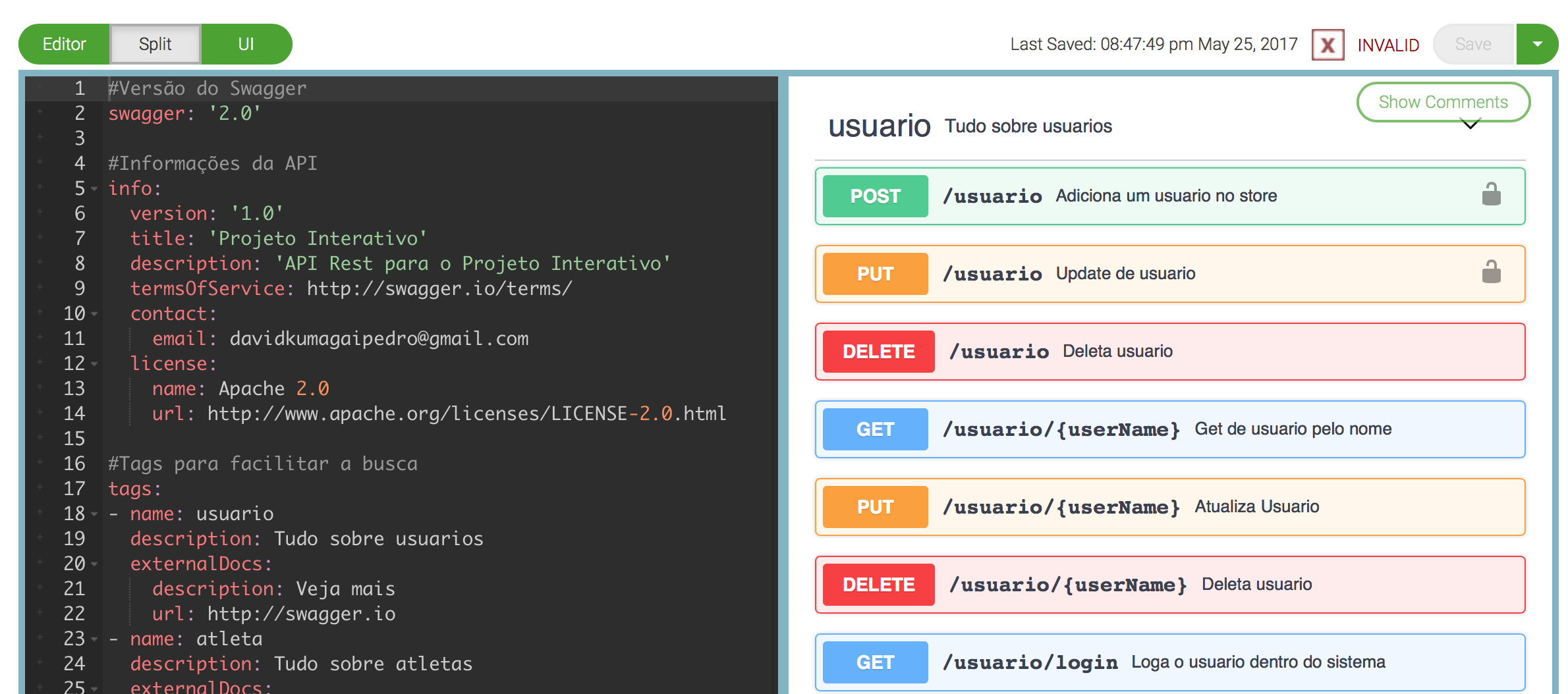
A API foi documentada no swagger, podendo assim ser exportada para determinadas linguagens delimitadas pelo mesmo, sendo assim, uma solução mais prática para o desenvolvimento e expansão futura, pela maior praticidade da escolha da linguagem que o respectivo desenvolvedor que futuramente consumir os dados da API.

Ela tem como base "Models" para uma melhor arquitetura da informação já que os dados estarão no mesmo requisito básico requerido para recebimento, assim como os verbos para cada requisição da API, já estão documentados e validados com o banco de dados, a construção foi feita para que a exemplificação fique simples, para que caso haja necessidade de expansão, haja exemplos dentro da própria documentação, facilitando assim, o desenvolvimento de um futuro desenvolver BackEnd do sistema.

Ela utiliza basicamente quatro verbos de requisição do protocolo HTTP, sendo eles GET(Requisita os dados), PUT(Atualiza os dados), POST(Insere os dados) e DELETE(Apaga os dados).

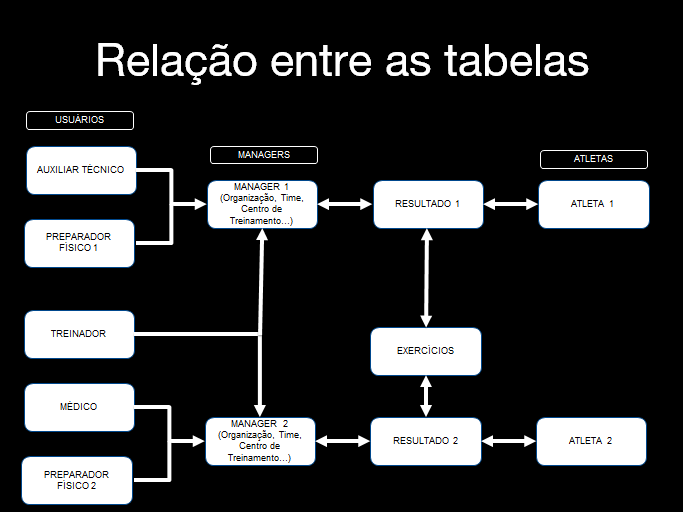
Base da API no Swagger: <https://app.swaggerhub.com/apis/DavidKumagai/ProjetoInterativo/1.0>

Segue imagem para visualização da interface:



**Banco de dados**

Nosso projeto tem o intuito de cadastrar atletas, organizações (Managers) e usuários que são profissionais ligados há uma organização. O propósito disso é que os resultados possam ser acessados tanto pelas entidades de organizações quanto pelos atletas. Isso faz com que possamos criar uma base completa de um atleta que acompanha o atleta desde de o início de sua carreira, os resultados não serão restritos ao dispositivo utilizados, resultados pertencem tantos aos atletas quanto às associações onde foram obtidas, assim permitimos que a organização (um time por exemplo) possa construir sua própria base de dados com os atletas que já passaram por ela. Usuários podem acessar os resultados através da organização ao qual estão vinculados, e eles podem estar vinculado a ‘n’ organizações. Dessa forma garantimos que exista um cadastro único para cada uma dessas entidades.

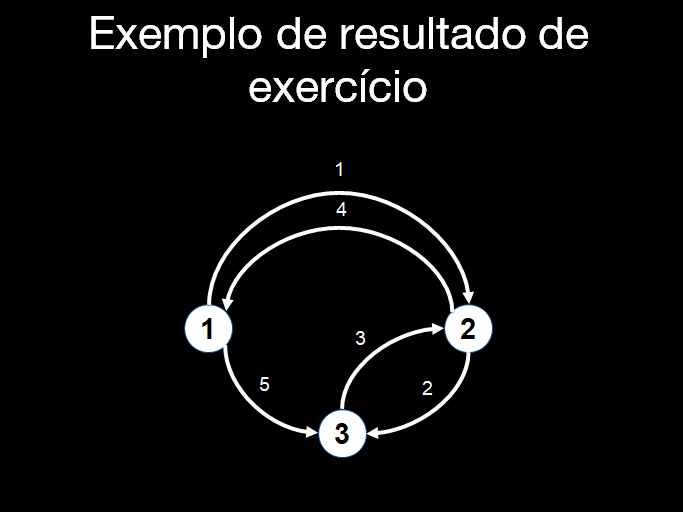


Exemplo da relação entre as entidades.

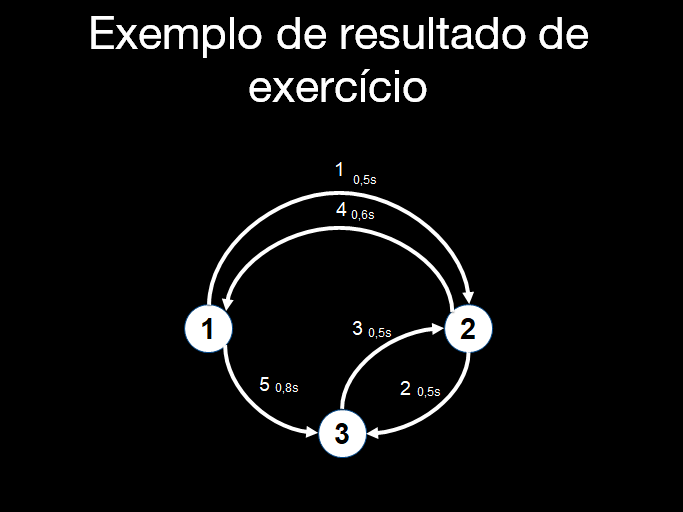


Modelo relacional de testes.

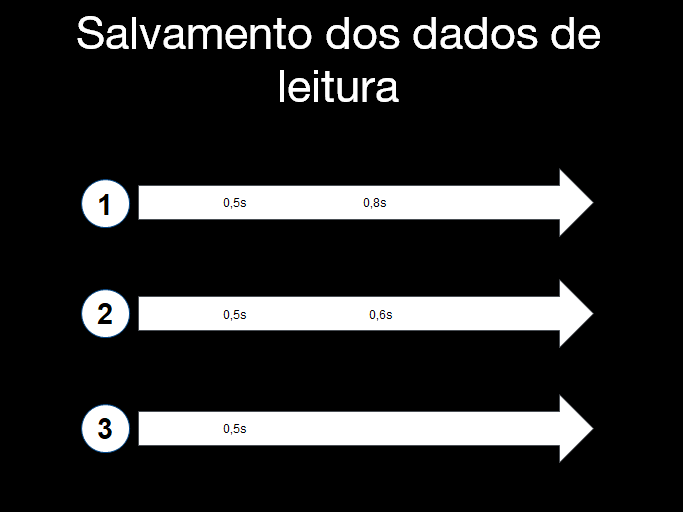
Para salvar os dados lidos, consideramos que cada dispositivo guarda uma String com os tempos lidos. Para o modelo de testes consideramos que existem três dispositivos diferentes lendo os tempos do atleta, ao final do exercício teremos três Strings com os tempos obtidos por cada dispositivo. A raspberry sabe a ordem de ativação do dispositivo e por isso teremos uma quarta String que guarda a ordem de ativação desses tempos.



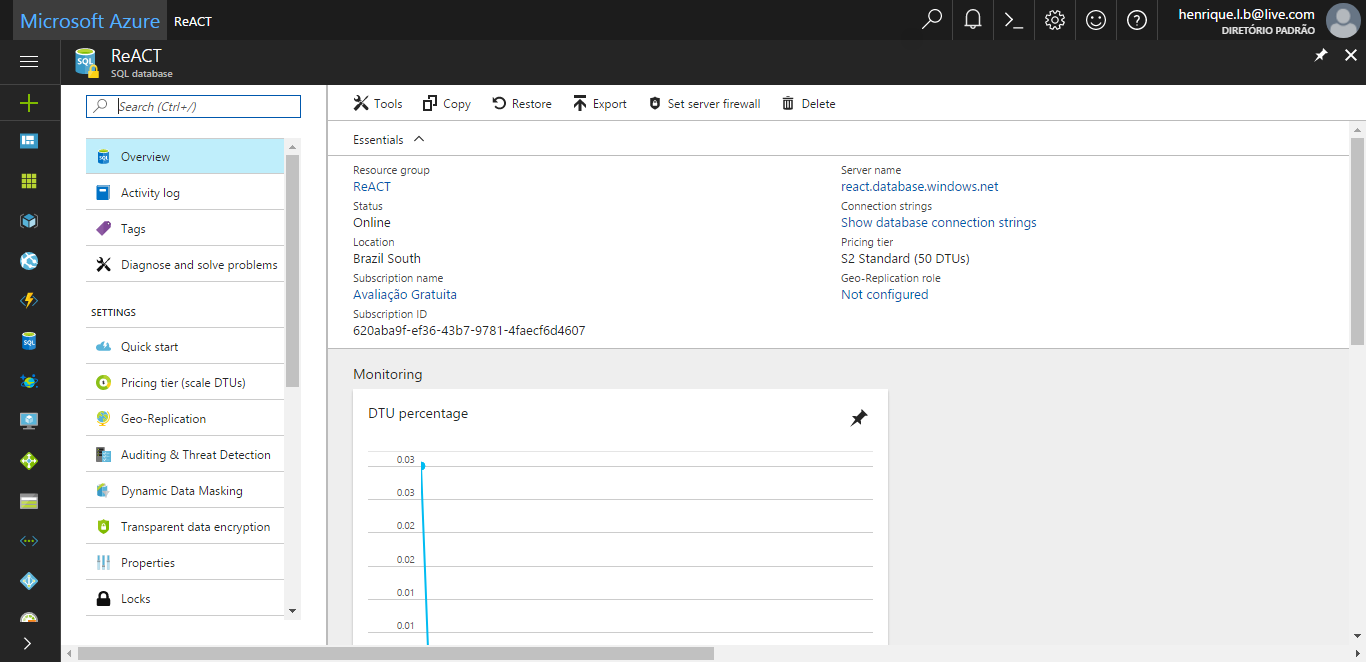
Ordem de ativação dos dispositivos.



Tempo de resposta em cada passo.



Strings de cada dispositivo.



Para realizar testes implementamos o modelo acima na plataforma Microsoft Azure.

**Layout**

O layout foi pensado a partir da idéia de imagens representativas, para entendermos o meio que o produto se encontrará ou seja um painel referencial(para entender a referência do que já existe nesse meio) e um painel conceitual(para entender o meio que essa aplicação se encontra), além de uma tipografia moderna, pensada a partir da modernização de sensores e conexões:

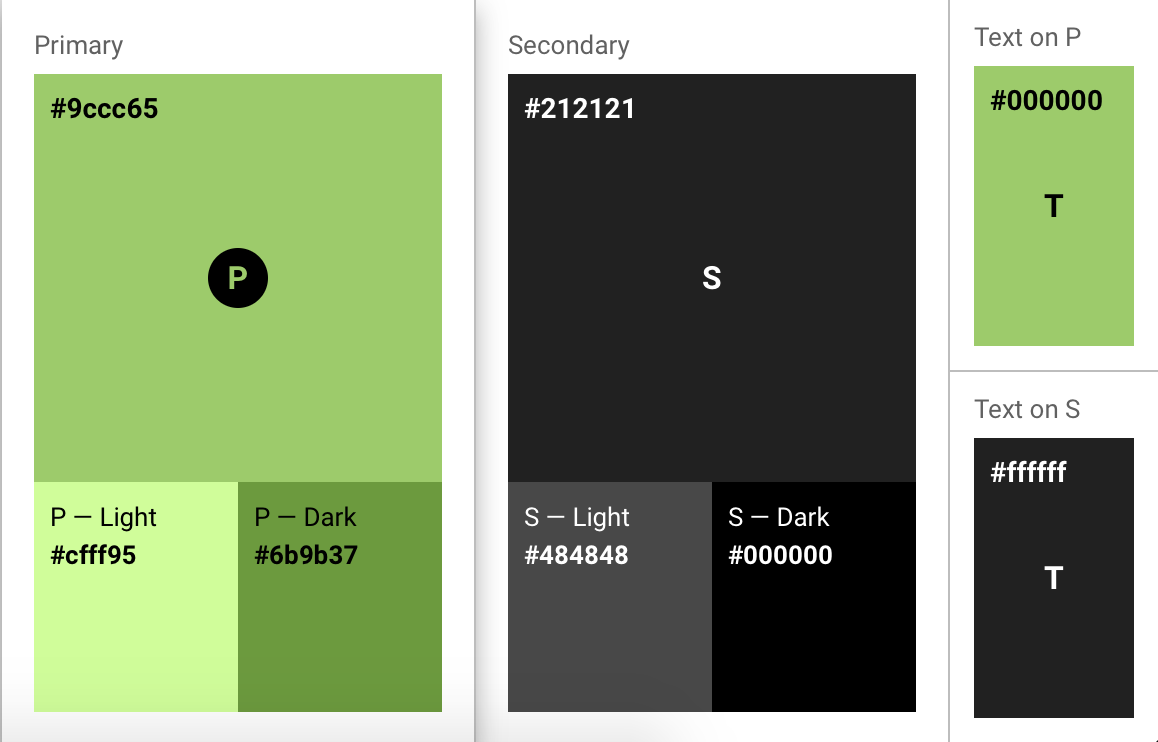


Painel Conceitual

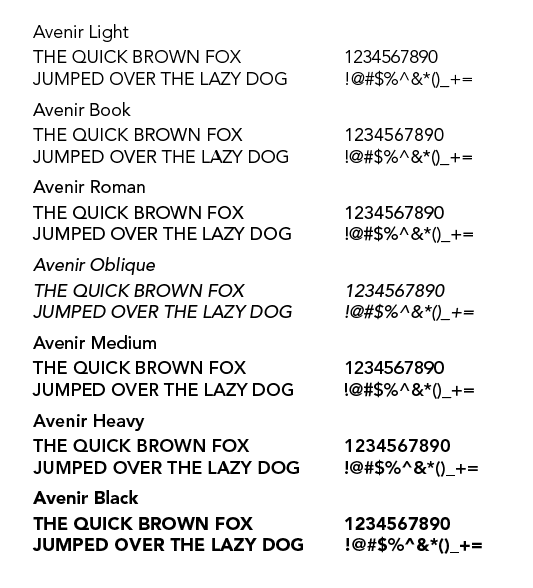


Painel Referencial

Essa idéia nos leva a um protótipo da paleta de cores a ser usada na aplicação com a tipografia Avenir, pela mesma ser moderna e atual.



Paleta de Cores



Tipografia

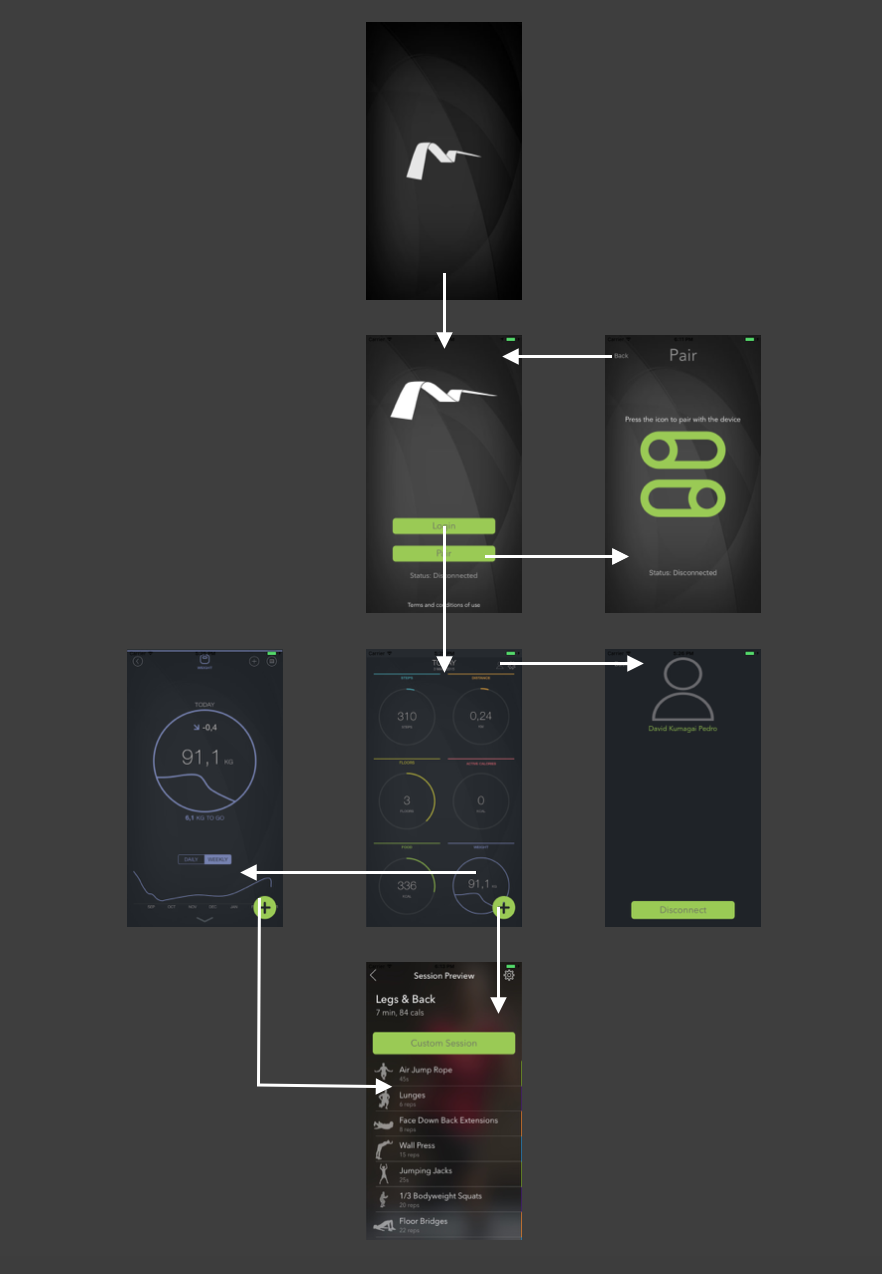
**Aplicação Mobile**

A aplicação mobile foi desenvolvida em Swift 3.1, com modelos já criados pela API exportados para a aplicação, e com validação de dados, a partir de algumas requisições no Firebase, para facilitar a validação, das requisições, e da autenticação. Além disso, foram prototipados alguns "MockUps", para validação do design, e do fluxo do sistema,

segue abaixo os protótipos:



Mockups, do projeto



Fluxograma da aplicação

**Aplicação Front-end**

O objetivo desta aplicação é mostrar o consumo do front-end de um dos módulos criados na API utilizando angular 2 e persistindo os dados no banco postgresql.

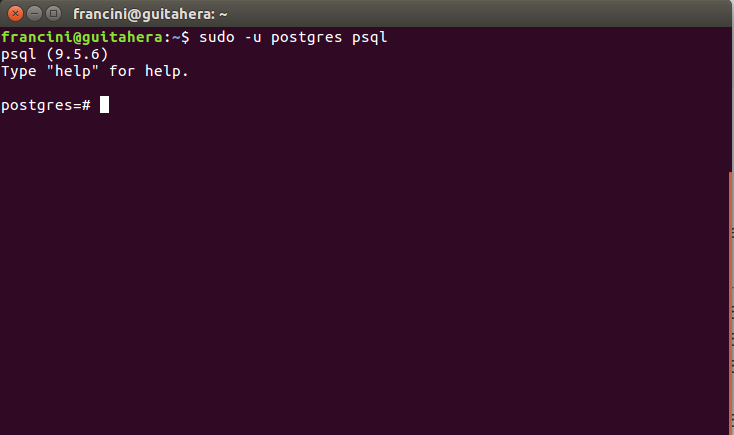
A API foi gerada na linguagem Nodejs para demonstrar o consumo da API utilizando-se outra linguagem.

Para rodar a aplicação que ficará disponível no git hub é necessário seguir os seguintes passos:

Instale o banco postgresql no seu computador e logo após esse passo navegue até o banco utilizando o terminal com este comando:

**sudo -u postgres psql**

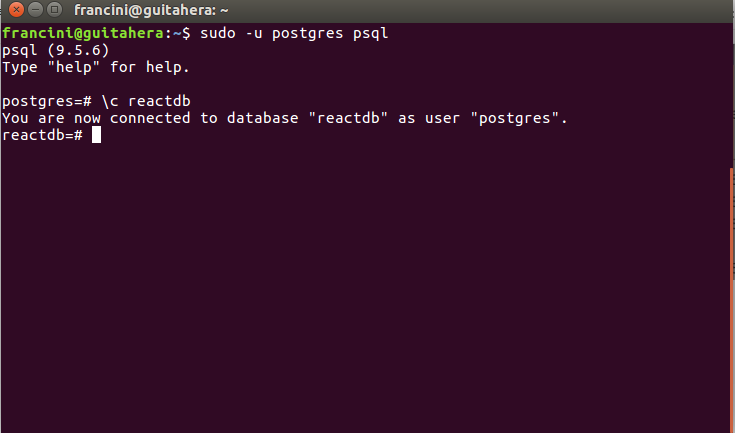
Aparecerá a seguinte resposta no terminal:



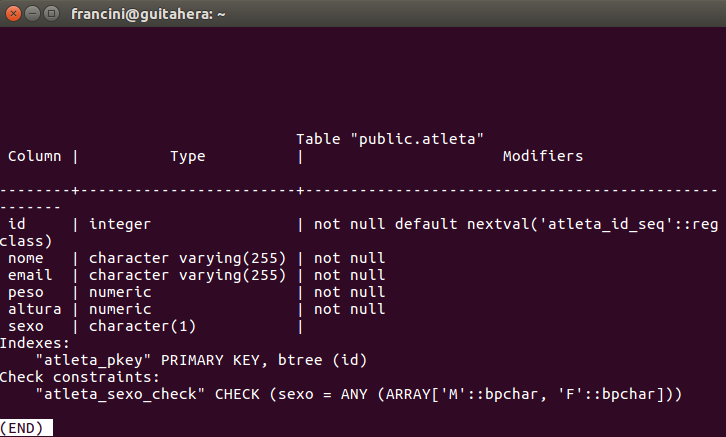
Crie uma base de dados chamada **reactdb** com o seguinte comando:

**CREATE DATABASE reactdb**

Para conectar-se a base de dados, utilize o \c passando o nome da base:



Crie uma tabela com a seguinte estrutura:

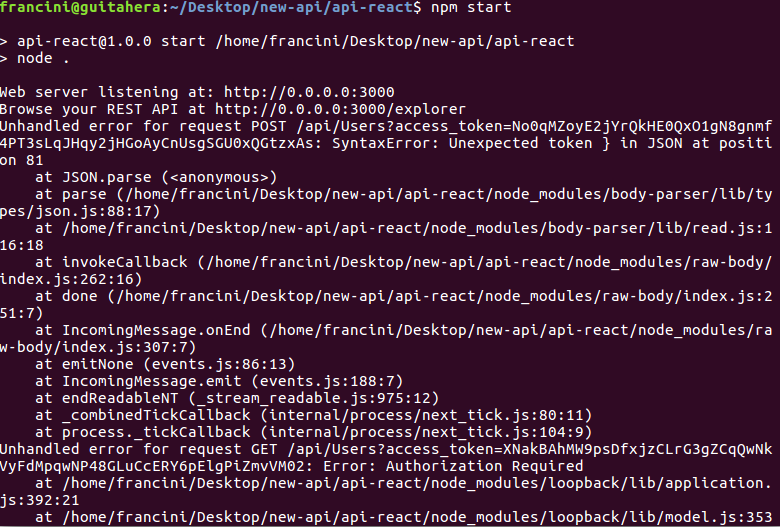


Siga para o pŕoximo passo:

Navegue até a pasta api-react através do terminal do S.O e insira o seguinte comando para iniciar a api:

**npm start ou node .**

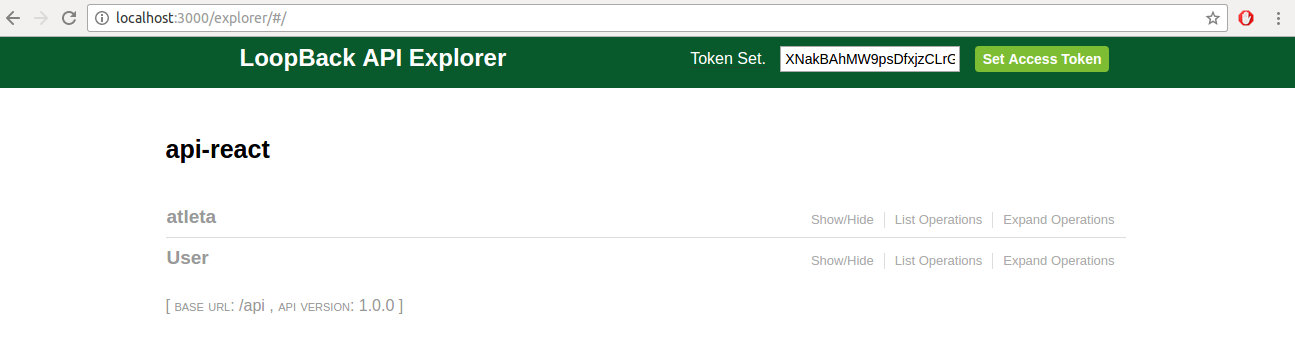
Aparecerá a seguinte resposta no terminal:



Abra uma janela do navegador e insira o seguinte caminho:

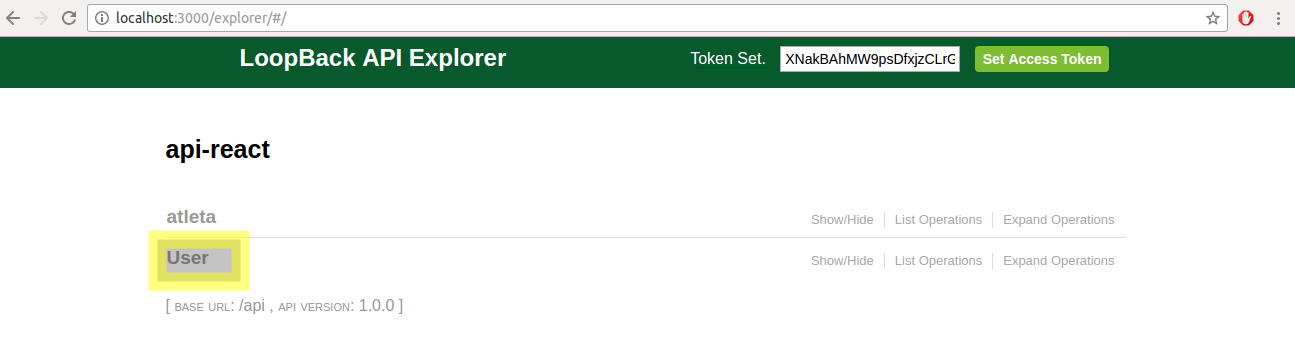
[**http://localhost:3000/explorer/**](http://localhost:3000/explorer/)

Esta será a tela visualizada:

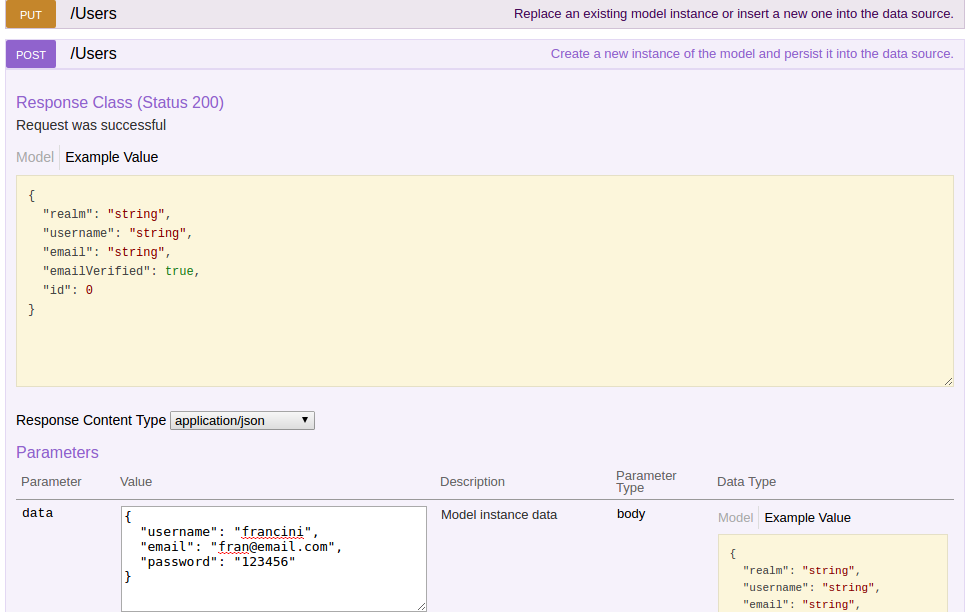


A API possui permissão de usuário, com isso, é necessário criar um usuário antes de acessar o layout da aplicação. Para criar um usuário siga os seguintes passos:

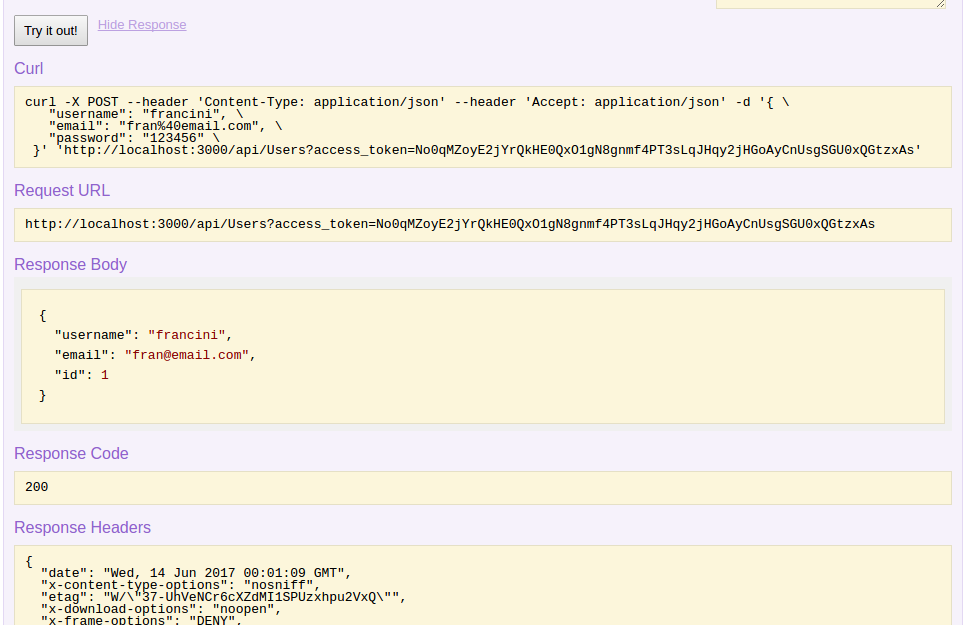
Clique em user para visualizar os verbos http disponíveis para criar o usuário:



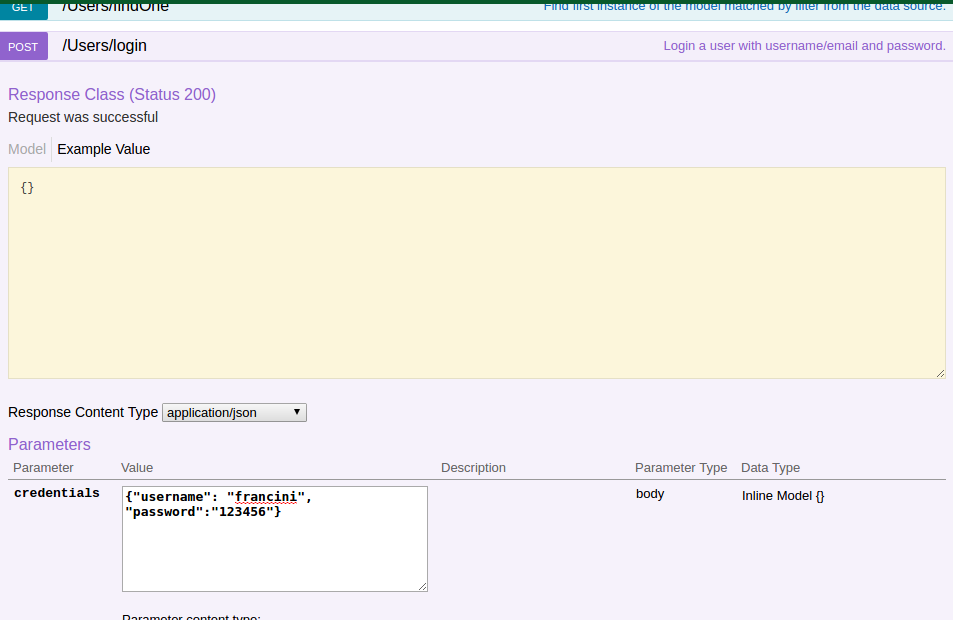
Navegue até o verbo post do user e insira um usuário(username), um email e uma senha(password)



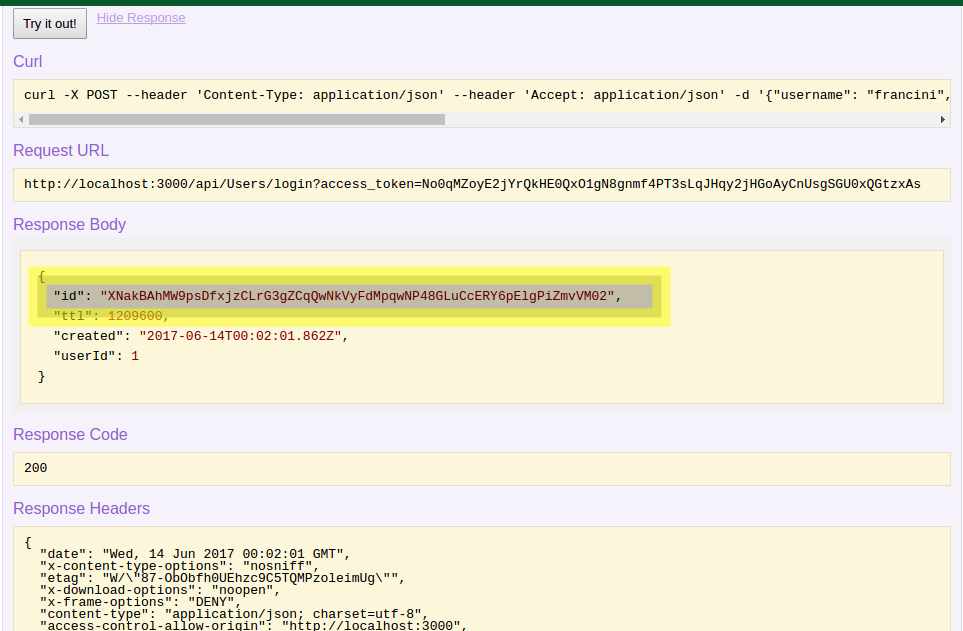
Clique em **Try it out** e aparecerá a resposta com o cabeçalho:



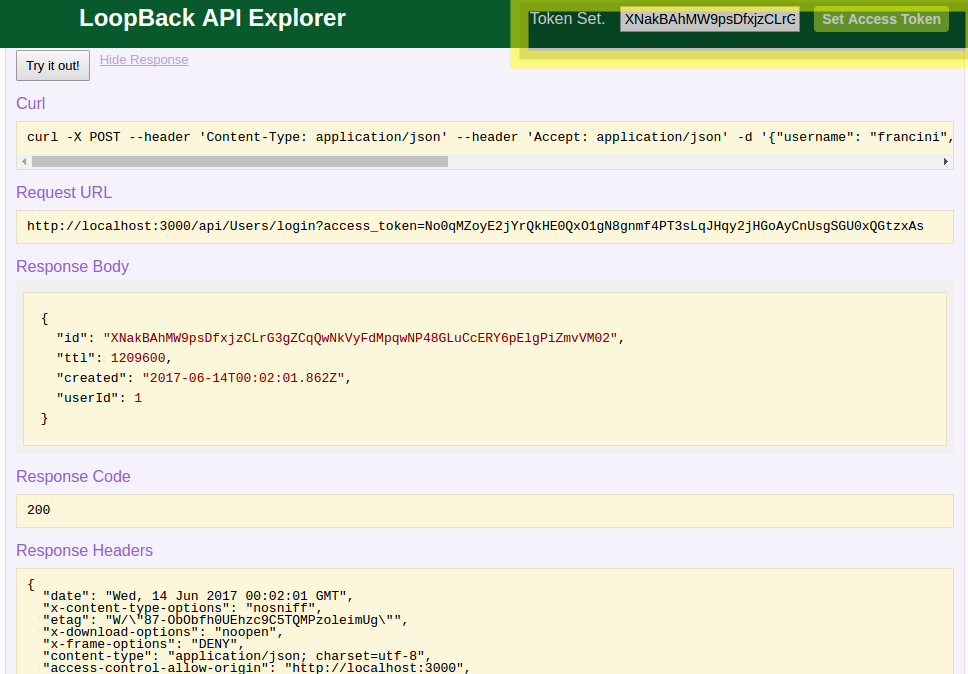
Após este passo navegue até o vero http user/login e faça login na aplicação:



Clique em Try it out e aparecerá um token no campo id para que você acesse a aplicação:



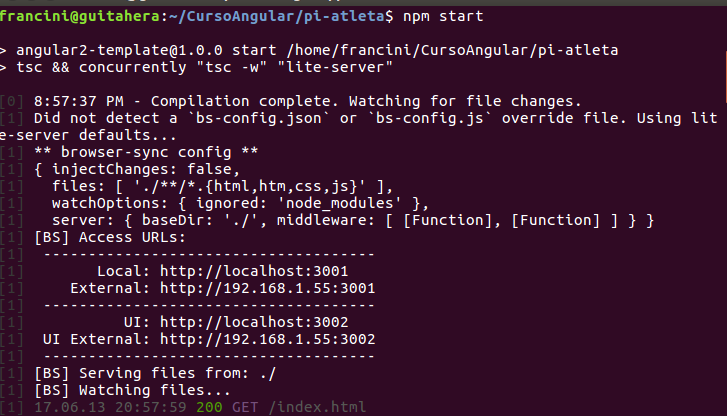
Copie este token e cole no campo toke set. e clique no botão set access token liberar o acesso a aplicação:



Depois de criar um usuário e ter acesso a api é necessário acessar o layout da aplicação. Para isto abra uma nova guia no terminal navegue até a pasta pi-atleta e insira o seguinte comando:

**npm start**

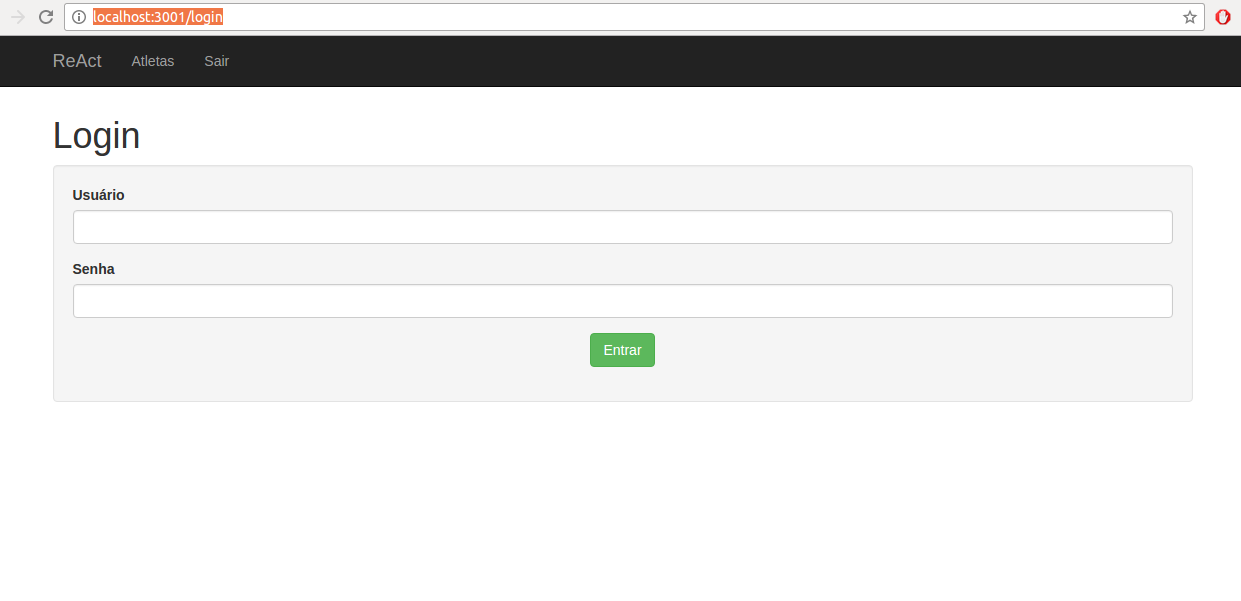
Aparecerá a seguinte resposta no terminal:



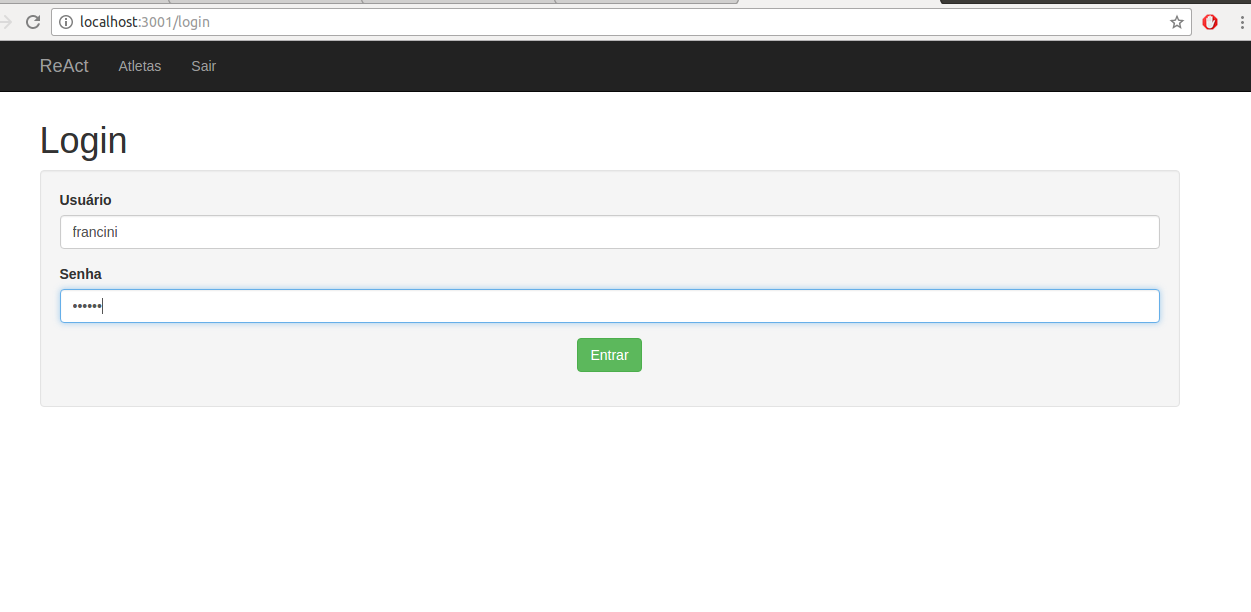
Abra uma janela do navegador e insira o seguinte caminho:

**http://localhost:3001/login**

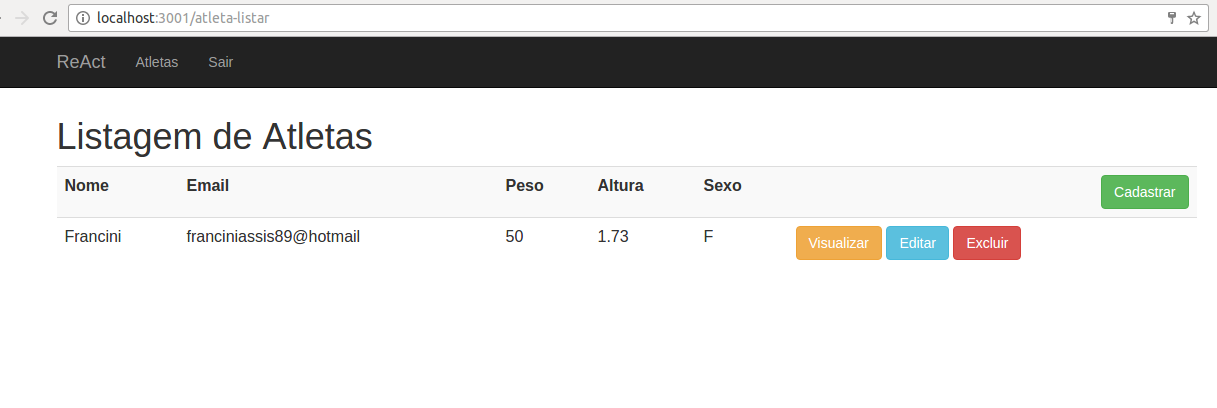
Esta será a tela visualizada:



Acesse a aplicação com as credenciais criadas anteriormente na API:

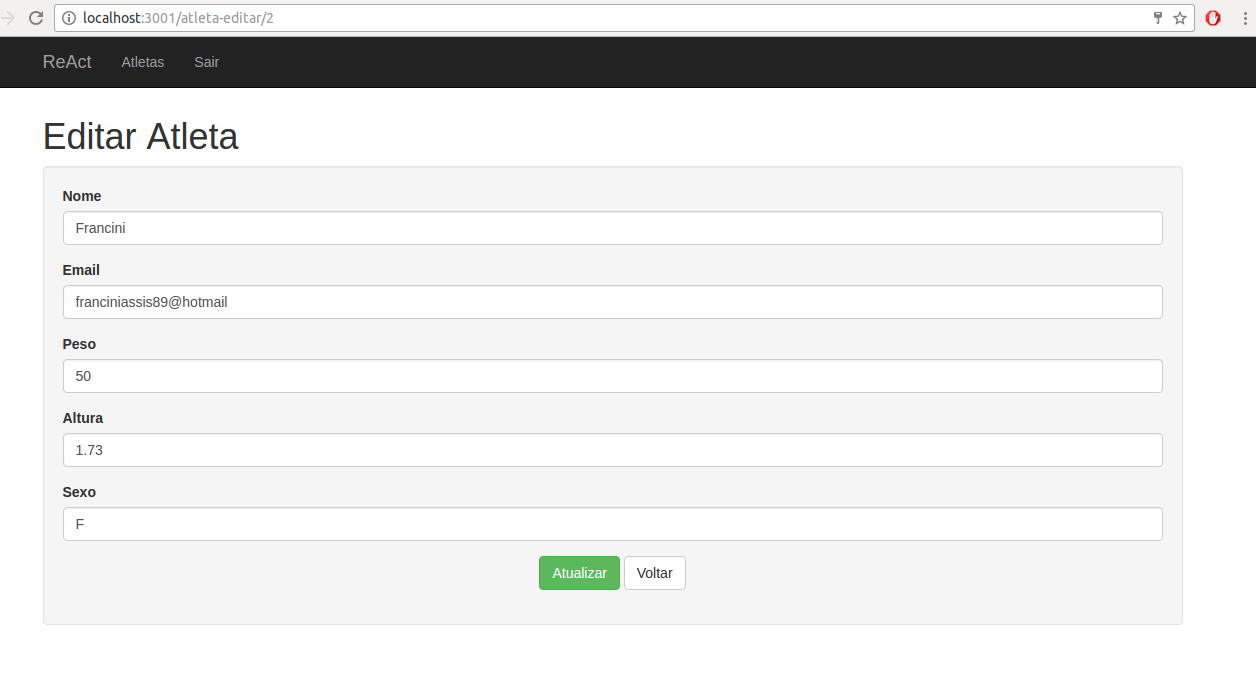


Será exibida a tela de listagem de cliente após se logar na aplicação:

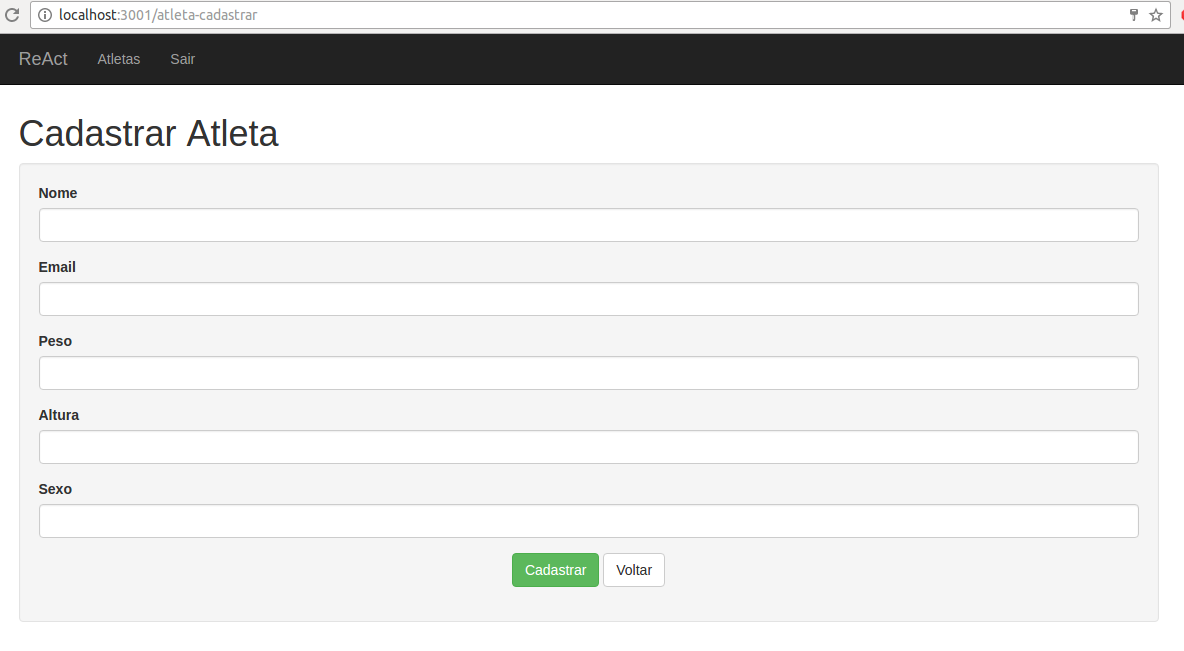


Com esta aplicação é possível criar, visualizar, editar e remover um atleta como será visto nas telas abaixo:

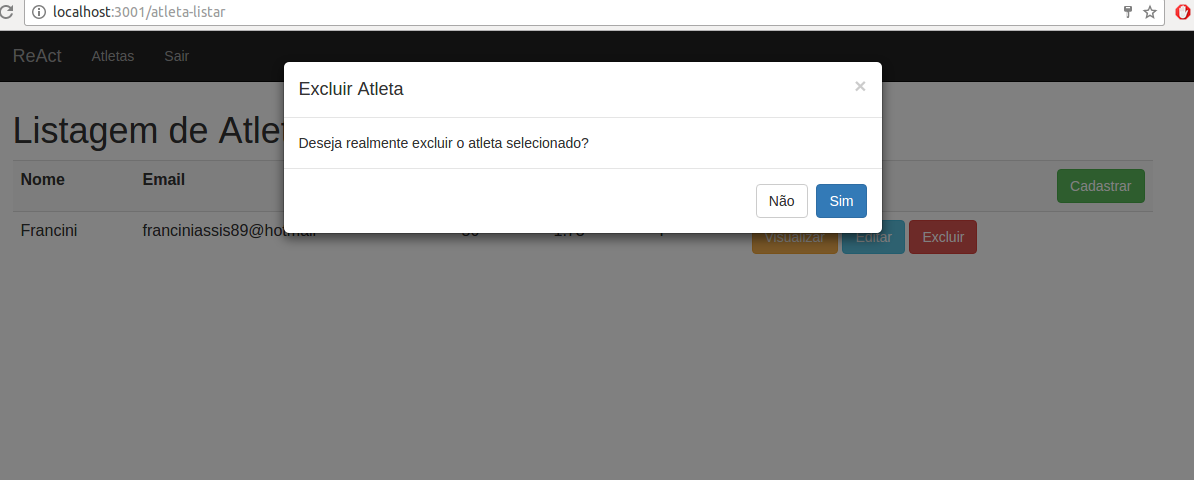
Editar:



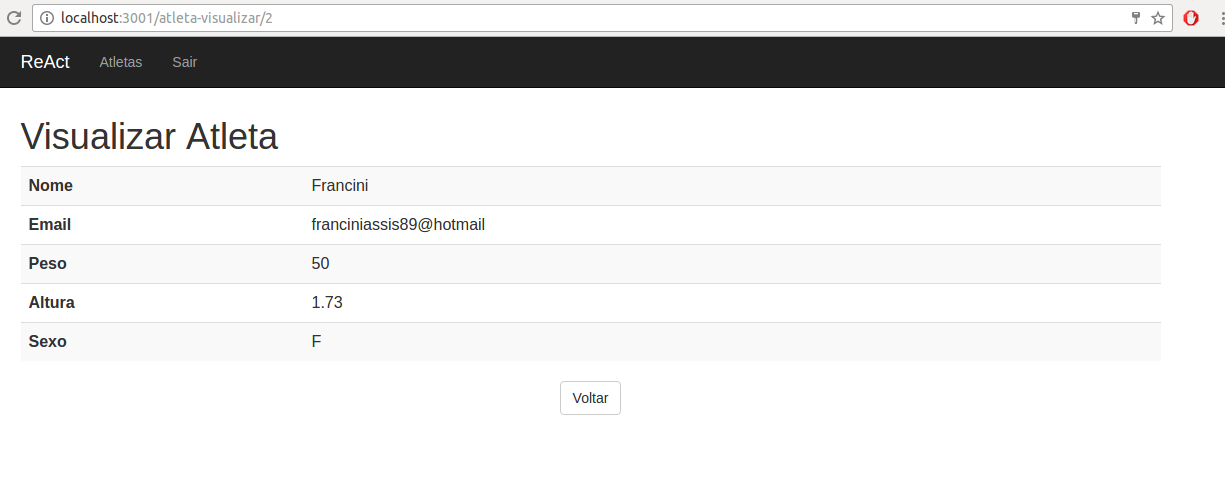
Cadastrar:



Excluir:



Visualizar:



**Hardware**

O hardware do projeto consiste em dois sistemas:

* O primeiro, a raspberry, é server do projeto. Nele ficará instalado a API e ele ficará responsável por controlar todos sensores conectados para os exercícios.



Para esse projeto, foi escolhido usar a Raspberry PI 3, pois já conta com 1gb de memória Ram, wifi e bluetooth integrados e clock de 1,2Ghz.

* O segundo, os sensores, popularmente conhecido como “rosquinhas”. Cada uma é composta de um controlador ESP-8266, um sensor de contato, leds e uma bateria.



A ESP-8266 escolhida, tem poder de hardware suficiente para fazer muito mais do que está sendo designada, ela contém uma série de portas lógicas e wifi embutido.

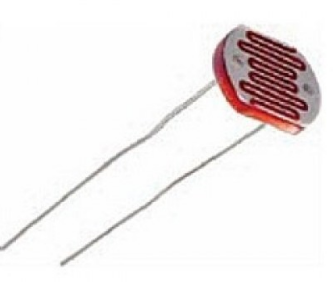
Os sensores selecionados para a função foram 3 para o começo, seriam:



Sensor ultrassônico, o sensor emitirá uma onda sonora que ao encontrar um obstáculo enviará um sinal de volta em direção ao módulo.



Sensor capacitivo, detecta sem contato e faz a medição linear de pequenos deslocamentos.

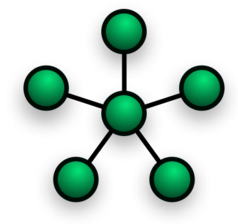


Sensor de luminosidade, a resistência dele varia de acordo com a luz e ele é muito utilizado em sensores de presença.

**A comunicação**

Ambos os sistemas se comunicam via wifi, a raspberry em modo access point, recebendo a conexão de todos os sensores. A raspberry tem o dever de informar quem deve acender e os sensores retornam o momento que eles forem apagados.

A topologia escolhida foi a do tipo estrela,



no qual a raspberry seria o meio e os sensores os terminais, assim todos os dados passariam obrigatoriamente pela raspberry e ela poderia gerenciar todas sem ter que delegar a tarefa de entrega para outro hardware.

**Primeiros problemas**

Os primeiros problemas encontrados são:

Sensor ultrassônico: Como o sensor trabalha com ondas o tempo inteiro, isso gera um desperdício de bateria.

Sensor Luminosidade: Utilizar a variação de luz para sabermos se o sensor for tocado ou não têm o problema de a luz local estar variando/fraca e o sensor não funcionar corretamente.

ESP-8266: O Esp tem uma interface de programação um pouco confusa de ser usado.

Raspberry: A raspberry utiliza um cartão de memória para armazenar o sistema operacional, o cartão de memória tem vida útil de escrita, o que nos limita ficar sobrescrevendo os dados nele. Uma opção aqui seria utilizar um HD para armazenar os dados ou uma integração com nuvem.

Gestão de Bateria: Como trabalharemos com sensores, termos que encontrar a melhor maneira de economizar energia e a melhor fonte de energia para deixarmos o sistema o menor possível.