



Universidade Federal de Ouro Preto - UFOP  
Escola de Minas  
Colegiado do curso de Engenharia de Controle e  
Automação - CECAU



Camilo Esteves Mendes de Avelar

**Domótica aplicada no controle de água utilizando comunicação MQTT, arquitetura de microsserviços e Docker: uma solução IoT.**

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, 2019

Camilo Esteves Mendes de Avelar

**Domótica aplicada no controle de água utilizando comunicação  
MQTT, arquitetura de microserviços e Docker: uma solução IoT.**

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Filipe Augusto Santos Rocha

Ouro Preto, 2019

---

Camilo Esteves Mendes de Avelar

Domótica aplicada no controle de água utilizando comunicação MQTT, arquitetura de microserviços e Docker: uma solução IoT./ Camilo Esteves Mendes de Avelar. – Ouro Preto, 2019-

63 p. : il. (algumas color.) ; 30 cm.

Orientador: Filipe Augusto Santos Rocha

Monografia de Graduação em Engenharia de Controle e Automação – Universidade Federal de Ouro Preto - UFOP, 2019.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

---

Monografia defendida e aprovada, em *XX* de *XX* de 2019, pela comissão avaliadora constituída pelos professores:

---

**Filipe Augusto Santos Rocha**  
Orientador

---

**Convidado**

---

**Convidado**

---

Ouro Preto, 2019

# Resumo

**Palavras-chaves:**

# Abstract

This is the english abstract.

**Key-words:**

# Listas de ilustrações

|           |  |    |
|-----------|--|----|
| Figura 1  | Diferença entre microprocessador de microcontrolador . . . . .     | 14 |
| Figura 2  | ESP8266 . . . . .  | 15 |
| Figura 3  | RaspberryPi . . . . .  | 15 |
| Figura 4  | Sensor de fluxo de água YF-S201 . . . . .                          | 16 |
| Figura 5  | Teclado Matricial de Membrana . . . . .                            | 17 |
| Figura 6  | Circuito do teclado . . . . .                                      | 17 |
| Figura 7  | Pinagem do teclado . . . . .                                       | 18 |
| Figura 8  | Válvula Solenoide . . . . .  | 18 |
| Figura 9  | Dashboard genérico do HomeAssistant . . . . .                      | 19 |
| Figura 10 | Exemplo arquitetura de microsserviços . . . . .                    | 22 |
| Figura 11 | Exemplo da arquitetura do MQTT . . . . .                           | 24 |
| Figura 12 | Exemplo configuração <i>headless</i> . . . . .                     | 27 |
| Figura 13 | Exemplo configuração do IP estático . . . . .                      | 27 |
| Figura 14 | Modelo Entidade Relacionamento do PostgreSQL . . . . .             | 28 |
| Figura 15 | Código de configuração do InfluxDB . . . . .                       | 30 |
| Figura 16 | Divisão dos arquivos do Sistema de Usuários . . . . .              | 31 |
| Figura 17 | Página inicial do <i>balenaEtcher</i> . . . . .                    | 33 |
| Figura 18 | Página inicial do HomeAssistant . . . . .                          | 33 |
| Figura 19 | Página inicial do Grafana . . . . .                                | 35 |
| Figura 20 | Configuração do gráfico no Grafana . . . . .                       | 36 |
| Figura 21 | Exemplo de <i>Dockerfile</i> do Sistema de Usuários . . . . .      | 36 |
| Figura 22 | Cadastro de usuários . . . . .                                     | 38 |
| Figura 23 | Edição de tempo permitido do usuário . . . . .                     | 39 |
| Figura 24 | Edição de senha do usuário . . . . .                               | 39 |
| Figura 25 | Adicionando banhos ao usuário . . . . .                            | 40 |
| Figura 26 | Recuperar dados do usuário . . . . .                               | 40 |
| Figura 27 | Recuperar dados de banhos do usuário . . . . .                     | 41 |
| Figura 28 | Exemplo de usuário não autorizado . . . . .                        | 41 |
| Figura 29 | Exemplo de usuário autorizado . . . . .                            | 42 |
| Figura 30 | Exemplo do gráfico no Grafana para usuários diferentes . . . . .   | 42 |
| Figura 31 | Exemplo do sensor no HomeAssistant quando ligado . . . . .         | 43 |
| Figura 32 | Exemplo do sensor no HomeAssistant quando desligado . . . . .      | 43 |
| Figura 33 | Exemplo de um novo sensor quando novo usuário cadastrado . . . . . | 44 |

|           |   |    |
|-----------|---|----|
| Figura 34 | Código para criar usuário . . . . .                                   | 50 |
| Figura 35 | Código para editar senha do usuário . . . . .                         | 51 |
| Figura 36 | Código para editar tempo do usuário . . . . .                         | 52 |
| Figura 37 | Código para autorizar usuário . . . . .                               | 53 |
| Figura 38 | Código para cadastrar banho do usuário . . . . .                      | 54 |
| Figura 39 | Código para recuperar usuários . . . . .                              | 55 |
| Figura 40 | Código para comunicação mqtt . . . . .                                | 56 |
| Figura 41 | Código que lida com o tempo do banho . . . . .                        | 57 |
| Figura 42 | Código que lida com as teclas apertadas no teclado numérico . . . . . | 58 |
| Figura 43 | Código que lida com a comunicação com o InfluxDB . . . . .            | 59 |

# **Lista de abreviaturas e siglas**

|         |   |
|---------|---|
| IP      | <i>Internet Protocol</i>                    |
| TSDB    | Banco de Dados de Séries Temporais          |
| MQTT    | <i>Message Queueing Telemetry Transport</i> |
| IoT     | <i>Internet of Things</i>                   |
| TCP     | <i>Transmission Control Protocol</i>        |
| SSH     | <i>Secure Shell</i>                         |
| RAM     | <i>Random Access Memory</i>                 |
| I/O     | <i>Input / Output</i>                       |
| MHz     | <i>mega hertz</i>                           |
| WiFi    | <i>Wireless Fidelity</i>                    |
| PWM     | <i>Pulse with modulation</i>                |
| mm      | milimetro                                   |
| HTTP    | <i>Hypertext Transfer Protocol</i>          |
| RESTful | <i>Representational State Transfer</i>      |
| API     | <i>Application Programming Interface</i>    |
| BLE     | <i>Bluetooth Low Energy</i>                 |
| SD      | <i>Secure Digital</i>                       |

# Sumário

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introdução . . . . .</b>                       | <b>11</b> |
| 1.1      | Objetivos . . . . .                               | 12        |
| 1.1.1    | Objetivos específicos . . . . .                   | 12        |
| 1.2      | Justificativas e Relevância . . . . .             | 13        |
| 1.3      | Organização do trabalho . . . . .                 | 13        |
| <b>2</b> | <b>Materiais e Softwares . . . . .</b>            | <b>14</b> |
| 2.1      | Microcontroladores e Microprocessadores . . . . . | 14        |
| 2.1.1    | ESP8266 . . . . .                                 | 14        |
| 2.1.2    | RaspberryPi . . . . .                             | 15        |
| 2.2      | Sensores e atuadores . . . . .                    | 16        |
| 2.2.1    | Sensor de fluxo YF-S201 . . . . .                 | 16        |
| 2.2.2    | Teclado matricial de membrana . . . . .           | 16        |
| 2.2.3    | Válvula solenoide . . . . .                       | 17        |
| 2.3      | Softwares . . . . .                               | 19        |
| 2.3.1    | HomeAssistant . . . . .                           | 19        |
| 2.3.2    | Banco de dados de séries temporais . . . . .      | 20        |
| 2.3.2.1  | InfluxDB . . . . .                                | 21        |
| 2.3.2.2  | Grafana . . . . .                                 | 21        |
| 2.3.3    | Banco de dados relacional . . . . .               | 21        |
| 2.3.3.1  | PostgreSQL . . . . .                              | 21        |
| 2.3.4    | Node.JS . . . . .                                 | 21        |
| 2.3.4.1  | Arquitetura de microsserviços . . . . .           | 22        |
| 2.3.5    | Protocolo de comunicação . . . . .                | 23        |
| 2.3.5.1  | MQTT . . . . .                                    | 23        |
| 2.3.6    | MQTT Mosquitto . . . . .                          | 24        |
| 2.3.7    | Docker . . . . .                                  | 24        |
| <b>3</b> | <b>Metodologia . . . . .</b>                      | <b>26</b> |
| 3.1      | Configuração do RaspberryPi . . . . .             | 26        |
| 3.2      | Criação das tabelas no PostgreSQL . . . . .       | 27        |
| 3.3      | Configuração do InfluxDB . . . . .                | 28        |
| 3.4      | Microsserviços . . . . .                          | 29        |
| 3.4.1    | <i>Clean Architecture</i> . . . . .               | 29        |
| 3.4.2    | Sistema de usuários . . . . .                     | 29        |
| 3.4.3    | Sistema de comunicação MQTT . . . . .             | 31        |

|          |  |           |
|----------|--|-----------|
| 3.4.4    | Sistema de simulação dos dados . . . . .                       | 32        |
| 3.5      | Configuração do HomeAssistant . . . . .                        | 32        |
| 3.5.1    | Sensores . . . . .   | 34        |
| 3.6      | Configuração do Grafana . . . . .                              | 34        |
| 3.7      | Configuração do Docker . . . . .                               | 34        |
| 3.7.1    | Docker-compose . . . . .                                       | 36        |
| <b>4</b> | <b>Experimentos e Resultados . . . . .</b>                     | <b>38</b> |
| 4.1      | Manipulação de usuários . . . . .                              | 38        |
| 4.2      | Visualização via Grafana . . . . .                             | 42        |
| 4.3      | Estado do atuador via HomeAssistant . . . . .                  | 43        |
| <b>5</b> | <b>Considerações Finais . . . . .</b>                          | <b>45</b> |
|          | <b>Referências . . . . .</b>                                   | <b>46</b> |
|          | <b>Apêndices</b>   | <b>49</b> |
|          | <b>APÊNDICE A Código sistema de usuários . . . . .</b>         | <b>50</b> |
|          | <b>APÊNDICE B Código sistema de comunicação MQTT . . . . .</b> | <b>56</b> |
|          | <b>APÊNDICE C Docker-compose.yml . . . . .</b>                 | <b>60</b> |

# 1 Introdução

Setenta por cento da superfície do planeta é coberta por água, quase toda salgada e, portanto, imprópria para o consumo humano. Apenas 2,5% desse total é potável e a maior parte das reservas (cerca de 80%) está concentrada em geleiras nas calotas polares. Essa quantidade mínima de recursos aliada ao contínuo e intenso crescimento demográfico ao longo dos anos, o desenvolvimento industrial e, por consequência, o aumento do consumo de água nas grandes cidades, tem sido um dos principais temas de discussões e palestras de conscientização por todo o mundo. (ÁGUA..., 2018)

Pesquisas mostram que, em poucas décadas, as reservas de água doce do planeta não serão suficientes para suprir as necessidades humanas caso os níveis de consumo não sejam controlados desde já (DIÁRIAS et al., 2007). A escassez deste recurso essencial à vida acarretará em problemas de ordem política, econômica, sanitária, podendo até originar conflitos similares aos causados pelo domínio do petróleo.

A economia de água é um assunto recorrente que há muito deixou de ser restrito às regiões áridas e desérticas com baixa disponibilidade de água per capita, faz com que governos e organizações de todo o mundo estejam com atenções voltadas para a criação de políticas de consumo sustentável, programas de educação ambiental, alternativas e soluções para a redução e controle do uso da água. (FERREIRA; HEROSO; ZALESKI, 2014).

A fim de evitar consequências como a escassez da água, o consumo responsável encabeça a lista de medidas a serem tomadas, por se tratar de uma atitude factível a todas as pessoas. (DIÁRIAS et al., 2007). Recentemente, avanços em recursos computacionais e tecnologias de eletrônicos permitiram a criação do paradigma do IoT (Internet of Things ou Internet das Coisas). PERUMAL; SULAIMAN; LEONG (2016) descrevem a Internet das Coisas como sendo um método para conectar coisas em torno do ambiente e realizar um certo tipo de troca de mensagem entre eles, integrando-os.

O IoT representa uma rede mundial de objetos interconectados e unicamente endereçados. É uma interconexão de dispositivos sensores e atuadores que proveem a habilidade de compartilhar informações entre plataformas através de um framework unificado, desenvolvendo uma comum capacidade de criar aplicações inovadoras. Isto é possível devido a sensores, análise de dados e representação de informações através de Computação em Nuvem como o framework unificado. (Risteska Stojkoska; TRIVODALIEV, 2017)

Uma das grandes influências do IoT é no campo do monitoramento do ambiente físico em que vivemos, sistemas de alarmes e análise de dados ambientais (PERUMAL; SULAIMAN; LEONG, 2016). Este trabalho propõe a elaboração, planejamento e implementação de um sistema de baixo custo para monitoramento e economia de água no uso de chuveiros elétricos residenciais.

Segundo Varela De Souza et al. (2016), a palavra "Domótica" resulta da junção da palavra latina "Domus", que significa casa, com "Robótica", que pode ser entendido como controle automatizado de algum processo ou equipamento, seu uso pode trazer significativas vantagens aos seus usuários como a otimização e gestão de recursos, praticidade e segurança, controle e monitoramento remoto dos dispositivos automatizados.

Compreender a evolução da tecnologia ajuda a entender como a Domótica evoluiu na forma como vivemos. O desenvolvimento de tecnologias de infraestrutura no início do século XX, como as redes de água e esgoto, gás encanado e eletricidade fizeram com que as residências se conectassem com o meio externo, tornando-se um nó de uma grande rede (FORTY, 2007). Com o advento da Internet, essa ligação se acentuou, permitindo ainda mais conectividade. (Varela De Souza et al., 2016)

## 1.1 Objetivos

Desenvolver um sistema modular, de baixo custo, baseado em microsserviços e no paradigma do IoT, para monitoramento e controle de consumo de água de chuveiros elétricos através da integração entre microprocessadores, microcontroladores, sensores e atuadores.

O sistema será capaz de armazenar e exibir dados vindos dos sensores de fluxo de água em sistemas remotos, conectados através da rede Wi-Fi, além de comandar um atuador para interromper o fornecimento da água.

### 1.1.1 Objetivos específicos

- Armazenar dados para levantamentos estatísticos;
- Implementar do sistema de identificação e controle de usuários;
- Implementar do sistema de interface;
- Implementar do sistema de atuação;
- Integrar todos os sistemas;
- Monitorar online dos parâmetros do sistema;

## 1.2 Justificativas e Relevância

Segundo Alves da Silva; Gomes de Santana (2014), o crescente consumo de água tem feito do uso consciente uma necessidade primordial. Essa prática deve ser considerada parte de uma atividade mais abrangente que é o uso racional da água, o qual inclui também, o controle de perdas e a redução do consumo de água.

Ao passar dos anos, os desperdícios da água utilizada atingem níveis nunca imaginados (REBOUÇAS, 2003). Ao se juntar o interesse e conhecimento em eletrônica e automação, será possível otimizar o monitoramento do gasto de água em residências e prédios, visando coletar dados para diminuir este tipo de desperdício. Este trabalho se justifica pela urgente necessidade de controle do uso da água em todas as esferas da sociedade.

## 1.3 Organização do trabalho

O presente trabalho está organizado em 5 Capítulos. No Capítulo 1 encontra-se a apresentação do problema e suas possíveis soluções, além de apresentar os objetivos propostos, que consistem no desenvolvimento de um sistema para domótica, baseado no IoT para monitorar o consumo de água, utilizando a arquitetura de microsserviços e microcontroladores.

O Capítulo 2 consiste na revisão sobre os hardwares e softwares utilizados no projeto, encontram-se as definições e explicações dos mesmos.

No Capítulo 3 é apresentado a estrutura geral do sistema, a metodologia em que foi construído. Explica-se também as etapas do desenvolvimento e códigos implementados.

O Capítulo 4 nos fala sobre os testes do sistema e seus resultados.

No Capítulo 5 são abordadas as considerações finais do trabalho e os possíveis trabalhos futuros.

## 2 Materiais e Softwares

### 2.1 Microcontroladores e Microprocessadores

Para uma melhor eficiência no processamento de dados, na década de 70 começaram a ser utilizados microprocessadores em computadores (MARTINS, 2005). Os microprocessadores são componentes dedicados ao processamento de informações com capacidade de cálculos matemáticos e endereçamento de memória externa (CHASE; ALMEIDA, 2007).

Já os microcontroladores são pequenos sistemas computacionais poderosos que englobam em um único chip: interfaces de entrada/saída digitais e analógicas, memória RAM, memória FLASH, interfaces de comunicação serial, conversores analógicos/digitais e temporizadores/contadores. Com o advento dos microcontroladores de 16 e 32 bits (atualmente o padrão é de 8bits), a capacidade de gerenciar soluções mais complexas e maior velocidade de processamento se iguala ao do microprocessador. (CHASE; ALMEIDA, 2007).

Na Figura 1 pode-se observar algumas diferenças entre Microprocessador e Microcontrolador.

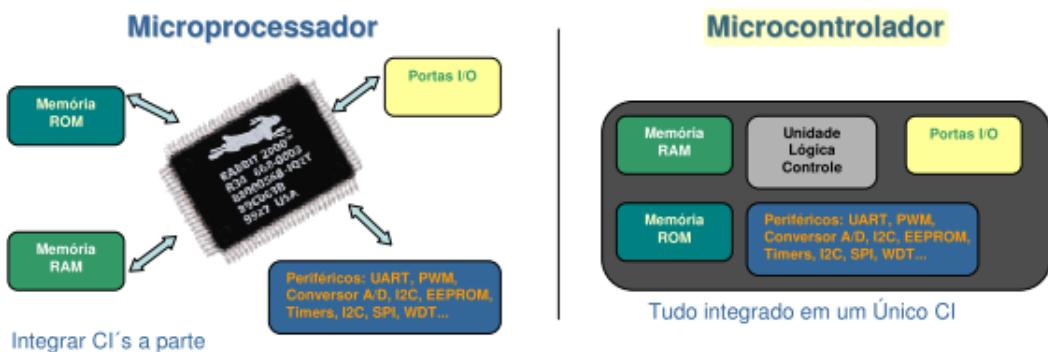


Figura 1 – Diferença entre microprocessador e microcontrolador

Fonte: CHASE; ALMEIDA (2007)

#### 2.1.1 ESP8266

O ESP8266, mostrado na Figura 2, é um circuito integrado, com interfaces de I/O digitais e analógicas e, interface Wi-Fi, com um processador de 32 bits, capaz de executar tarefas a 160 MHz.

Os módulos baseados no microcontrolador ESP8266 representam um grande avanço na relação de preço-recursos e pode ser um componente muito interessante para soluções IoT.(OLIVEIRA, 2017)

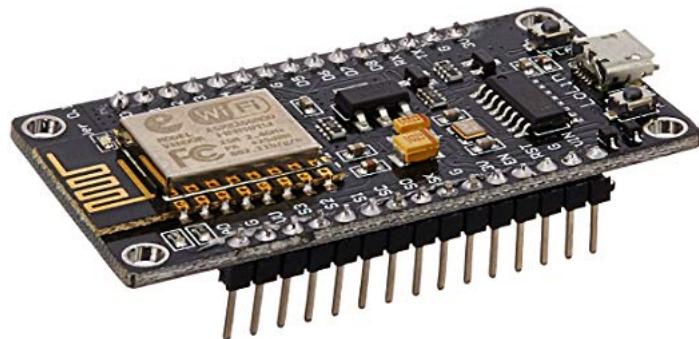


Figura 2 – ESP8266

Fonte: KODALI; SORATKAL (2017)

### 2.1.2 RaspberryPi

RaspberryPi, que pode ser visto na Figura 3, é um minicomputador criado pela Raspberry Pi Foundation com o objetivo de estimular o ensino da ciência da computação nas escolas e universidades. Apesar de o Raspberry Pi possuir o hardware em uma única placa eletrônica de tamanho reduzido, seu potencial de processamento é significativo. O Raspberry Pi pode ser usado em diversos projetos tecnológicos, como experimentos remotos nos quais sua função é ser um micro servidor web.(CROTTI et al., 2013)



Figura 3 – RaspberryPi

## 2.2 Sensores e atuadores

### 2.2.1 Sensor de fluxo YF-S201

Sensores YF-S201, mostrados na Figura 4, são sensores do tipo turbina que medem a quantidade de líquido que passa pela tubulação, girando uma turbina que gera pulsos de onda quadrada através de um sensor de efeito Hall (ROQUE; SABINO, 2018) O sensor usa esse efeito para enviar um sinal PWM e, através da contagem deste pulso é possível mensurar a quantidade de água que passa pelo cata-vento no interior do sensor, cada pulso mede aproximadamente 2,25 mm.(JÚNIOR; ARÊAS; SENA, 2017)



Figura 4 – Sensor de fluxo de água YF-S201

### 2.2.2 Teclado matricial de membrana

Teclados são geralmente utilizados em aplicações na qual o usuário precisar interagir com um sistema, como computadores, calculadoras, controles remotos entre outros.(OLIVEIRA, 2018). O Teclado Matricial de Membrana 4X4, com 16 teclas foi desenvolvido com a finalidade de facilitar a entrada de dados em projetos com plataformas microcontrolada (PICORETI, 2017), pode ser visto na Figura 5. Este teclado possui 16 teclas, onde 10 teclas são numerais, 4 literais e 2 de caracteres. As 16 teclas estão dispostas em 4 linhas por 4 colunas e o teclado possui um conector de 8 pinos para ligação.

Como o nome indica, este teclado é formado de botões organizados em linhas e colunas de modo a formar uma matriz. Quando pressionado, um botão conecta a linha com a coluna na qual está ligado. O circuito do teclado está exemplificado na Figura 6.

O teclado matricial possui a seguinte pinagem:

- Pino 1 (Esquerda na Figura 7) – Primeira Linha (L1)
- Pino 2 – Segunda Linha (L2)
- Pino 3 – Terceira Linha (L3)



Figura 5 – Teclado Matricial de Membrana

Fonte: OLIVEIRA (2018)

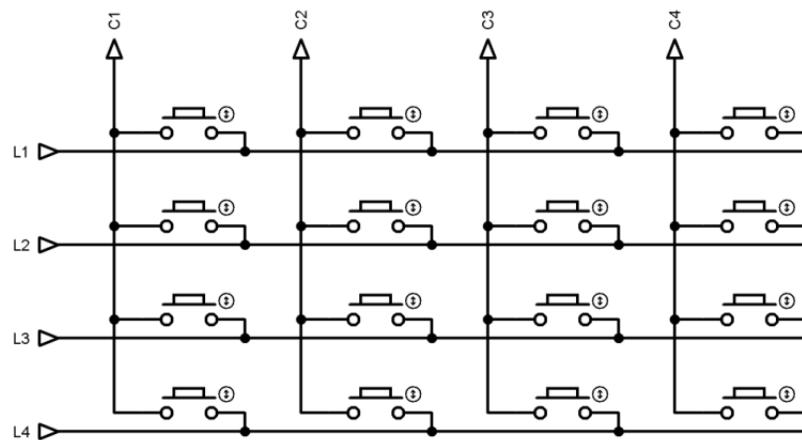


Figura 6 – Circuito do teclado

Fonte: PICORETI (2017)

- Pino 4 – Quarta Linha (L4)
- Pino 5 – Primeira Coluna (C1)
- Pino 6 – Segunda Coluna (C2)
- Pino 7 – Terceira Coluna (C3)
- Pino 8 – Quarta Coluna (C4)

Esta pinagem está mostrada na Figura 7.

### 2.2.3 Válvula solenoide

Solenóides são dispositivos eletromecânicos baseados no deslocamento causado pela ação de um campo magnético gerado por uma bobina e são muito utilizados na construção de outros dispositivos, como é o caso das válvulas para controle de fluidos. Em particular,

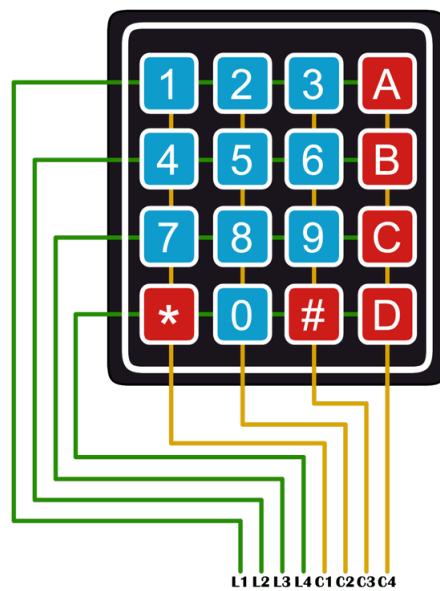


Figura 7 – Pinagem do teclado

Fonte: PICORETI (2017)

as válvulas para baixas vazões (da ordem de mililitros por minuto) e baixas pressões têm sido amplamente aplicadas em equipamentos e montagens para uso em laboratórios clínicos e químicos. Elas são de pequenas dimensões e requerem baixa tensão e corrente de acionamento. (SILVA; LAGO, 2002)

A válvula utilizada neste trabalho pode ser vista na Figura 8.



Figura 8 – Válvula Solenoide

Ainda segundo SILVA; LAGO (2002), a estratégia para fechamento e abertura dos canais fluídicos depende do fabricante, mas o princípio de acionamento elétrico é comumente o mesmo. Uma tensão de alguns volts é aplicada sobre um solenoide que faz com que um núcleo metálico ferromagnético se desloque, causando a alteração do estado da válvula. O núcleo ferromagnético comprime uma mola que é responsável por deslocar o núcleo para sua posição original quando a corrente elétrica é interrompida.

## 2.3 Softwares

### 2.3.1 HomeAssistant

HomeAssistant é um plataforma de automação escrita em Python. Iincluso componentes contribuídos por usuários que permite a interface com dispositivos e Web Services. (LUNDRIGAN et al., 2017) Em seu núcleo, HomeAssistant é um protocolo de troca de mensagens, facilitando a comunicação entre dispositivo e componentes funcionais na rede, provendo simples abstrações de componentes de automação residencial como sensores, câmeras, players de música, etc.

A Figura 9 é uma pagina inicial genérica do HomeAssistant com vários sensores configurados na parte superior, na parte esquerda, encontra-se o menu do HomeAssistant, e na parte central inferior pode-se observar informações sobre o clima, *switches* para acionamento de interruptores e iluminação.

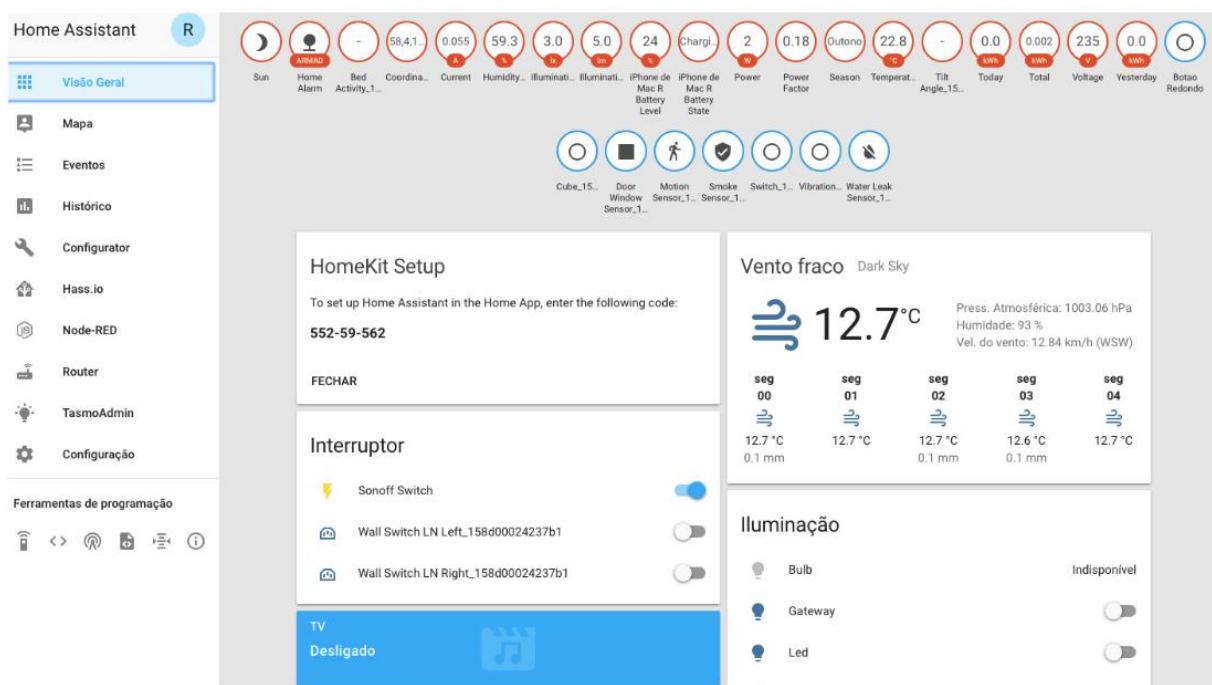


Figura 9 – Dashboard genérico do HomeAssistant

Fonte: Almeida Costa (2018)

O HomeAssistant tem suporte para diversos tipos de protocolos wireless, como BLE, ZigBee, Z-Wave e WiFi. Conta também com um RESTful API e suporta HTTP, MQTT, TCP sockets e componentes customizados. Estes componentes customizados permitem aos usuários a adicionar funções próprias ao HomeAssistant sem a necessidade de mudar o seu código fonte. Isto torna a integração de novos dispositivos e sensores muito mais fácil com o Home Assistant. (GOMES; SOUSA; VALE, 2018)

O HomeAssistant conta com uma grande comunidade de desenvolvedores com mais de 1.450 contribuidores e 23.700 estrelas no GitHub<sup>1</sup>, o que significa uma abundância em sua documentação sobre os seus componentes, fóruns e chats para conseguir ajuda de outros usuários, e diversos posts em blogs e vídeos sobre como começar a utilizar o programa.

Embora o Home Assistant tenha suporte a uma grande gama de protocolos wireless, este trabalho foca em sensores WiFi pois, ainda segundo LUNDIGAN et al. (2017):

- Hardware WiFi são mais baratos e palpáveis.
- Wi-Fi é o mais comum do que outros protocolos wireless.
- Sensores WiFi conseguem integrar com o resto da casa pois utiliza IP, tornando o sensor mais fácil de debugar e monitorar.

Almeida Costa (2018) nos diz que o HomeAssistant pode ser instalado em qualquer sistema operacional, suportado pelo Python 3 e é muito pequeno e leve, o que é compatível para o uso no Raspberry Pi como um hub de automação pequeno e barato. É importante lembrar que o Home Assistant age apenas como uma central de controlo que pode informar outros serviços, como o Philips Hue ou o Nest, para realizar alguma função. O Home Assistant é barato e de fácil configuração.

### 2.3.2 Banco de dados de séries temporais

Um Banco de Dados de Séries Temporais (TSDB) é um tipo de banco que é otimizado para dados que contém traços de tempo ou séries temporais. É construído especificamente para lidar com métricas, eventos ou medidas que variam inclusive no tempo. Um TSDB é otimizado para medidas que mudam com o tempo, permite o usuário criar, enumerar, alterar, destruir e organizar várias séries temporais de um método mais eficiente. Atualmente, a maioria das empresas estão gerando uma quantidade gigante de dados sobre métricas e eventos que levam em consideração o tempo, mostrando que a necessidade de bancos de dados de séries temporais é inevitável.(NOOR et al., 2017)

Ainda segundo NOOR et al. (2017), aplicações comuns para os TSDBs são IoT, DevOps, Análise de Dados, etc. Alguns casos de uso incluem monitoramento de sistemas de software como máquinas virtuais, monitoramento de sistemas físicos como algum equipamento, dispositivos conectados, o ambiente, sistemas de automação residencial, corpo humano, dentre outros.

<sup>1</sup> <https://github.com/home-assistant/home-assistant>

### 2.3.2.1 InfluxDB

InfluxDB é o Banco de Dados de Séries Temporais usado neste projeto. É projetado especificamente para dados do tipo séries temporais. O que combina perfeitamente com os dados coletados do sensor que será guardado no banco. (LUNDIGAN et al., 2017)

É um projeto *open-source* com o opcional de armazenamento em nuvem desenvolvido pela empresa InfluxData. É escrito na linguagem de programação Go e otimizado para lidar com dados de séries temporais. Provém de uma linguagem de consulta parecida com o SQL. (NOOR et al., 2017)

### 2.3.2.2 Grafana

Grafana é um projeto *open-source* para análise de dados de séries temporais (NOOR et al., 2017) que realiza consultas de séries temporais a partir do InfluxDB mostrando estes dados em gráficos.(CHANG et al., 2017)

### 2.3.3 Banco de dados relacional

Um Banco de dados relacional é capaz de salvar e referenciar dados para uma consulta posterior, possui uma coleção de tabelas, todas com nomes únicos, que compõem a base de dados, podendo estar relacionada a uma ou mais tabelas. Conceitos como integridade referencial de dados – que garantem que um dado referenciado em uma tabela esteja presente na tabela que está sendo referenciada – e chaves primárias estão presentes e garantem que um conjunto de informações possa ser representado de maneira consistente, independente da forma de acesso. (BOSCAROLI et al., 2006)

#### 2.3.3.1 PostgreSQL

PostgreSQL é uma excelente implementação de um banco de dados relacional, cheio de funções, de código aberto e sem custos para o uso. (STONES; MATTHEW, 2006) PostgreSQL pode ser usado com a maioria das linguagens de programação existentes.

O PostgreSQL ganhou diversos prêmios, incluindo o *Linux Journal Editor's Choice Award for Best Database* três vezes e o *2004 Linux New Media Award for Best Database System*.

### 2.3.4 Node.JS

Node.JS, também chamado de Node, é um ambiente de servidor que utiliza a linguagem de programação JavaScript. É baseado no runtime do Google, o chamado motor V8. O V8 e o Node são basicamente implementados em C e C++, focando na performance e baixo consumo de memória. Embora o V8 suporte principalmente o uso de JavaScript

no navegador, o Node foca no suporte de processos de servidores. (TILKOV; VINOSKI, 2010)

Node.JS utiliza um paradigma baseado em eventos e não-bloqueador de I/O, o que o torna leve e eficiente. É perfeito para aplicações de tempo real que lidam com dados intensos em dispositivos de baixo poder de processamento. (SAPES; SOLSONA, 2016)

O Node é um dos ambientes e *frameworks* mais famosos que suportam o desenvolvimento de servidores utilizando o JavaScript. A comunidade criou um grande ecossistema de bibliotecas e vasta documentação que dão suporte ao Node. (TILKOV; VINOSKI, 2010)

#### 2.3.4.1 Arquitetura de microsserviços

Seguindo a definição de NAMIOT; SNEPS-SNEPPE (2014), a arquitetura de microsserviços é uma abordagem no desenvolvimento de uma aplicação que baseia na existência de diversos pequenos serviços independentes. Cada um dos serviços deve rodar em seu próprio processo independente. Estes serviços podem comunicar entre si utilizando mecanismos leves de comunicação (geralmente em torno no HTTP). Os serviços devem ser absolutamente independentes.

Um exemplo da arquitetura de microsserviços está na Figura 10.

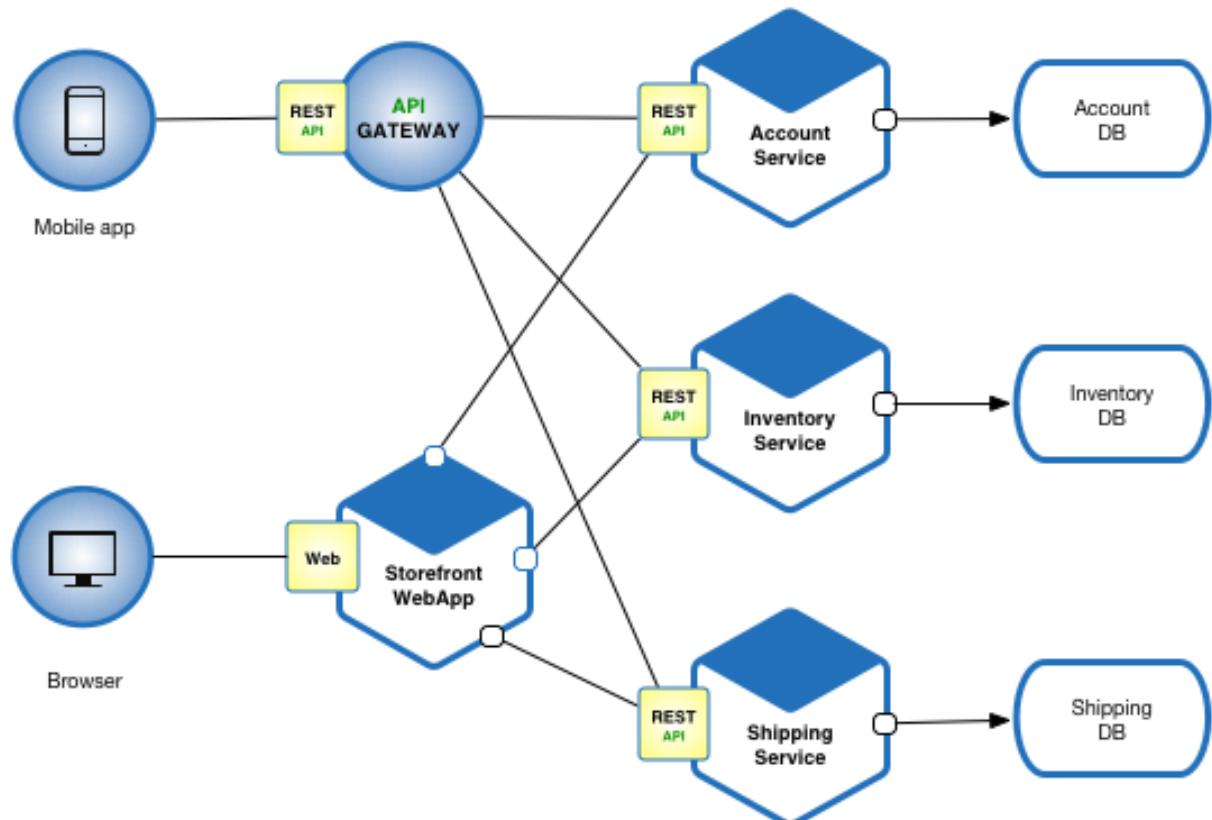


Figura 10 – Exemplo arquitetura de microsserviços

Fonte: VERTIGO (2018)

Microsserviços são os resultados da decomposição funcional de uma aplicação. São caracterizados pela definição de sua interface e função no sistema. Como cada serviço deve ser independente, uma alteração na sua implementação não deve afetar o funcionamento dos demais. (PAHL; JAMSHIDI, 2016)

### 2.3.5 Protocolo de comunicação

#### 2.3.5.1 MQTT

MQTT significa *Message Queuing Telemetry Transport*, é um protocolo de transporte leve que otimiza o uso da largura de banda de rede.<sup>2</sup> O MQTT trabalha no TCP e garante a entrega de mensagens de um nó para um servidor. Sendo um protocolo orientado por troca de mensagens, MQTT é ideal para aplicações IoT, que tem recursos e capacidades limitados.

É um protocolo inicialmente desenvolvido pela IBM<sup>3</sup> em 1999, e recentemente foi reconhecido como padrão pela OASIS (Organization for the Advancement of Structured Information Standards).<sup>4</sup>

KODALI; SORATKAL (2017) definiu o MQTT como um protocolo baseado em *publish/subscribe*. Qualquer conexão MQTT envolve dois tipos de agentes, os clientes MQTT e um MQTT *broker* público, ou servidor MQTT. Os dados que são transportados pelo MQTT são referenciados como mensagens da aplicação. Qualquer dispositivo ou programa que é conectado pela rede e troca mensagens através do MQTT é chamado de cliente MQTT. Um cliente MQTT pode ser tanto um *publisher* ou um *subscriber*. Um *publisher* publica mensagens e um *subscriber* requisita o recebimento de mensagens. Um MQTT *server* é um programa que interconecta os clientes MQTT. Ele aceita e transmite as mensagens através de múltiplos clientes conectados à ele.

Dispositivos como sensores, celulares, são considerados como clientes MQTT, quando um cliente MQTT tem alguma informação para transmitir, ele publica o dado para o *broker* MQTT.

O *broker* MQTT (Figura 11), ou servidor MQTT, responsável por coletar e organizar os dados. As mensagens publicadas por clientes MQTT são transmitidas para outros clientes MQTT que se inscreverem ao tópico. O MQTT é desenhado para simplificar a implementação no cliente por concentrar todas as complexidades no *broker*. Os *publishers* e *subscribers* são isolados, o que significa que eles não precisam conhecer a existência do outro.

<sup>2</sup> <http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>

<sup>3</sup> <http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt>

<sup>4</sup> <https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard>

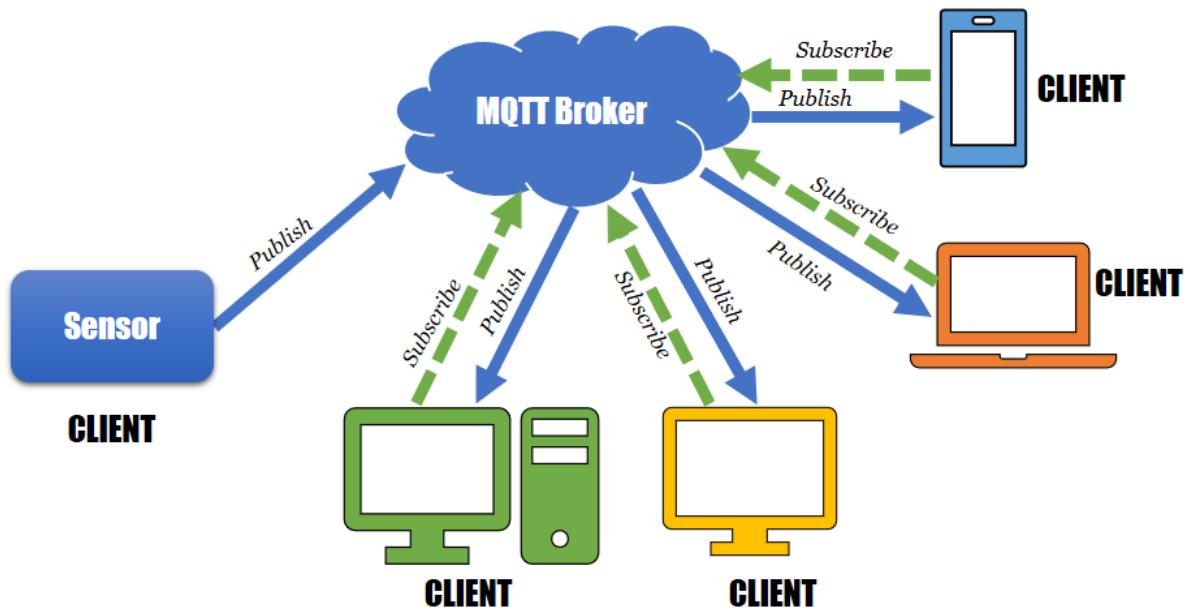


Figura 11 – Exemplo da arquitetura do MQTT

Fonte: LABORATORY (2018)

### 2.3.6 MQTT Mosquitto

O Mosquitto é um *broker* MQTT de código aberto (KODALI; SORATKAL, 2017) que entrega uma implementação padrão de servidor e cliente MQTT. Utiliza o modelo *publish/subscribe*, tem uma baixa utilização de rede e pode ser implementado em dispositivos de baixo custo como microcontroladores que podem ser usados em sensores remotos de IoT. (LIGHT, 2017)

Segundo LIGHT (2017), Mosquitto é recomendado para o uso sempre em que se necessita de mensagens leves, particularmente em dispositivos com recursos limitados.

O Projeto Mosquitto é um membro da Eclipse Foundation. Existem três partes no projeto:

- O servidor principal Mosquitto.
- Os clientes mosquitto *pub* e mosquitto *sub*, que contém ferramentas para se comunicar com o servidor MQTT.
- Uma biblioteca cliente MQTT, escrita em C.

### 2.3.7 Docker

Docker é um projeto *open source* que foi inicialmente lançado em 2013, atraiu grande atenção na industria de TI. É uma plataforma de conteinerização que possibilita

usuários a construir sua aplicação dentro de um conteiner e transferir conteiners através de máquinas com diferentes sistemas operacionais de um jeito simples. (CHANG et al., 2017)

Existem três componentes principais do Docker:

- *Docker images*: são *templates* de leitura que servem como base para a criação de conteiners.;
- *Docker registries*: é o local onde estão uma grande coleção de *Docker images*.;
- *Docker containers*: são as instâncias virtuais em que as aplicações estão rodando. Cada conteiner contém uma aplicação rodando e todas os seus arquivos de dependências, como o código, bibliotecas e utilitários do sistema.

A construção de imagens pode ser feita de duas maneiras. É possível criar um conteiner através de uma imagem já existente (*docker run*), realizar modificações e instalações dentro do conteiner, parar o container e depois salvar o estado atual do conteiner como uma nova imagem (*docker commit*). Este processo é parecido com uma instalação clássica de uma máquina virtual, mas deve ser feito para cada imagem caso haja alguma atualização, já que as imagens são padronizadas. Para automatizar o processo, *Dockerfiles* nos permite especificar uma imagem de base e uma sequência de comandos que serão executados quando a imagem é construída, juntamente com outras opções de especificações, como portas a serem expostas. A imagem é depois construída com o comando *docker build*.(PIETRO ANTHONY MARTIN, 2016)

# 3 Metodologia

Este Capítulo apresenta a estrutura geral do sistema, a configuração inicial do RaspberryPi, a construção dos microsserviços, assim como as etapas de realização do projeto. A primeira etapa consiste no desenvolvimento do microsserviço de cadastro e controle de usuários. Em seguida, foi desenvolvido o sistema que recebe os dados via MQTT, guardando-os no InfluxDB. Com os dados sendo recebidos e devidamente guardados, foi configurado o HomeAssistant juntamente com o Grafana. Para facilitar os testes, também foi desenvolvido um sistema que simula os dados que são recebidos via sensor de fluxo.

## 3.1 Configuração do RaspberryPi

Para a configuração inicial do RaspberryPi é preciso fazer o *download* de um sistema operacional compatível com o microcontrolador. Para o sistema deste projeto utilizamos o Hassbian<sup>1</sup>, que foi instalado no cartão SD que será inserido no RaspberryPi.

Os sistemas deste projeto foram desenvolvidos diretamente no RaspberryPi, e, para possibilitar este desenvolvimento, é necessário primeiramente configurar o ambiente do RaspberryPi.

O RaspberryPi foi configurado para ser *headless*<sup>2</sup>, o que faz com que ele não necessite de teclado, mouse ou monitor para poder ser acessado. Para conseguir acessar o RaspberryPi nesta configuração é necessária a utilização do SSH, ou *Secure Shell*, que é um protocolo de rede criptográfico para operação de serviços de rede de forma segura<sup>3</sup>. O SSH permite o login remoto no sistema operacional do Raspberry, possibilitando o completo controle do sistema operacional de forma remota.

A configuração de rede do RaspberryPi foi realizada para conter um IP estático, para facilitar o acesso SSH ao sistema, pois, para realizar o acesso é necessário saber o IP do RaspberryPi na rede, no sistema desenvolvido o IP do RaspberryPi foi configurado para ser sempre *192.168.2.60*.

A Figura 12 mostra um exemplo do arquivo *wpa\_supplicant.conf*, que faz parte da configuração *headless*, serve para conectar à rede automaticamente ao iniciar os RaspberryPi. Já na Figura 13, podemos observar a configuração do ip estático.

---

<sup>1</sup> <https://www.home-assistant.io/docs/installation/hassbian/installation/>

<sup>2</sup> <https://www.raspberrypi.org/documentation/configuration/wireless/headless.md>

<sup>3</sup> <https://www.ssh.com/ssh/>

```
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
    ssid=<Name of your WiFi>
    psk=<Password for your WiFi>
}
```

Figura 12 – Exemplo configuração *headless*

```
#static IP configuration

interface enxb827ebdc0d1f
static ip_address=192.168.1.15/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1
```

Figura 13 – Exemplo configuração do IP estático

### 3.2 Criação das tabelas no PostgreSQL

Em bancos de dados relacionais, para salvar os dados é necessário criar um banco de dados com o nome desejado, dentro do banco, são criadas tabelas que se relacionam entre si, e, dentro das tabelas, existem colunas, referente ao dado que deseja guardar. Nesta seção estão as descrições do banco de dados criado, suas tabelas e colunas.

Para o sistema desenvolvido foi criado um banco de dados com o nome *users* com três tabelas: *user\_baths*, *user\_settings* e *users*.

A tabela *users* contém 5 colunas, que guardam informações dos usuários cadastrados, que são:

- *id*: gera um id que se auto incrementa e é único para cada usuário
- *name*: referente ao nome do usuário
- *password*: informação da senha do usuário, que é criptografada
- *created\_at*: data e hora da criação do usuário
- *updated\_at*: data e hora da ultima atualização do usuário

A tabela *user\_settings*, guarda informações de configuração dos usuários, e contém 5 colunas:

- *id*: identificador único

- *user\_id*: referente ao id do usuário
- *allowedBathTime*: tempo de banho permitido do usuário
- *created\_at*: data e hora da criação do usuário
- *updated\_at*: data e hora da ultima atualização do usuário

A tabela *userBaths* guarda informações de tempo de todos os banhos tomados, possui 5 colunas:

- *id*: identificador único
- *user\_id*: referente ao id do usuário
- *time*: tempo total do banho, em milissegundos
- *created\_at*: data e hora da criação do usuário
- *updated\_at*: data e hora da ultima atualização do usuário

A Figura 14 nos mostra o diagrama Entidade Relacionamento do banco de dados *users* criado para a aplicação.

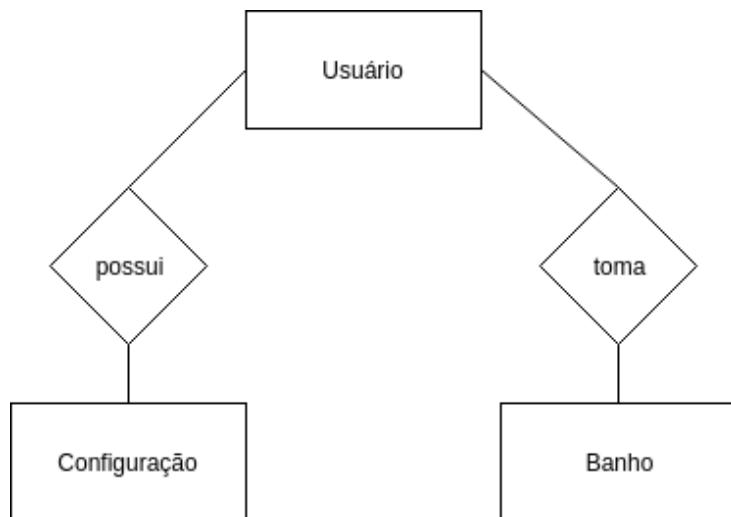


Figura 14 – Modelo Entidade Relacionamento do PostgreSQL

### 3.3 Configuração do InfluxDB

Na configuração do InfluxDB, foi utilizada a biblioteca *influx*<sup>4</sup>. Esta biblioteca permite a criação do banco de dados diretamente do código fonte do sistema, assim como as tabelas para guardar os dados temporais.

<sup>4</sup> <https://www.npmjs.com/package/influx>

Foi criado o banco de dados *water\_flow\_data*, que guardará os dados temporais dos banhos dos usuários. Neste banco os dados são guardados com a *tag* referente ao id do usuário cadastrado no banco relacional PostgreSQL, os *fields* são guardados os dados que chegam diretamente do sensor, que significa o fluxo de água naquele determinado instante.

O InfluxDB salva automaticamente os *timestamps* de cada dado incluído nele, o que facilita a manipulação dos dados, pois não é necessário informar o *timestamp* toda vez que for incluir algum dado no banco.

A Figura 15 está a parte do código de configuração e implementação das funções do InfluxDB no sistema.

## 3.4 Microsserviços

Nesta seção será apresentado as etapas de desenvolvimento dos microsserviços utilizados no sistema elaborado, assim como o serviço de simulação para facilitar os testes e validação do sistema.

### 3.4.1 Clean Architecture

Os microsserviços desenvolvidos foram projetados para seguir a *Clean Architecture*. Esta arquitetura consiste em dividir as responsabilidades dentro de uma aplicação, encapsulando e abstraindo o código para facilitar a leitura e entendimento das devidas funções de cada arquivo.

É possível ver a divisão dos arquivos do Sistema de Usuários na Figura 16.

### 3.4.2 Sistema de usuários

O propósito desta aplicação está em ter o controle e informações sobre o consumo de água em residências, para isso, é necessário ter um meio para conseguir as informações sobre os usuários do sistema.

O microsserviço de usuários é responsável por cadastrar e editar usuários, cadastrar informações sobre o tempo de banho do usuário e autorizar o usuário à tomar o banho.

As operações são realizadas através dos *endpoints*, que são *links* em que são passados parâmetros contendo a informação específica que o *endpoint* requer.

Este microsserviço salva todos os dados recebidos nas tabelas do PostgreSQL.

O sistema possui os seguintes *endpoints*:

- /cadastrar: possibilita cadastrar o usuário com as informações do nome, senha e tempo de banho permitido.

```
class InfluxHandler {
  constructor() {
    this.influx;
    this.host = 'influxdb';
    this.database = 'water_flow_data'
  }

  connect() {
    this.influx = new Influx.InfluxDB( options: {
      host: this.host,
      database: this.database,
      schema: [{
        measurement: 'flow_data',
        fields: { flow: Influx.FieldType.INTEGER },
        tags: ['user_id']
      }]
    });

    this.influx.getDatabaseNames()
      .then( onfulfilled: names => {
        if(!names.includes( searchElement: this.database)) {
          return this.influx.createDatabase( databaseName: this.database);
        }
      })
      .then( onfulfilled: () => {
        console.log( message: 'InfluxDB connected!');
      })
  }

  getPoints(id) {
    console.log( message: id)
    return this.influx.query(
      ` select * from flow_data
      | where user_id = '${id}'` )
      .catch( onrejected: err => {
        console.log( message: err);
      })
      .then( onfulfilled: results => {
        return results;
      })
  }

  writePoints(id, data) {
    console.log( message: id, optionalParams[0]: data);
    return this.influx.writePoints( points: [
      {
        measurement: 'flow_data',
        tags: { user_id: id },
        fields: { flow: data }
      }
    ], options: {
      database: this.database,           CamiloAvelar, a month ago · first commit
      precision: 's',
    })
    .catch( onrejected: err => console.error( message: `Error saving data to InfluxDB! ${err.stack}`))
  }
}
```

Figura 15 – Código de configuração do InfluxDB

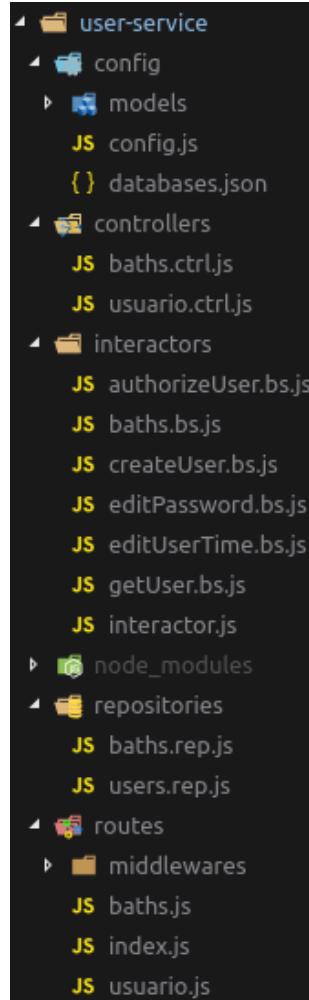


Figura 16 – Divisão dos arquivos do Sistema de Usuários

- /autorizar: recebe o id do usuário e a senha, compara a senha enviada com a senha cadastrada e retorna se o usuário está ou não autorizado.
- /editar-tempo: possibilita editar o tempo de banho permitido do usuário.
- /editar-senha: possibilita editar a senha do usuário.
- /banho: salva no banco de dados informações do banho, o tempo e o usuário que tomou o banho.
- /banho/:id: retorna informações sobre todos os banhos do usuário.

Os códigos de implementação deste sistema podem ser observados no Apêndice A.

### 3.4.3 Sistema de comunicação MQTT

Este microsserviço é responsável pelo recebimento e manipulação dos dados recebidos dos sensores e do teclado numérico via MQTT. Também é responsável por comunicar com

o sistema de usuários para autorizar o usuário, por ligar ou desligar o atuador, além de salvar os dados no InfluxDB.

O sistema de comunicação recebe informações sobre as teclas digitadas e comunica com o sistema de usuários, para autorizar ou não o ligamento do atuador, que significa o início do banho.

O sistema se subscreve nos seguintes tópicos do *broker* MQTT para receber as informações:

- *keys*: é o tópico em que contém a tecla digitada no teclado numérico
- *actuator*: é o tópico para enviar informações do atuador, para liga-lo ou desliga-lo
- *user*: é o tópico para enviar informações do usuário, como o nome e se ele está autorizado ou não.
- *sensor*: é o tópico para enviar informações do sensor de fluxo.

Os códigos de implementação deste sistema podem ser observados no Apêndice B.

#### 3.4.4 Sistema de simulação dos dados

### 3.5 Configuração do HomeAssistant

O HomeAssistant pode ser instalado a partir de diversos métodos (colocar link no rodapé), neste sistema, a instalação foi feita utilizando o Hassbian, que é um sistema operacional linux para o RaspberryPi com o HomeAssistant pré-instalado.

Para configurar o HomeAssistant no RaspberryPi é necessário realizar o *download* da imagem do Hassbian, com o *download* concluído, o próximo passo é copiar esta imagem para o cartão SD que será inserido no RaspberryPi, para este passo, utilizamos o *balenaEtcher*, observado na Figura 17.

Com a imagem devidamente escrita no cartão SD, basta inseri-lo no RaspberryPi, configurar a rede como descrito no processo de configuração do RaspberryPi, e, ao iniciar o sistema, o HomeAssistant será carregado automaticamente.

O HomeAssistant utiliza a porta 8123, e, para conseguir visualizar sua interface, bastar acessar o ip do RaspberryPi na porta 8123. No caso deste sistema, o IP do RaspberryPi foi colocado como estático *192.168.2.60*, então, para acessar a interface basta estar conectado na mesma rede do RaspberryPi e acessar o link *http://192.168.2.60:8123*, como podemos ver na Figura 18.

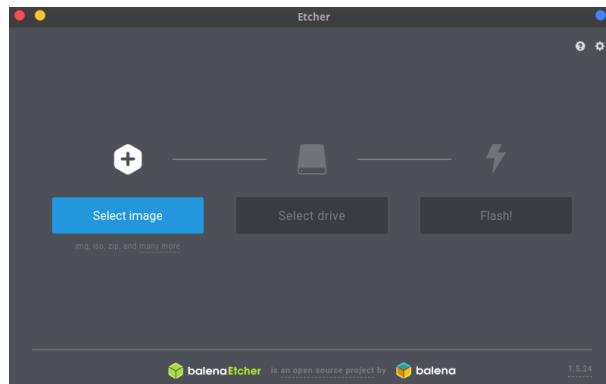
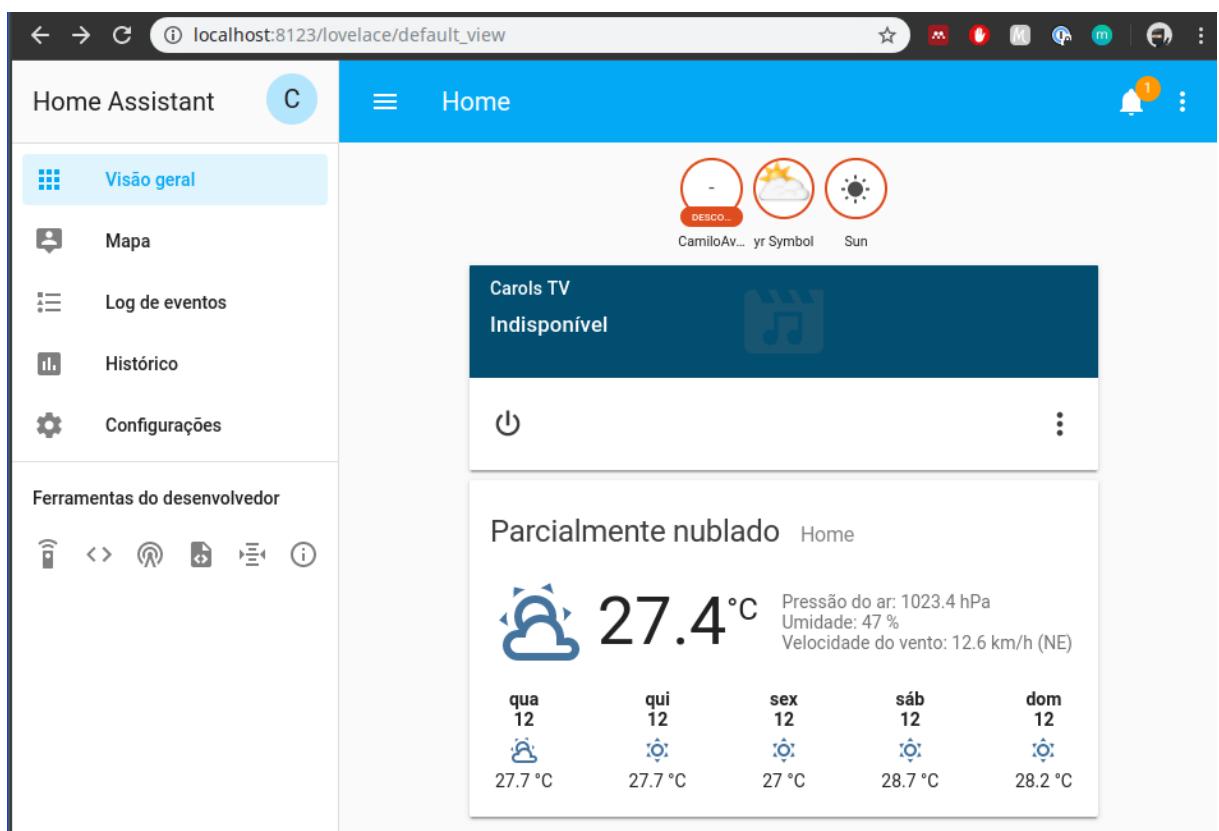
Figura 17 – Página inicial do *balenaEtcher*

Figura 18 – Página inicial do HomeAssistant

### 3.5.1 Sensores

A interface do HomeAssistant nos permite diversas configurações e inclusão de diferentes sensores, como mostra a Figura 9. No sistema desenvolvido neste trabalho, foi utilizado o sensor binário, que nos mostrará se o chuveiro está ligado ou desligado. Cada usuário terá um sensor próprio, que mostrará *ON* caso o usuário esteja com o chuveiro ligado, ou *OFF* caso o usuário não esteja tomando banho, como na imagem (COLOCAR IMAGEM).

## 3.6 Configuração do Grafana

A instalação do Grafana foi realizada através do *download* e extração do pacote para o sistema operacional Debian, que é o sistema utilizado pelo Hassbian instalado no RaspberryPi.

Com o pacote devidamente instalado no sistema, o Grafana utiliza a porta 3003 como interface, então, para acessar a sua interface, basta acessar a porta 3003 do RaspberryPi, ou seja, acessar o link <http://192.168.2.60:3003>, como na Figura 19.

Ao acessar a interface do Grafana, é necessário criar um *dashboard* para a visualização dos dados do InfluxDB. Neste sistema criamos o *dashboard AllBaths*, que se conecta no ip e porta do InfluxDB e realiza as consultas no banco *water\_flow\_data* criado para guardar os dados temporais do fluxo de água.

A configuração do gráfico no Grafana pode ser observada na Figura 20.

Após a configuração do *dashboard*, um atalho será criado na página inicial do Grafana, e para visualizar os dados, basta clicar neste atalho e será aberto um gráfico com os fluxos medidos no sensor.

## 3.7 Configuração do Docker

Toda a configuração do Docker é realizada a partir do arquivo *Dockerfile*, que deve ser criado no diretório do projeto em que se deseja configurar.

No *Dockerfile* alguns parâmetros devem ser passados para conseguir o resultado esperado, um deles é definir qual imagem do Docker será usada como base, no caso dos projetos desenvolvidos, todos utilizam o Node como imagem base, esta imagem é definida com o comando *FROM node*.

Outro parâmetro que precisa ser configurado é o *WORKDIR*, que define o diretório que será usado como base dentro da imagem do Docker, no caso, dentro da imagem do Node.

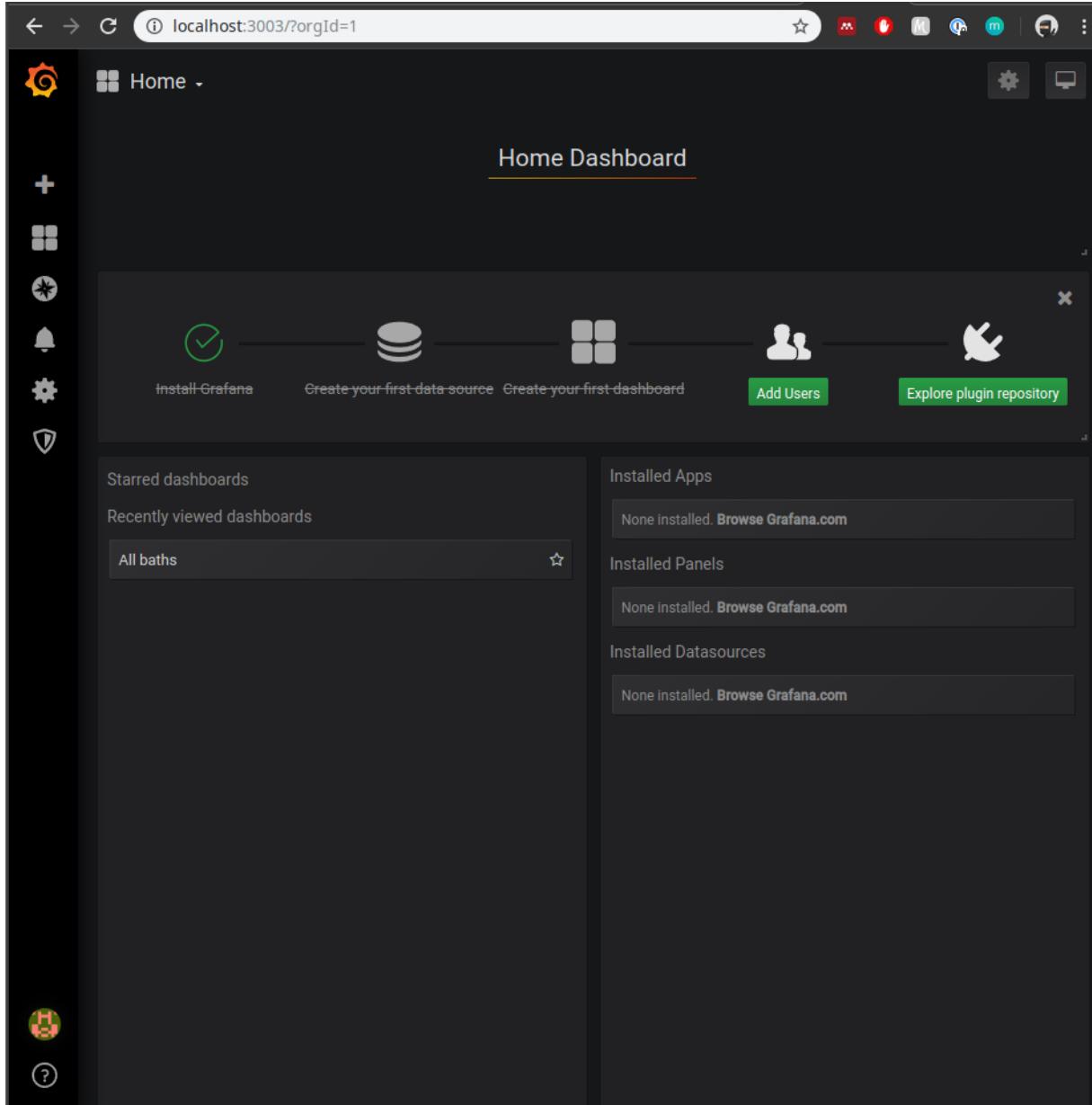


Figura 19 – Página inicial do Grafana

Um comando importante é o *COPY*, que serve para copiar os arquivos do diretório raiz do Docker, para o *WORKDIR* dentro da imagem do Node.

Após copiar os arquivos, é necessário executar os comandos para instalar os pacotes dos projetos, com o comando *RUN npm install*.

Finalmente, deve-se definir um comando de entrada, que irá executar o projeto dentro da imagem no Docker, que é o comando *CMD npm start*. Caso o projeto utilize alguma porta para configuração, deve-se também expor a porta para garantir o funcionamento com o comando *EXPOSE numeroDaPorta*.

O exemplo do *Dockerfile* do Sistema de Usuários pode ser verificado na Figura 21.

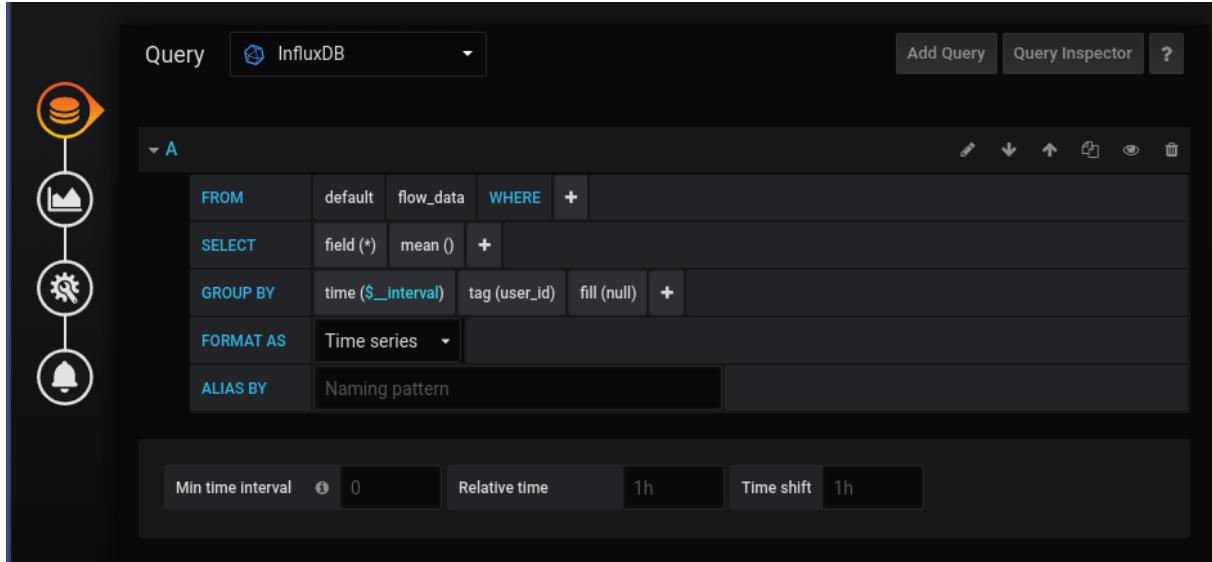


Figura 20 – Configuração do gráfico no Grafana

```
FROM node:10
WORKDIR /app
COPY package.json /app
RUN npm install -g nodemon
RUN npm install
RUN npm install -g gulp
RUN npm install -g sequelize
RUN npm install -g sequelize-cli
RUN npm install -g pg
COPY . /app
RUN git clone https://github.com/vishnubob/wait-for-it.git
CMD npm start
EXPOSE 3001
```

Figura 21 – Exemplo de *Dockerfile* do Sistema de Usuários

### 3.7.1 Docker-compose

O *Dockerfile* permite a execução única do projeto em que ele está presente, para iniciar os projetos utilizando apenas o *Dockerfile* é necessário iniciar um por um.

Para contornar este problema, o *docker-compose* permite a execução de todos os projetos com apenas um comando, desde que um arquivo *docker-compose.yml* esteja devidamente configurado. Este arquivo permite também a instalação e configuração automática de todos os programas utilizados neste trabalho, como o PostgreSQL, o InfluxDB, o Grafana e o Mosquitto.

A configuração do *docker-compose.yml* é simples como a do *Dockerfile*, nele, deve-se especificar a versão que deseja utilizar, neste projeto foi utilizada a 3.5, com o comando *version:"3.5"*.

Após a definição da versão, deve-se especificar os *services*, que são as imagens que devem iniciar com o Docker. Neste projeto foram utilizados os *builds* dos projetos já implementados com os devidos *Dockerfiles* configurados.

Além dos projetos já implementados, foram especificados também os programas que foram utilizados no sistema, com o comando *image:postgres*, *image:homeassistant*, *image:eclipse-mosquitto*, *image:grafana*, *image:influxdb*.

O código implementado do *docker-compose.yml* do projeto pode ser verificado no Apêndice C.

# 4 Experimentos e Resultados

Neste Capítulo encontram-se os experimentos realizados com os sistemas devidamente implementados e configurados. Discutiremos os resultados obtidos, testando os sistemas com todas as suas funcionalidades, como criar e editar usuários, mudar a senha, autenticar, adicionar banhos, visualizar o gráfico dos dados do sensor em tempo real via Grafana e o estado do atuador via HomeAssistant.

## 4.1 Manipulação de usuários

O projeto de usuários utiliza a porta 3001, para utilizá-lo e testá-lo, devemos utilizar esta porta juntamente com os *endpoints* disponíveis. Aqui realizaremos as consultas em todos estes *endpoints* e discutiremos os resultados obtidos.

Para criar um usuário devemos consumir o *endpoint* `http://localhost:3001/usuario/cadastrar` passando os parâmetros via *POST* no formato *JSON*, como na Figura 22.

A resposta deste *endpoint* pode ser conferida na Figura 22, e, para garantir que o usuário foi incluído no banco de dados, podemos observar a Figura 22.

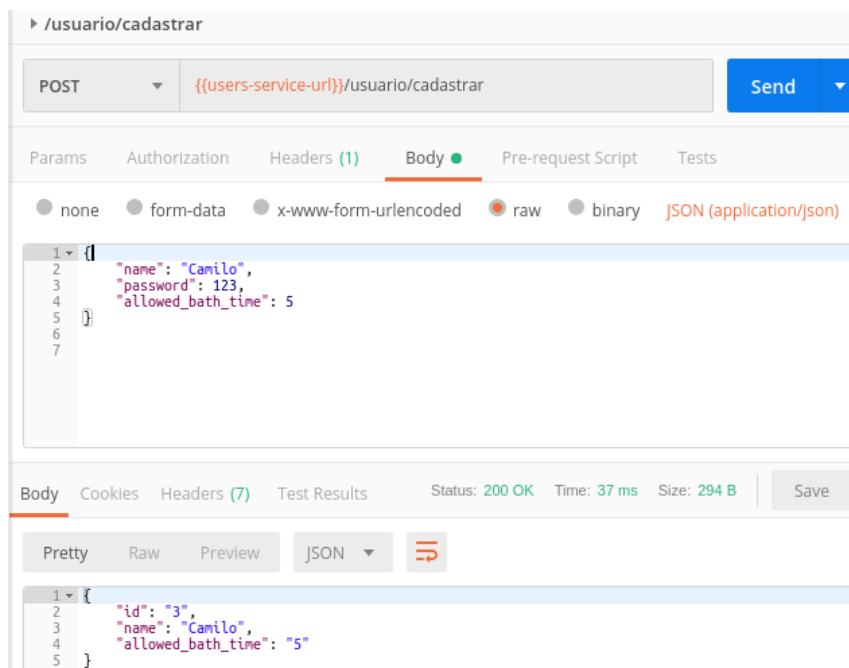


Figura 22 – Cadastro de usuários

Para editar um usuário, o *endpoint* `http://localhost:3001/usuario/editar-tempo`,

para editar tempo, e o `http://localhost:3001/usuario/editar-senha` para editar a senha, deve ser consumido passando os parâmetros via *POST* como na Figura 23, para editar o tempo, e a imagem, para editar a senha.

A resposta pode ser observada na Figura 23, para o tempo, e Figura 24 para a senha.

The screenshot shows a Postman interface for a POST request to `http://{{users-service-url}}/usuario/editar-tempo`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "id": 3,
3   "new_time": 2
4 }

```

The response tab shows a successful 200 OK status with the message:

```

1 {
2   "status": 200,
3   "message": "Usuário editado com sucesso"
4 }

```

Figura 23 – Edição de tempo permitido do usuário

The screenshot shows a Postman interface for a POST request to `http://{{users-service-url}}/usuario/editar-senha`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "id": 3,
3   "new_password": 1234
4 }

```

The response tab shows a successful 200 OK status with the message:

```

1 {
2   "status": 200,
3   "message": "Usuário editado com sucesso"
4 }

```

Figura 24 – Edição de senha do usuário

Conseguimos cadastrar um banho ao consumir o *endpoint* `http://localhost:3001/banho` via *POST* com os parâmetros e resposta mostrados na Figura 25.

The screenshot shows the Postman interface for a POST request to `http://{{users-service-url}}/banho`. The Body tab is selected, showing a JSON payload:

```

1 [{}]
2   "user_id": 3,
3   "bath_time": 10
4 ]

```

The response tab shows a successful `200 OK` status with the following JSON data:

```

1 {
2   "id": 4,
3   "user_id": "3",
4   "time": "10",
5   "updatedAt": "2019-07-24T11:46:37.826Z",
6   "createdAt": "2019-07-24T11:46:37.826Z",
7   "userId": 3
8 }

```

Figura 25 – Adicionando banhos ao usuário

Para recuperar as informações de um usuário, basta consumir o *endpoint* `http://localhost:3001/usuario` com o método *GET*, obtendo o resultado da Figura 26.

The screenshot shows the Postman interface for a GET request to `http://{{users-service-url}}/usuario/3`. The Params tab is selected, showing a key-value pair:

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

The response tab shows a successful `200 OK` status with the following JSON data:

```

1 {
2   "id": 3,
3   "name": "Camilo",
4   "allowedBathTime": "5"
5 }

```

Figura 26 – Recuperar dados do usuário

Conseguimos as informações de todos os banhos dos usuários consumindo o *endpoint* `http://localhost:3001/banho/idDoUsuario`, via *GET*, e conseguiremos o resultado exibido na Figura 27.

```

1 [ 
2   {
3     "id": 4,
4     "user_id": "3",
5     "time": "10",
6     "createdAt": "2019-07-24T11:46:37.826Z",
7     "updatedAt": "2019-07-24T11:46:37.826Z",
8     "userId": "3"
9   },
10  {
11    "id": 5,
12    "user_id": "3",
13    "time": "5",
14    "createdAt": "2019-07-24T11:47:19.779Z",
15    "updatedAt": "2019-07-24T11:47:19.779Z",
16    "userId": "3"
17  }
18 ]

```

Figura 27 – Recuperar dados de banhos do usuário

Finalmente, podemos autenticar os usuários enviando via *POST* os parâmetros para o endpoint `http://localhost:3001/usuario/autorizar`, como a imagem x, obtendo como resposta a Figura 29, para usuário autenticado, e Figura 28, para usuário não autenticado, caso a senha esteja errada.

```

1 [ 
2   {
3     "id": 3,
4     "password": 123
5   }
6 ]

```

```

1 [ 
2   "allowed": false
3 ]

```

Figura 28 – Exemplo de usuário não autorizado

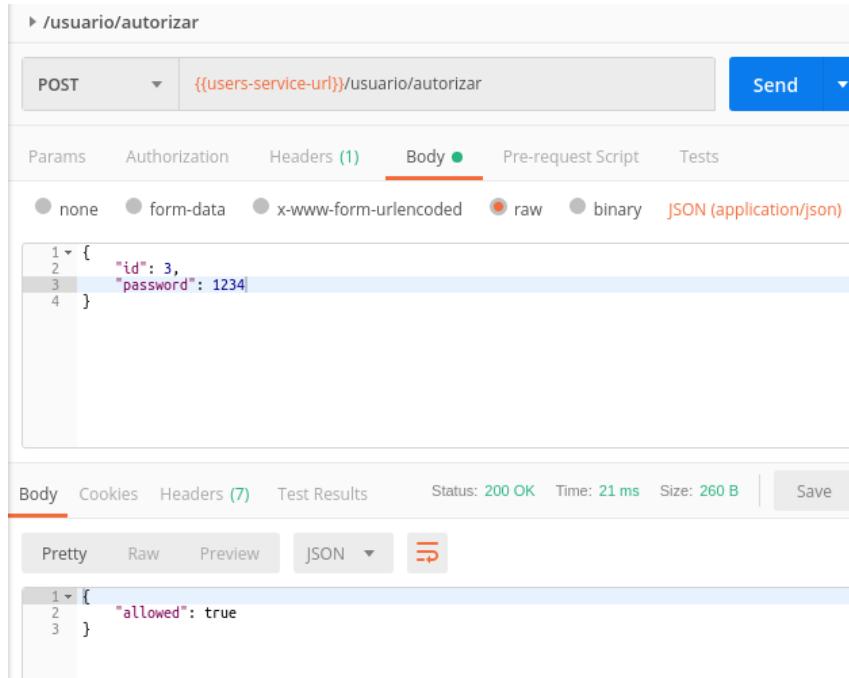


Figura 29 – Exemplo de usuário autorizado

## 4.2 Visualização via Grafana

Ao autenticar um usuário, o atuador é ativado e o sensor de fluxo inicia a medição do fluxo, que é automaticamente enviado para o InfluxDB, e pode ser visualizado via Grafana, os gráficos do grafana podem ser acessados via <http://localhost:3003>, como na Figura 19.

Pode ser observado na Figura 30 um gráfico do fluxo medido pelo sensor no horário de 09h30m até 09h35m, por dois usuários diferentes.

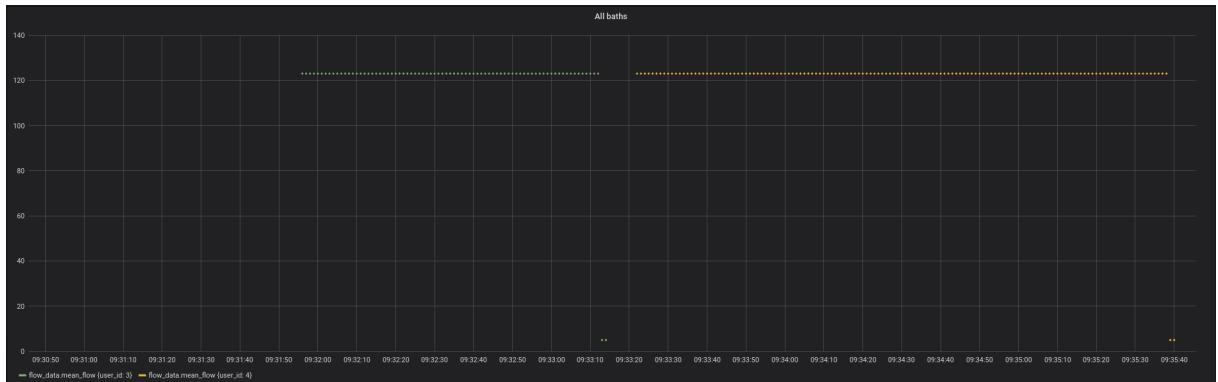


Figura 30 – Exemplo do gráfico no Grafana para usuários diferentes

### 4.3 Estado do atuador via HomeAssistant

O HomeAssistant é acessado via `http://localhost:8123`, ao acessar o link, observamos os estado dos sensores dos usuários cadastrado na Figura 32, que encontram-se em estado *OFF*. Ao digitar corretamente a senha, o estado do sensor muda para *ON*, como observado na Figura 31.

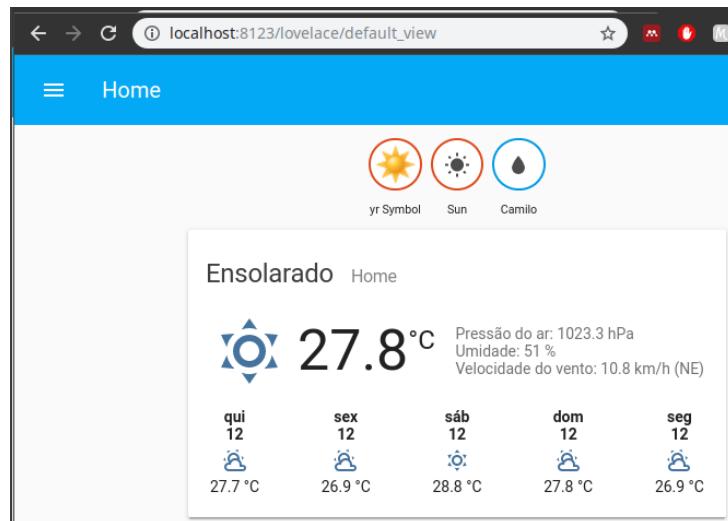


Figura 31 – Exemplo do sensor no HomeAssistant quando ligado

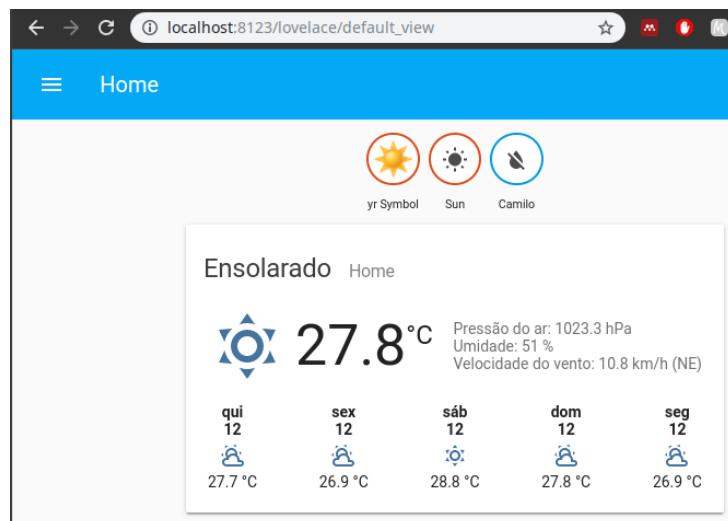


Figura 32 – Exemplo do sensor no HomeAssistant quando desligado

Ao cadastrar um usuário, o seu sensor é inserido no HomeAssistant automaticamente, Figura 33.



Figura 33 – Exemplo de um novo sensor quando novo usuário cadastrado

## 5 Considerações Finais

O propósito deste projeto foi criar um sistema capaz de conseguir monitorar remotamente a utilização da água em uma residência. Este propósito foi atingido como mostrado no Capítulo de Resultados, conseguimos cadastrar, monitorar e editar usuários e banhos com gráficos e estados de sensores em tempo real remotamente.

Para trabalhos futuros, proponho a implementação física em chuveiros com sensores e proteção mais robustos que sejam resistentes ao vapor de água presente ao se tomar banho.

Seria interessante a implementação de outros sistemas de monitoramento com outros sensores mostrando os dados destes sensores no HomeAssistant realizando apenas pequenas modificações e configurações extras. O sistema foi arquitetado e construído com o intuito de facilitar esta inclusão de sensores e reutilização.

## Referências

Almeida Costa, R. *Instituto Politécnico de Viseu*. [S.l.], 2018. Citado 2 vezes nas páginas 19 e 20.

Alves da Silva, M.; Gomes de Santana, C. *REUSO DE ÁGUA: possibilidades de redução do desperdício nas atividades domésticas*. [S.l.], 2014. Disponível em: <<https://www.tratamentodeagua.com.br/wp-content/uploads/2016/05/REUSO-DE-ÁGUA-possibilidades-de-redução-do-desperdício-nas-a>>. Citado na página 13.

BOSCAROLI, C. et al. *Uma reflexão sobre Banco de Dados Orientados a Objetos*. [S.l.], 2006. Citado na página 21.

CHANG, C.-C. et al. A kubernetes-based monitoring platform for dynamic cloud resource provisioning. In: IEEE. *GLOBECOM 2017-2017 IEEE Global Communications Conference*. [S.l.], 2017. p. 1–6. Citado 2 vezes nas páginas 21 e 25.

CHASE, O.; ALMEIDA, F. Sistemas embarcados. *Mídia Eletrônica. Página na internet:<www.sabajovem.org/chase>*, capturado em, 2007. v. 10, n. 11, p. 13, 2007. Citado na página 14.

CROTTI, Y. et al. RaspberryPi e experimentação remota. In: *ICBL2013. International Conference on Interactive Computer aided Blended Learning*. [S.l.: s.n.], 2013. Citado na página 15.

DIÁRIAS, A. et al. ANÁLISE DE CONSUMO E DESPERDÍCIO DE ÁGUA EM. 2007. v. 3, p. 0–5, 2007. Citado na página 11.

FERREIRA, H. S.; HEROSO, L. F.; ZALESKI, R. H. Sistema de monitoramento de consumo de água doméstico com a utilização de um hidrômetro digital. 2014. 2014. Citado na página 11.

FORTY, A. *Objetos de desejo*. [S.l.]: Editora Cosac Naify, 2007. Citado na página 12.

GOMES, L.; SOUSA, F.; VALE, Z. An agent-based IoT system for intelligent energy monitoring in buildings. *IEEE Vehicular Technology Conference*, 2018. IEEE, v. 2018-June, p. 1–5, 2018. ISSN 15502252. Citado na página 19.

JÚNIOR, J. M. M. L.; ARÉAS, R. C.; SENA, A. J. C. Automação residencial: Monitoramento de consumo de energia elétrica e água. *INOVA TEC*, 2017. v. 1, 2017. Citado na página 16.

KODALI, R. K.; SORATKAL, S. R. MQTT based home automation system using ESP8266. *IEEE Region 10 Humanitarian Technology Conference 2016, R10-HTC 2016 - Proceedings*, 2017. IEEE, p. 1–5, 2017. Citado 3 vezes nas páginas 15, 23 e 24.

- LABORATORY, E. *Getting Started with MQTT using Mosquitto*. 2018. <http://embeddedlaboratory.blogspot.com/2018/01/getting-started-with-mqtt-using.html>. Citado na página 24.
- LIGHT, R. A. Mosquitto: server and client implementation of the MQTT protocol. 2017. 2017. Disponível em: <<https://www.theoj.org/joss-papers/joss.00265/10.21105.joss.00265.pdf>>. Citado na página 24.
- LUNDRIGAN, P. et al. *EpiFi: An In-Home Sensor Network Architecture for Epidemiological Studies*. [S.l.], 2017. Disponível em: <<https://arxiv.org/pdf/1709.02233.pdf>>. Citado 3 vezes nas páginas 19, 20 e 21.
- MARTINS, N. A. Sistemas microcontrolados. *Uma abordagem com o Microcontrolador PIC 16F84*. Editora Novatec Ltda, 1<sup>a</sup> edição, 2005. 2005. Citado na página 14.
- NAMIOT, D.; SNEPS-SNEPPE, M. On micro-services architecture. *International Journal of Open Information Technologies*, 2014. v. 2, n. 9, p. 24–27, 2014. Citado na página 22.
- NOOR, S. et al. *Time Series Databases and InfluxDB*. [S.l.], 2017. Citado 2 vezes nas páginas 20 e 21.
- OLIVEIRA, E. *Blog Masterwalker*. 2018. <http://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-teclado-matricial-de-membrana-4x4/>. Accessed: 2019-06-02. Citado 2 vezes nas páginas 16 e 17.
- OLIVEIRA, S. de. *Internet das Coisas com ESP8266, Arduino e Raspberry Pi*. [S.l.]: Novatec Editora, 2017. Citado na página 15.
- PAHL, C.; JAMSHIDI, P. *Microservices: A Systematic Mapping Study*. [S.l.: s.n.], 2016. ISBN 9789897581823. Citado na página 23.
- PERUMAL, T.; SULAIMAN, M. N.; LEONG, C. Y. Internet of Things (IoT) enabled water monitoring system. *2015 IEEE 4th Global Conference on Consumer Electronics, GCCE 2015*, 2016. IEEE, p. 86–87, 2016. Citado 2 vezes nas páginas 11 e 12.
- PICORETI, R. *Portal Vida de Silício*. 2017. <https://portal.vidadesilicio.com.br/teclado-matricial-e-multiplexacao/>. Accessed: 2019-06-02. Citado 3 vezes nas páginas 16, 17 e 18.
- PIETRO ANTONY MARTIN, T. C. R. D. To Docker or Not to Docker: A Security Perspective Blockchain View project Virtualization Security View project To Docker or not to Docker: a security perspective. 2016. 2016. Disponível em: <<https://www.researchgate.net/publication/309965523>>. Citado na página 25.
- REBOUÇAS, A. d. C. Água no brasil: abundância, desperdício e escassez. *Bahia análise & dados*, 2003. v. 13, p. 341–345, 2003. Citado na página 13.
- Risteska Stojkoska, B. L.; TRIVODALIEV, K. V. A review of Internet of Things for smart home: Challenges and solutions. 2017. 2017. Disponível em: <<http://dx.doi.org/10.1016/j.jclepro.2016.10.006>>. Citado na página 11.
- ROQUE, I. R.; SABINO, J. H. M. Sistema de diluição de líquidos concentrados e envaze automático. 2018. 2018. Citado na página 16.

- SAPES, J.; SOLSONA, F. Fingerscanner: Embedding a fingerprint scanner in a raspberry pi. *Sensors (Switzerland)*, 2016. v. 16, n. 2, p. 1–18, 2016. ISSN 14248220. Citado na página 22.
- SILVA, J. A. F. da; LAGO, C. L. do. Módulo eletrônico de controle para válvulas solenóides. *Química Nova*, 2002. SciELO Brasil, v. 25, n. 5, p. 842–843, 2002. Citado na página 18.
- STONES, R.; MATTHEW, N. *Beginning databases with PostgreSQL: from novice to professional*. [S.l.]: Apress, 2006. Citado na página 21.
- TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 2010. v. 14, n. 6, p. 80–83, 2010. ISSN 10897801. Citado na página 22.
- Varela De Souza, M. et al. *UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE INSTITUTO METRÓPOLE DIGITAL PÓS-GRADUAÇÃO EM ENGENHARIA DE SOFTWARE Domótica de baixo custo usando princípios de IoT*. [S.l.], 2016. Citado na página 12.
- VERTIGO. *O que são microserviços*. 2018. <https://vertigo.com.br/o-que-sao-microservicos/>. Citado na página 22.
- ÁGUA e consumo consciente. 2018. <http://www.brasil.gov.br/noticias/educacao-e-ciencia/2010/10/agua-e-consumo-consciente>. Accessed: 2018-12-05. Citado na página 11.

# Apêndices

# APÊNDICE A – Código sistema de usuários

Neste apêndice encontra-se algumas partes importantes do código do Sistema de Usuários. O código inteiro pode ser baixado no *link*: <https://github.com/CamiloAvelar/user-service>

```
import Interactor from './interactor';          CamiloAvelar, a month ago · Padroniza ESG imports/exports e
import usersRep from '../repositories/users.rep';
import bcrypt from 'bcrypt';
import config from '../../config/config';

CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class CreateUserBs extends Interactor {
  constructor(){
    super();
  }

  async execute({ name, password, allowedBathTime }) {
    if (!allowedBathTime) {
      allowedBathTime = 10;
    }

    const hash = await bcrypt.hash( data: password.toString(), saltOrRounds: config.bcrypt.saltRounds);

    const user = await usersRep.createUser( name: { name, password: hash, allowedBathTime });

    if(!user) {
      throw 'Não foi possível criar o usuário!';
    }

    const response = {
      id: user.user_id,
      name: user.user.name,
      allowed_bath_time: user.allowed_bath_time,
    };

    return response;
  }
}

export default CreateUserBs;
```

Figura 34 – Código para criar usuário

```
import Interactor from './interactor';      CamiloAvelar, a month ago - Cria mdw de authorization e rotas de e
import usersRep from '../repositories/users.rep';
import bcrypt from 'bcrypt';
import config from '../../config/config';

CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class EditPasswordBs extends Interactor {
  constructor(){
    super();
  }

  async execute({ id, new_password }) {
    const hash = await bcrypt.hash( data: new_password.toString(), saltOrRounds: config.bcrypt.saltRounds);

    const editedUser = await usersRep.editPassword( id: { id, password: hash });

    const response = editedUser[0] === 1 ? {
      status: 200,
      message: 'Usuário editado com sucesso'
    } :
    {
      status: 500,
      message: 'Erro ao editar usuário'
    };

    return response;
  }
}

export default EditPasswordBs;
```

Figura 35 – Código para editar senha do usuário

```
import Interactor from './interactor';      CamiloAvelar, a month ago · Pad
import usersRep from '../repositories/users.rep';

CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class EditUserTimeBs extends Interactor {
  constructor() {
    super();
  }

  async execute({ id, new_time }) {
    const editedUser = await usersRep.editUserTime({ id, new_time });

    const response = editedUser[0] === 1 ? {
      status: 200,
      message: 'Usuário editado com sucesso'
    } : {
      status: 500,
      message: 'Erro ao editar usuário'
    };

    return response;
  }
}

export default EditUserTimeBs;
```

Figura 36 – Código para editar tempo do usuário

```
import Interactor from './interactor';      CamiloAvelar, a month ago · Padroniza ESl import
import usersRep from '../repositories/users.rep';
import bcrypt from 'bcrypt';

CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class AuthorizeUserBs extends Interactor {
  constructor(){
    super();
  }

  async execute({ id, pass }) {
    const user = await usersRep.getUser( id: { id });

    if(!user) {
      throw {error: 'Usuário não encontrado'};
    }

    const match = await bcrypt.compare( data: pass.toString(), encrypted: user.password);

    return {
      allowed: match
    };
  }
}

export default AuthorizeUserBs;
```

Figura 37 – Código para autorizar usuário

```
import Interactor from './interactor';           CamiloAvelai
import bathsRep from '../repositories/baths.rep';

CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class BathsBs extends Interactor {
  constructor(){
    super();
  }

  async execute({ user_id, bath_time }) {

    const bath = await bathsRep.createBath( user: {
      user_id,
      bath_time
    });

    if(!bath) {
      throw 'Não foi possível cadastrar o banho';
    }

    return bath;
  }

  async getBaths ({ user_id }) {

    const baths = await bathsRep.getBaths( user: {
      user_id,
    });

    return baths;
  }
}

export default BathsBs;
```

Figura 38 – Código para cadastrar banho do usuário

```
import Interactor from './interactor';      CamiloAvelar, a month ago | 1 author (CamiloAvelar)
import usersRep from '../repositories/users.rep';

class GetUserBs extends Interactor{
    constructor(){
        super();
    }

    async execute({ id }) {
        const user = await usersRep.getUser( id: { id });

        if(!user) {
            throw {error: 'Não foi possível localizar o usuário!'};
        }

        const response = {
            id: user.id,
            name: user.name,
            allowedBathTime: user.user_setting.allowed_bath_time
        };

        return response;
    }
}

export default GetUserBs;
```

Figura 39 – Código para recuperar usuários

# APÊNDICE B – Código sistema de comunicação MQTT

Neste apêndice encontra-se algumas partes importantes do código do Sistema de comunicação. O código inteiro pode ser baixado no *link*: <https://github.com/CamiloAvelar/mqtt-logger-service>

```
const EventEmitter = require('events');

You, 22 days ago | 2 authors (CamiloAvelar and others)
class MqttHandler extends EventEmitter {
  constructor() {
    super();
    this mqttClient = null;
    this host = 'mqtt://mosquitto';
    this.username = 'mqtt_user'; // mqtt credentials if these are needed to connect
    this.password = '100200300';
  }

  connect() {
    const self = this;
    this mqttClient = mqtt.connect( brokerUrl: this.host, opts: { username: this.username, password: this.password });

    this mqttClient.on( event: 'error', cb: (err) => {
      console.log( message: err);
      this mqttClient.end();
    });

    this mqttClient.on( event: 'connect', cb: () => {
      console.log( message: `mqtt client connected to ${this.host}`);
    });

    this mqttClient.on( event: 'close', cb: () => {
      console.log( message: `mqtt client disconnected`);
    });

    this mqttClient.on( event: 'message', cb: (topic, message) => {
      const builtMessage = {
        topic,
        message: message.toString()
      };

      self.emit( event: 'messageReceived', args[0]: builtMessage);
    })
  }

  subscribe(topic) {
    this mqttClient.subscribe(topic, {qos: 0});
  }

  sendMessage(message, topic) {
    this mqttClient.publish(topic, message);
  }
}

module.exports = MqttHandler;
```

Figura 40 – Código para comunicação mqtt

```
const requestService = require( id: './requestService');      CamiloAvelar, a month ago · A
CamiloAvelar, a month ago | 1 author (CamiloAvelar)
class timeHandler {
  constructor(mqttClient) {
    this.mqttClient = mqttClient;
    this.allowedTime;
    this.interval;
    this.nowDate;
    this.endDate;
    this.userId;
  }

  countTime(allowedTime, id) {
    this.userId = id;
    this.allowedTime = allowedTime;
    this.nowDate = new Date();

    this.interval = setInterval( callback: () => {
      this.mqttClient.sendMessage('stop', 'actuator');
      this.clearBathInterval();
    }, ms: this.allowedTime);
  }

  async endTime() {
    this.clearBathInterval();
    this.endDate = new Date();

    const bathTime = this.endDate - this.nowDate;

    console.log( message: 'BATHTIME>>>', optionalParams[0]: bathTime);
    await this._requestUserService( time: bathTime);
  }

  clearBathInterval(){
    clearInterval( intervalId: this.interval);
  }

  async _requestUserService(time) {
    const requestOptions = {
      type: 'POST',
      endpoint: 'banho',
      body: {
        user_id: this.userId,
        bath_time: time
      },
    };

    return await requestService.userRequest( type: requestOptions);
  }
}
```

Figura 41 – Código que lida com o tempo do banho

```
const requestService = require( id: './requestService' );

You, a few seconds ago | 2 authors (CamiloAvelar and others)
class KeysHandler {
  constructor(mqttClient) {
    this.keyBuffer = '';
    this.working = false;
    this.gettingPassword = false;
    this.userId;
    this.mqttClient = mqttClient;
  }

  async handle(message) {
    if(message === '*') {
      this.working = !this.working;
      this.gettingPassword = false;
      return;
    }

    if((this.working || this.gettingPassword) && message !== '#') {
      this.keyBuffer += message;
    }

    if(message === '#') {
      this.working = false;
      try {
        const response = await this._requestUserService();
        console.log( message: response);
        this.gettingPassword ? (response.allowed ? this.mqttClient.sendMessage('start', 'actuator') : console.log( message: 'NAO AUTORIZADO')) : this.mqttClient.sendMessage(JSON.stringify(response), 'user');
        this.gettingPassword = !this.gettingPassword;
      } catch (err) {
        console.log( message: err.message)
      } finally {
        this.keyBuffer = '';
      }
    }
  }

  async _requestUserService() { ... }
}

module.exports = KeysHandler| CamiloAvelar · a month ago · first commit
```

Figura 42 – Código que lida com as teclas apertadas no teclado numérico

```

const Influx = require( id: 'influx' );

You, 22 days ago | 2 authors (CamiloAvelar and others)
class InfluxHandler {
  constructor() {
    this.influx;
    this.host = 'influxdb';
    this.database = 'water_flow_data'
  }

  connect() {
    this.influx = new Influx.InfluxDB( options: {
      host: this.host,
      database: this.database,
      schema: [{
        measurement: 'flow_data',
        fields: { flow: Influx.FieldType.INTEGER },
        tags: ['user_id']
      }]
    });
    this.influx.getDatabaseNames()
      .then( onfulfilled: names => {
        if(!names.includes( searchElement: this.database)) {
          return this.influx.createDatabase( databaseName: this.database);
        }
      })
      .then( onfulfilled: () => {
        console.log( message: 'InfluxDB connected!');
      })
  }

  writePoints(id, data) {
    console.log( message: id, optionalParams[0]: data);
    return this.influx.writePoints( points: [
      measurement: 'flow_data',
      tags: { user_id: id },
      fields: { flow: data }
    ], options: {
      database: this.database,
      precision: 's',
    }).catch( onrejected: err => console.error( message: `Error saving data to InfluxDB! ${err.stack}`))
      .then( onfulfilled: () => {return 'Dados salvos no banco'});
  }
}

module.exports = InfluxHandler;      CamiloAvelar, a month ago · first commit

```

Figura 43 – Código que lida com a comunicação com o InfluxDB

# APÊNDICE C – Docker-compose.yml

```

version: "3.5"

services:
  user:
    user:
      build: ./user-service
      command: gulp
      depends_on:
        - postgres
    volumes:
      - ./user-service:/app
    ports:
      - "3001:3001"
    networks:
      - default

  mqtt-logger:
    build: ./mqtt-logger
    command: npm start
    depends_on:
      - mosquitto
    volumes:
      - ./mqtt-logger:/app
    ports:
      - "3002:3002"

  postgres:
    image: postgres
    restart: always
    environment:
      POSTGRES_USER: pi
      POSTGRES_PASSWORD: 100200300
      POSTGRES_DB: postgres

```

```
CONFIGS: "listen_addresses: '*'"
volumes:
  - ./postgres/data:/var/lib/postgresql/data
expose:
  - "5432"
ports:
  - "5432:5432"
networks:
  - default

migration:
  image: src_user:latest
  command: [ "./wait-for-it/wait-for-it.sh", "postgres:5432" ]
  links:
    - postgres
depends_on:
  - postgres

homeassistant:
  container_name: homeassistant
  restart: unless-stopped
  image: homeassistant/home-assistant
  volumes:
    - ./homeassistant:/config
  depends_on:
    - mosquitto
  network_mode: host
  privileged: true
  expose:
    - "8123"
  ports:
    - "8123:8123"

influxdb:
  image: influxdb:latest
  container_name: influxdb
  restart: always
  ports:
    - "8083:8083"
```

```

      - "8086:8086"
      - "8090:8090"

  volumes:
    # Data persistency
    # sudo mkdir -p /srv/docker/influxdb/data
    - ./influxdb/data:/var/lib/influxdb

grafana:
  image: grafana/grafana
  container_name: grafana
  restart: always
  ports:
    - "3003:3000"
  links:
    - influxdb
  volumes:
    # Data persistency
    # sudo mkdir -p /srv/docker/grafana/data; chown 465:465 ./grafana/data:/var/lib/grafana

mosquitto:
  image: eclipse-mosquitto
  hostname: mosquitto
  container_name: mosquitto
  restart: always
  user: 1883:1883
  expose:
    - "1883"
    - "9001"
  ports:
    - "1883:1883"
    - "9001:9001"
  volumes:
    - ./mosquitto/config:/mosquitto/config
    # sudo chown 1883:1884 /mosquitto/logs
    - ./mosquitto/logs:/mosquitto/logs

  networks:
    - default

networks:

```

```
default:  
  driver: bridge  
  ipam:  
    config:  
      - subnet: 172.18.1.0/24
```