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 monitoramento de água utilizando comunicação MQTT e arquitetura de microsserviços: uma solução IoT.

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, agosto de 2019

Camilo Esteves Mendes de Avelar

Domótica aplicada no monitoramento de água utilizando comunicação MQTT e arquitetura de microsserviços: 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, agosto de 2019

A948d Avelar, Camilo E. Mendes.
Domótica aplicada no monitoramento de água utilizando comunicação MQTT e arquitetura de microsserviços [manuscrito]: uma solução IoT / Camilo E. Mendes Avelar. - 2019.

62f.: il.: color; tabs.

Orientador: Prof. MSc. Filipe A. S. Rocha.
Coorientador: Prof. Dr. Agnaldo José da R. Reis.

Monografia (Graduação). Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Controle e Automação e Técnicas Fundamentais.

1. Internet das coisas (IOC). 2. Message Queuing Telemetry Transport (MQQT). 3. Arquitetura de software - Microsserviços. I. Rocha, Filipe A. S.. II. Reis, Agnaldo José da R.. III. Universidade Federal de Ouro Preto. IV. Titulo.

Catalogação: ficha.sisbin@ufop.edu.br CDU: 658.5

A comissão avaliadora constituída pelo Pesquisador Filipe Augusto Santos Rocha e pelos Professores Agnaldo José da Rocha Reis, Wolmar Araujo Neto e Thomas Vargas Barsante e Pinto atesta que a monografia intitulada "Domótica Aplicada no Monitoramento de Água Utilizando Comunicação MQTT e Arquitetura de Microsserviços: Uma Solução IoT" foi defendida e aprovada em 01 de agosto de 2019.

Pesquisador Me. Filipe Augusto Santos Rocha - Orientador

Prof. Dr. Agnaldo José da Rocha Reis – Coorientador

Prof. Dr. Wolmar Araujo Neto – Professor Convidado

Prof. Thomas Vargas Barsante e Pinto – Professor Convidado

Resumo

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. Com isto, existe uma crescente demanda para aumentar a economia de água em todo o mundo. Com o avanço da tecnologia e seu fácil acesso, tornou-se mais simples desenvolver sistemas visando uma melhora nesta economia. Este trabalho tem o propósito de criar um sistema de Domótica para controle de água simulando dados de um sensor de fluxo de água, uma válvula solenoide como atuador e utilizando um RaspberryPi. O sistema será capaz de monitorar o uso de água, mostrando os dados em gráfico e salvando-os em um banco de dados. Será construído um sistema de controle de usuários que será capaz de criar, editar e salvar dados de clientes para integrar no sistema. A comunicação entre o sensor e o Raspberry Pi será feita via MQTT. Os sistemas de comunicação e usuários serão construídos de acordo com a arquitetura de microsserviços, sendo dois sistemas diferentes que se completam para realizar as operações necessárias. Os sistemas de usuário e comunicação, juntamente com os softwares utilizados para monitoramento serão configurados, testados e os resultados serão apresentados no decorrer no trabalho. A validação do projeto será feita através de dados emulados a partir de um sistema criado para simular as informações dos sensores.

Palavras-chaves: iot, monitoramento, microsserviços, mqtt, node, domótica, raspberry pi, economia de água.

Abstract

Water saving is a recurring issue that has long ceased to be restricted to arid and desert regions with low per capita water availability. Hence, there is a growing demand to increase water savings worldwide. Since the tecnology has been modernised, it has become simpler to develop systems aimed at improving water savings. This paper has the porpouse of create a Domotic system for water control simulating data from a water flux sensor, solenoid valve as actuator and using a Raspberry Pi. The system will be able to monitorize the use of the water, showing the data and saving it in a database. A user control system will also be created that will be able to create, edit and save data about the system users to integrate with the whole system. All the comunications between the sensor and the Raspberry Pi will be done via MQTT and the comunication system and the users system will be built using the microservices architeture, being two different systems that comunicate with each other to do the necessary operations. The users and comunication services, together with the softwares used for the monitoring will be set up, tested and the results will be shown in this paper.

Key-words: iot, monitoring, microservices, mqtt, node, domotic, raspberry pi, water saving.

Lista de ilustrações

Figura 1 Arquitetura básica de microprocessadores e microcontroladores. 15
Figura 2 Raspberry Pi. 16
Figura 3 Sensor de fluxo de água YF-S201. 17
Figura 4 Teclado Matricial de Membrana. 17
Figura 5 Circuito do teclado. 18
Figura 6 Pinagem do teclado. 18
Figura 7 Válvula Solenoide. 19
Figura 8 Dashboard genérico do HomeAssistant 20
Figura 9 Exemplo arquitetura de microsserviços. 23
Figura 10 Exemplo da arquitetura do MQTT. 24

Figura 11 Arquitetura geral do projeto. 26
Figura 12 Modelo Entidade Relacionamento do PostgreSQL 29
Figura 13 Modelo das tabelas criadas no banco de dados 31
Figura 14 Divisão dos arquivos do Sistema de Usuários 33
Figura 15 Exemplo do sistema de simulação 35
Figura 16 Página inicial do HomeAssistant acessado por um cliente conectado na mesma rede local do servidor Hassbian. 36
Figura 17 Página inicial do Grafana. 37
Figura 18 Configuração do gráfico no Grafana. 38

Figura 19 Cadastro de usuários. 39
Figura 20 Edição de tempo permitido do usuário. 40
Figura 21 Edição de senha do usuário. 40
Figura 22 Adicionando banhos ao usuário. 41
Figura 23 Recuperar dados do usuário. 41
Figura 24 Recuperar dados de banhos do usuário. 42
Figura 25 Exemplo de usuário não autorizado. 42
Figura 26 Exemplo de usuário autorizado. 43
Figura 27 Exemplo do gráfico no Grafana para usuários diferentes. 44
Figura 28 Exemplo do sensor no HomeAssistant quando ligado. 44
Figura 29 Exemplo do sensor no HomeAssistant quando desligado. 44
Figura 30 Exemplo de um novo sensor quando um novo usuário é cadastrado. . . 45

Lista de tabelas

Tabela 1 Tabela de disposição dos pinos do teclado numérico. 19

Tabela 2 Colunas da tabela *users*. 30
Tabela 3 Colunas da tabela *user_settings*. 30
Tabela 4 Colunas da tabela *user_baths*. 30
Tabela 5 *Endpoints* do sistema de usuários. 34
Tabela 6 Tópicos do *broker* MQTT. 34

Lista de códigos

3.1 Exemplo de configuração *headless* 28
3.2 Exemplo de configuração do IP estático 29
3.3 Exemplo do código de configuração do InfluxDB 32
A.1 Exemplo do código do *interactor* de criação do usuário 51
A.2 Exemplo do código do *interactor* de edição de senha de usuários 52
A.3 Exemplo do código do *interactor* de edição de tempo de banho dos usuários . 53
A.4 Exemplo do código do *interactor* de autorização de usuários 54
A.5 Exemplo do código do *interactor* de cadastro de banhos 55
A.6 Exemplo do código do *interactor* para recuperar informações dos usuários . . . 56
B.1 Exemplo do código de comunicação MQTT 57
B.2 Exemplo do código responsável por lidar com informações do tempo de banho 58
B.3 Exemplo do código que lida com a comunicação com o InfluxDB 60
B.4 Código principal do sistema 61

Lista de abreviaturas e siglas

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

Sumário

Lista de códigos **8**

1 Introdução **12**
1.1 Objetivos 13
1.1.1 Objetivos específicos 13
1.2 Organização do trabalho 14

2 Materiais e Softwares **15**
2.1 Microcontroladores e Microprocessadores 15
2.1.1 Raspberry Pi 16
2.2 Sensores e atuadores 16
2.2.1 Sensor de fluxo YF-S201 16
2.2.2 Teclado matricial de membrana 17
2.2.3 Válvula solenoide 19
2.3 Softwares 20
2.3.1 HomeAssistant 20
2.3.2 Banco de dados de séries temporais 21
2.3.2.1 InfluxDB 21
2.3.2.2 Grafana 22
2.3.3 Banco de dados relacional 22
2.3.3.1 PostgreSQL 22
2.3.4 Node.JS 22
2.3.5 Arquitetura de microsserviços 23
2.3.6 Protocolo de comunicação MQTT 23
2.3.6.1 MQTT Mosquitto 25

3 Metodologia **26**
3.1 Funcionamento do sistema 26
3.1.1 Parâmetros de operação 27
3.1.2 Dados gerados 27
3.1.3 Ligar/desligar atuador 27
3.2 Configuração do Raspberry Pi 28
3.3 Criação das tabelas no PostgreSQL 29
3.4 Configuração do InfluxDB 31
3.5 Microsserviços 32
3.5.1 *Clean Architecture* 32

3.5.2 Sistema de usuários 33
3.5.3 Sistema de comunicação MQTT 34
3.5.4 Sistema de simulação de dados 35
3.6 Configuração do HomeAssistant 36
3.6.1 Sensores 36
3.7 Configuração do Grafana 37

4 Experimentos e Resultados 39
4.1 Manipulação de usuários 39
4.2 Visualização via Grafana 43
4.3 Estado do atuador via HomeAssistant 43

5 Considerações Finais 46

Referências 47

Apêndices 50

APÊNDICE A Código sistema de usuários 51

APÊNDICE B Código sistema de comunicação MQTT 57

1 Introdução

Cerca de 70% 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 reduzida de recursos hídricos 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 ambiental por todo o mundo (FERREIRA et al., 2007).

Pesquisas em 2007 indicaram 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 e 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. Isto 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 IoT (Internet of Things ou Internet das Coisas), evidenciado no parágrafo a seguir. PERUMAL; SULAIMAN; LEONG (2016) descrevem a Internet das Coisas como sendo um método para conectar coisas como sensores e interruptores, em torno do ambiente e realizar um certo tipo de troca de mensagem entre eles, integrando-os.

O IoT representa uma rede mundial de dispositivos interconectados e unicamente endereçados. Esta conectividade entre sensores e atuadores permite o compartilhamento de informações entre plataformas através de um *framework* unificado, desenvolvendo uma comum capacidade de criar aplicações inovadoras (Risteska Stojkoska; TRIVODALIEV, 2017). Uma das grandes influências do IoT é no monitoramento do ambiente em que vivemos, sistemas de alarmes e análise de dados ambientais (PERUMAL; SULAIMAN;

LEONG, 2016).

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 a 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, como os celulares se conectando à dispositivos como eletrodomésticos (Varela De Souza et al., 2016).

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 a seus usuários como a otimização e gestão de recursos, praticidade, segurança, controle e monitoramento remoto dos dispositivos automatizados.

1.1 Objetivos

O objetivo deste trabalho é desenvolver um sistema modular, de baixo custo, baseado em microsserviços e em IoT para monitoramento e controle de consumo de água em chuveiros elétricos residenciais através da integração entre dispositivos e camadas de softwares.

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

Será criado também um sistema que emulará os dados provindos do sensor, atuador e teclado utilizados.

1.1.1 Objetivos específicos

- Armazenar dados para levantamentos estatísticos;
- Implementar o sistema de identificação e controle de usuários;
- Implementar o sistema de interface;
- Implementar o sistema de atuação;
- Implementar o sistema de emulação de dados;
- Integrar todos os sistemas a fim de garantir as funcionalidades do projeto;
- Monitorar em ambiente online os parâmetros do sistema;

1.2 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.

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 trata dos experimentos realizados no sistema e seus resultados.

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

2 Materiais e Softwares

Nas seções a seguir são apresentados os *hardwares* e *softwares* utilizados na construção do sistema proposto.

2.1 Microcontroladores e Microprocessadores

Buscando aumentar a 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).

Por sua vez, 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, temporizadores/contadores e um microprocessador (CHASE; ALMEIDA, 2007).

Na Figura 1, pode-se observar algumas das diferenças entre Microprocessadores e Microcontroladores. O microcontrolador embarca todos os componentes necessários ao seu funcionamento, enquanto o microprocessador precisa de comunicar externamente com os periféricos similares.

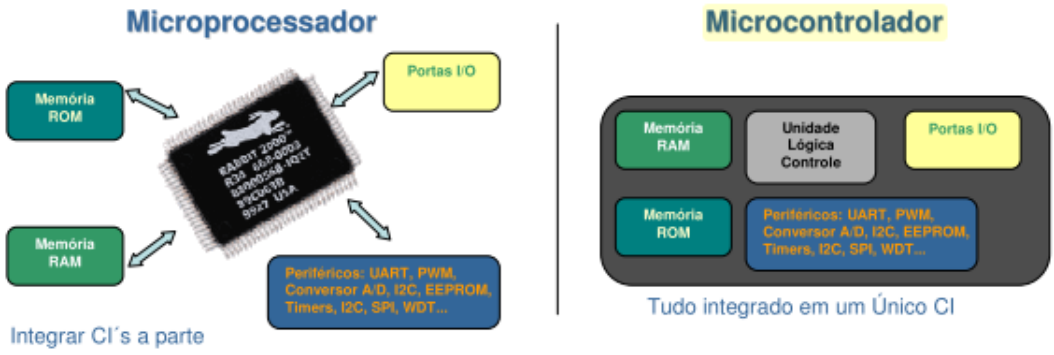

Figura 1 – Arquitetura básica de microprocessadores e microcontroladores.

Fonte: CHASE; ALMEIDA (2007)

2.1.1 Raspberry Pi

O Raspberry Pi (Figura 2) é um minicomputador criado pela Raspberry Pi Foundation. Seu objetivo é estimular o ensino da ciência da computação nas escolas e universidades. Apesar do Raspberry Pi possuir o hardware em uma única placa eletrônica de tamanho reduzido, seu potencial de processamento é significativo (CROTTI et al., 2013). 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 2 – Raspberry Pi.

2.2 Sensores e atuadores

Nesta seção serão apresentados os sensores e atuadores a terem seus sinais utilizados no projeto.

2.2.1 Sensor de fluxo YF-S201

O sensor YF-S201 (Figura 3) é um sensor do tipo turbina que mede a quantidade de líquido que passa pela tubulação, girando aletas que geram pulsos de onda quadrada através de um sensor de efeito Hall (ROQUE; SABINO, 2018). O sensor usa esse efeito para enviar um sinal PWM (*Pulse Width Modulation*) e, através da contagem deste sinal é possível mensurar a quantidade de água que passa pela turbina no interior do sensor. (JÚNIOR; ARÊAS; SENA, 2017)

Figura 3 – Sensor de fluxo de água YF-S201.

2.2.2 Teclado matricial de membrana

Teclados permitem que usuários insiram informações em diversos tipos de sistemas, como computadores, calculadoras, controles remotos entre outros (OLIVEIRA, 2018). O Teclado Matricial de Membrana 4X4 (Figura 4) foi desenvolvido com a finalidade de facilitar a entrada de dados em projetos com plataformas microcontrolada (PICORETI, 2017). Este teclado possui 16 teclas, sendo 10 teclas são numéricas, 4 literais e 2 caracteres especiais.

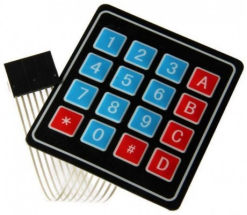

Figura 4 – Teclado Matricial de Membrana.

Fonte: OLIVEIRA (2018)

As teclas estão dispostas em 4 linhas por 4 colunas e o teclado possui um conector de 8 pinos para ligação. Quando um botão do teclado é pressionado, ele conecta a linha com a coluna na qual está ligado, gerando um sinal nos pinos referente àquela linha/coluna. Este sinal permite a identificação da tecla apertada pelo sistema. O circuito do teclado está exemplificado na Figura 5.

A Tabela 1 possui as informações da distribuição dos pinos em tabelas e colunas, exemplificada na Figura 6.

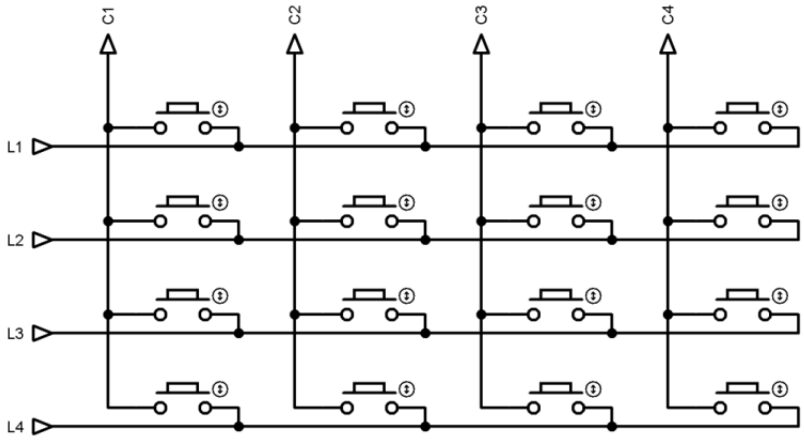

Figura 5 – Circuito do teclado.

Fonte: PICORETI (2017)

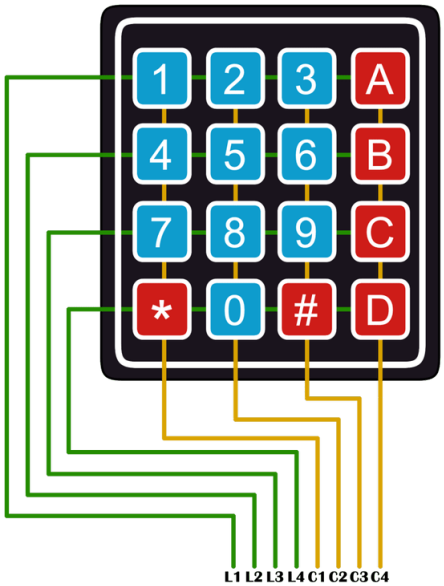

Figura 6 – Pinagem do teclado.

Fonte: PICORETI (2017)

Tabela 1: Tabela de disposição dos pinos do teclado numérico.

Pino	Localização
1	Linha 1
2	Linha 2
3	Linha 3
4	Linha 4
5	Coluna 1
6	Coluna 2
7	Coluna 3
8	Coluna 4

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, 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 (SILVA; LAGO, 2002). Elas são de pequenas dimensões e requerem baixa tensão e corrente de acionamento.

A válvula solenoide pode ser vista na Figura 7.

Figura 7 – Válvula Solenoide.

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 é aplicada sobre um solenoide, criando um campo magnético que desloca um núcleo ferromagnético móvel, causando a alteração do estado da válvula. O núcleo ferromagnético comprime uma mola que é a responsável por retornar o núcleo a sua posição original quando a corrente elétrica é interrompida.

2.3 Softwares

Os softwares que foram utilizados para a implementação do sistema estão presentes nesta seção.

2.3.1 HomeAssistant

O HomeAssistant é um plataforma de automação escrita em Python. Inclui componentes contribuídos por usuários que permite a interface com Web Services e dispositivos como sensores, microcontroladores e assistentes virtuais (LUNDRIGAN et al., 2017). Em seu núcleo, o HomeAssistant é um protocolo de troca de mensagens que facilita a comunicação entre dispositivos e componentes funcionais na rede. Provendo simples abstrações de componentes de automação residencial como sensores, câmeras, *players* de música, etc.

A Figura 8 apresenta um exemplo de tela inicial 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 e *switches* para acionamento de interruptores e iluminação.

Figura 8 – Dashboard genérico do HomeAssistant

Fonte: Adaptado de Almeida Costa (2018)

O HomeAssistant tem suporte para diversos tipos de protocolos *wireless*, como BLE, ZigBee, Z-Wave e Wi-Fi. Conta também com um RESTful API e suporta HTTP, MQTT, TCP *sockets* e componentes customizados. Estes componentes customizados permitem aos

usuários adicionar funções próprias no HomeAssistant sem a necessidade de mudar o seu código fonte. Isto torna a integração de novos dispositivos e sensores muito mais fácil com o HomeAssistant (GOMES; SOUSA; VALE, 2018).

Tendo em vista a gama de protocolos sem fio disponíveis no HomeAssistant, escolhemos o padrão Wi-Fi pois segundo LUNDRIGAN et al. (2017):

- O custo dos equipamentos com padrão Wi-Fi é reduzido.
- Wi-Fi é o mais difuso dos protocolos wireless.
- Sensores Wi-Fi tem integração facilitada com demais equipamentos residenciais.

O HomeAssistant pode ser instalado em qualquer sistema operacional devido ao fato de ser muito pequeno e leve. O que o faz ser compatível para o uso no Rasberry Pi como um hub de automação pequeno e barato. É importante lembrar que o HomeAssistant age apenas como uma central de controle que pode informar outros serviços, como o Philips Hue ou o Nest, que são produtos para automação residencial, para realizar alguma função (Almeida Costa, 2018). O Home Assistant é gratuito e de fácil configuração.

2.3.2 Banco de dados de séries temporais

Um Banco de Dados de Séries Temporais, do inglês *Temporal Series Database* (TSDB), é um tipo de banco otimizado para dados coletados no tempo. É implementado especificamente para lidar com métricas, eventos ou medidas que variam no tempo. Um TSDB permite o usuário criar, enumerar, alterar, destruir e organizar várias séries temporais de métodos mais eficientes. Atualmente, a maioria das empresas estão gerando um grande volume de dados sobre métricas e eventos que são mapeados no tempo, aumentando a relevância de tal arquitetura (NOOR et al., 2017).

Ainda segundo NOOR et al. (2017), aplicações comuns para os TSDBs são IoT, DevOps e Analise de Dados. Alguns casos de uso incluem monitoramento de sistemas de *software* como máquinas virtuais, monitoramento de sistemas físicos como dispositivos, ambiente, sistemas de automação residencial, dentre outros.

2.3.2.1 InfluxDB

O InfluxDB é o Banco de Dados de Séries Temporais usado neste projeto, sendo o mais apto a guardar os dados de fluxo no tempo. (LUNDRIGAN et al., 2017)

É um projeto *open-source* com o opcional de armazenamento pago em nuvem desenvolvido pela empresa InfluxData. É escrito na linguagem de programação Go e baseado em 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 destas séries temporais a partir do InfluxDB exibindo os dados de maneira gráfica (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 identificadores únicos, que compõem a base de dados. Conceitos como integridade referencial de dados – que garante sincronia de dados entre tabelas – e chaves primárias estão presentes, garantindo que um conjunto de informações possa ser representado de maneira consistente, independente da forma de acesso (BOSCARIOLI et al., 2006).

2.3.3.1 PostgreSQL

O PostgreSQL é uma implementação de um banco de dados relacional, de código aberto e de custo gratuito (STONES; MATTHEW, 2006). O PostgreSQL pode ser usado com diversas linguagens de programação, como por exemplo: Python, Javascript, Java e C++.

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

O Node.JS, também chamado de Node, é um ambiente de servidor que utiliza a linguagem de programação JavaScript. É baseado no *runtime* do Google, chamado de motor V8. O V8 e o Node são implementados em C e C++, focados no desempenho 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).

O 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 de suporte ao Node (TILKOV; VINOSKI, 2010).

2.3.5 Arquitetura de microsserviços

Seguindo a definição de NAMIOT; SNEPS-SNEPPE (2014), a arquitetura de microsserviços trata do 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 9.

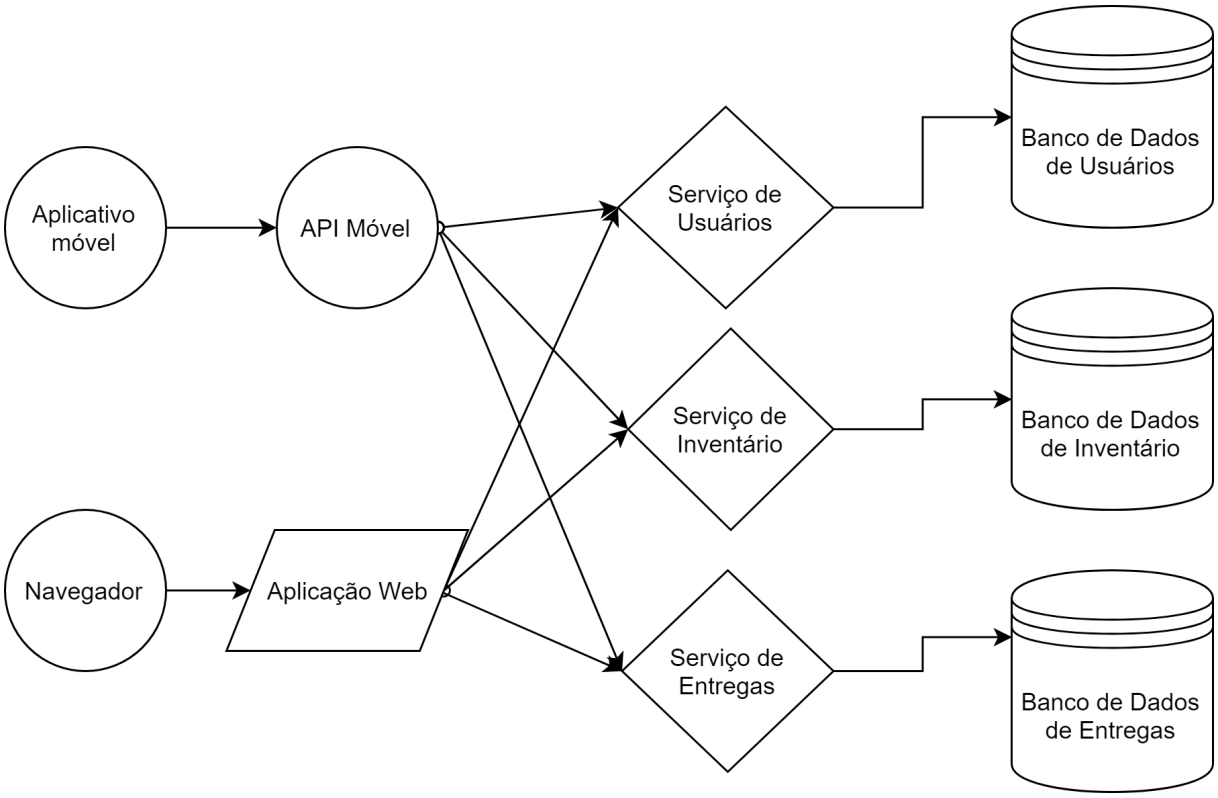

Figura 9 – Exemplo arquitetura de microsserviços.

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)

2.3.6 Protocolo de comunicação MQTT

MQTT significa *Message Queuing Telemetry Transport*, traduzido para o português como Transporte de Telemetria de Enfileiramento de Mensagens. É um protocolo de transporte leve que otimiza o uso da a largura de banda de rede[1]. O MQTT trabalha sobre

[1] `http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html`

o protocolo 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 comumente tem recursos e capacidades limitados.

É um protocolo inicialmente desenvolvido pela IBM[2] em 1999, sendo recentemente reconhecido como padrão pela OASIS (Organizarion for the Advancement of Structured Information Standards)[3].

KODALI; SORATKAL (2017) definiu o MQTT como um protocolo baseado em *publisher/subscriber*. Qualquer conexão MQTT envolve dois tipos de agentes, os clientes e um *broker*, ou servidor. Qualquer dispositivo ou programa que é conectado pela rede e troca mensagens através do MQTT é chamado de cliente. Um cliente pode ser tanto um *publisher* e/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. Ele aceita e transmite as mensagens através de múltiplos clientes conectados à ele.

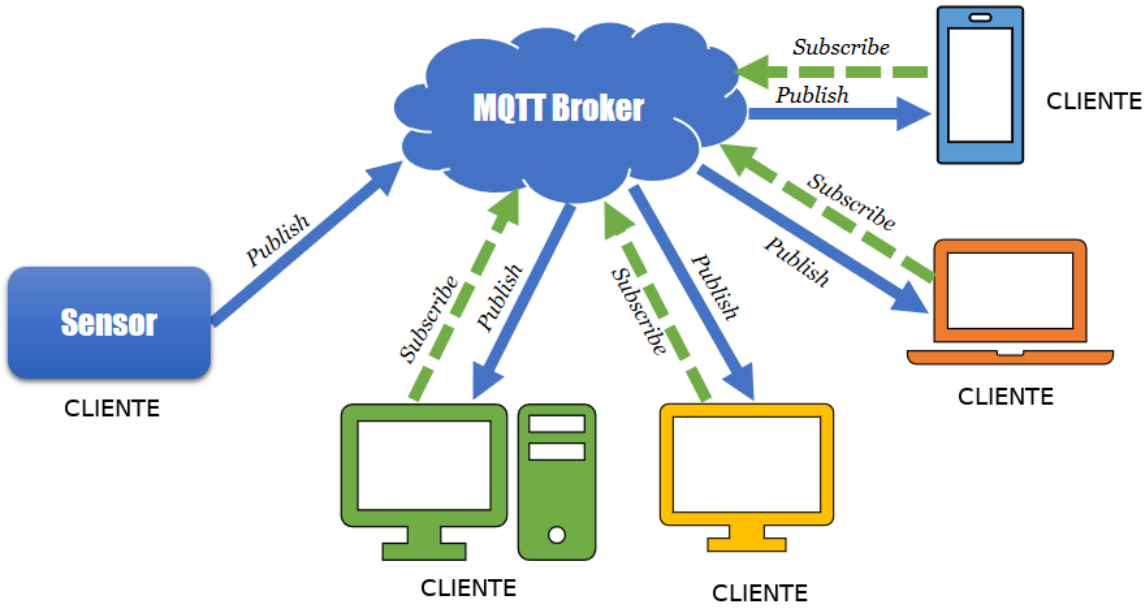

Figura 10 – Exemplo da arquitetura do MQTT.

Fonte: LABORATORY (2018)

[2] http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt
[3] https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard

Dispositivos como sensores e celulares podem ser vistos como clientes do ponto de vista da arquitetura MQTT. Quando um cliente tem alguma informação para transmitir, ele publica o dado para o *broker*.

A Figura 10 apresenta um exemplo de arquitetura MQTT comum. O *broker* MQTT, ou servidor MQTT é 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.

2.3.6.1 MQTT Mosquitto

O Mosquitto é um *broker* MQTT de código aberto (KODALI; SORATKAL, 2017) que entrega uma implementação de servidor e cliente MQTT. Utiliza o modelo *publisher/-subscriber*, tem uma baixa utilização de rede e pode ser implementado em dispositivos de baixo custo como microcontroladores. (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.

3 Metodologia

Este Capítulo apresenta a estrutura geral do sistema, a configuração inicial do Raspberry Pi, 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, foram configurados o HomeAssistant e Grafana.

A validação do projeto será feita através de um sistema criado para emular os dados vindos dos sensores, atuadores e teclado.

3.1 Funcionamento do sistema

Nesta seção estão descritos os parâmetros utilizados pelo sistema, os dados que são gerados, assim como as respostas esperadas. A arquitetura geral do sistema pode ser visualizada na Figura 11, mostrando os sistemas desenvolvidos, os tópicos do servidor MQTT e os sistemas que implementam os bancos de dados.

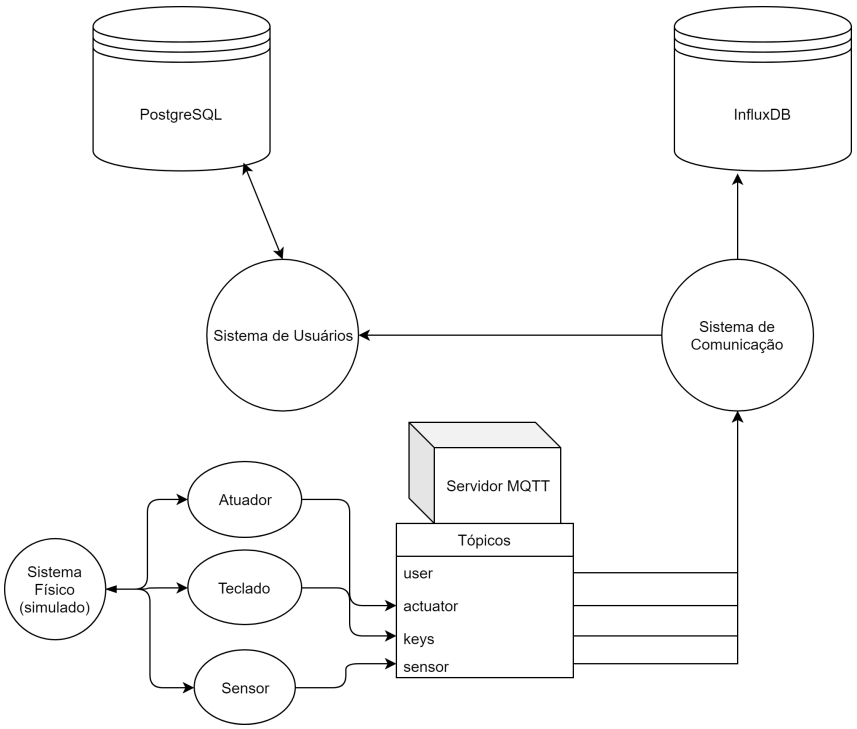

Figura 11 – Arquitetura geral do projeto.

3.1.1 Parâmetros de operação

Os parâmetros de operação são utilizados pelo sistema para garantir o funcionamento esperado. Estes parâmetros são:

- Tempo de banho permitido: é o tempo máximo de banho permitido para cada usuário;
- Estado do chuveiro: ligado ou desligado;
- Estado do usuário: autorizado ou não autorizado;

Ao se criar um usuário é necessário informar três dados: nome do usuário, senha e tempo de banho permitido. A senha é utilizada para autorizar o usuário a tomar o banho, para iniciar o banho, o usuário deve digitar sua senha no teclado numérico. O tempo de banho permitido é o tempo máximo em que o chuveiro pode ficar ligado; ao exceder este limite, o atuador irá interromper o fluxo de água.

O estado do chuveiro poderá ser observado no HomeAssistant, que dirá se o usuário está com o chuveiro ligado ou desligado. O HomeAssistant proverá informações personalizadas de acordo com o usuário.

3.1.2 Dados gerados

O usuário, ao digitar a sua senha correta no teclado numérico, liberará o atuador e o fluxo de água começará a passar pelo sensor de fluxo. Este sensor irá gerar dados que dizem informações a respeito do fluxo volumétrico de água passado por ele.

Os dados do sensor são gravados no InfluxDB e poderão ser vistos no Grafana em forma de gráfico, sendo possível distinguir os usuários. O sistema, ao perceber uma parada no fluxo, ou se o tempo de banho ultrapassar o limite de banho do usuário, fecha o atuador e contabiliza o tempo total do banho tomado, salvando-o no Banco de Dados PostgreSQL.

3.1.3 Ligar/desligar atuador

Para ligar o atuador, o sistema necessita identificar o usuário e autenticá-lo. Esta identificação e autenticação é realizada via teclado numérico. O usuário é identificado pelo seu id, que é fornecido no momento do seu cadastro. Para realizar a identificação, o usuário deve pressionar a tecla * seguido pelo seu id e #. Ao realizar a consulta, o sistema retorna o nome do usuário e requisita a senha.

Após digitar a senha, a tecla # deve ser pressionada e o sistema irá comparar a senha digitada com a cadastrada no banco de dados. Caso as senhas forem iguais, o atuador

é ligado; caso forem diferentes, o sistema retorna que o usuário não está autenticado e não liga o atuador.

A Figura 15 mostra este fluxo de digitação, assim como a identificação do usuário pelo id.

3.2 Configuração do Raspberry Pi

Para a configuração inicial do Raspberry Pi é preciso fazer o download de um sistema operacional compatível com o microcontrolador. Para o sistema deste projeto, utilizamos o Hassbian[1], que foi instalado no cartão SD a ser inserido no Raspberry Pi.

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

O Raspberry Pi foi configurado para ser *headless*[2], o que faz com que ele não necessite de teclado, mouse ou monitor para poder ser acessado. Para conseguir acessar o Raspberry Pi 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[3]. 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 Raspberry Pi foi realizada para conter um IP estático (*192.168.2.60*), que facilita o acesso SSH ao sistema.

O Código 3.1 mostra um exemplo do arquivo *wpa_supplicant.conf*, que faz parte da configuração *headless*, servindo para conectar à rede automaticamente ao iniciar os Raspberry Pi. Já no Código 3.2, podemos observar a configuração do ip estático.

```
1 update_config=1
2 ctrl_interface=\var\run\wpa_supplicant
3
4 network={
5     ssid="<Nome da sua rede Wi-Fi>"
6     psk="<Senha da sua rede Wi-Fi>"
7 }
```

Código 3.1 – Exemplo de configuração *headless*

[1] https://www.home-assistant.io/docs/installation/hassbian/installation/
[2] https://www.raspberrypi.org/documentation/configuration/wireless/headless.md
[3] https://www.ssh.com/ssh/

```
update_config=1
interface <Sua interface de rede>
static ip_adress=192.168.2.60
static routers=192.168.2.1
static domain_name_servers=192.168.2.1
```

Código 3.2 – Exemplo de configuração do IP estático

3.3 Criação das tabelas no PostgreSQL

Todos os dados de um banco de dados relacional são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Em um banco de dados podem existir uma ou centenas de tabelas, sendo que o limite pode ser imposto tanto pela ferramenta de *software* utilizada, quanto pelos recursos de hardware disponíveis no equipamento.

As tabelas associam-se entre si por meio de regras de relacionamentos, que consistem em associar um ou vários atributos de uma tabela com um ou vários atributos de outra tabela.

A Figura 12 mostra o diagrama Entidade Relacionamento do banco de dados *users* criado para a aplicação. Um diagrama Entidade Relacionamento é capaz de descrever um modelo de tabelas (entidades) que são ligadas umas as outras por relacionamentos que expressam as dependências e exigências entre si. No caso do sistema planejado, existem três entidades: Usuário, Configuração e Banho. O usuário se relaciona com as outras duas entidades: ele possui uma configuração e pode tomar um banho.

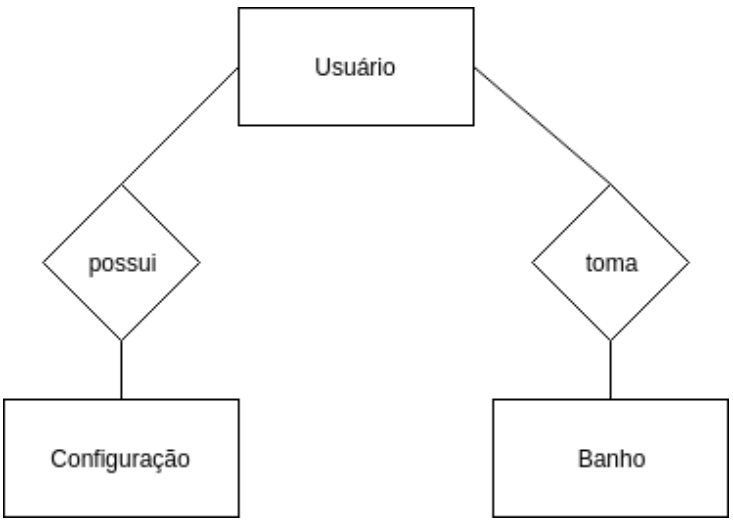

Figura 12 – Modelo Entidade Relacionamento do PostgreSQL

Tabela 2: Colunas da tabela *users*.

Coluna	Função
id	identificador único que se auto incrementa
name	nome do usuário cadastrado
password	senha criptografada do usuário cadastrado
created_at	data e hora da criação do usuário
updated_at	data e hora da última atualização no usuário

Tabela 3: Colunas da tabela *user_settings*.

Coluna	Função
id	identificador único que se auto incrementa
user_id	referência ao id do usuário
allowed_bath_time	tempo máximo de banho permitido
created_at	data e hora da criação do usuário
updated_at	data e hora da última atualização no usuário

Tabela 4: Colunas da tabela *user_baths*.

Coluna	Função
id	identificador único que se auto incrementa
user_id	referência 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 última atualização no usuário

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 mostradas na Tabela 2.

A Tabela 3 é a implementação da *user_settings*, que guarda informações de configuração dos usuários, e também contém 5 colunas.

A tabela *user_baths* guarda informações de tempo de todos os banhos tomados, possuindo 5 colunas, mostradas na Tabela 4.

A Figura 13 ilustra o banco de dados criado juntamente com as tabelas contidas nele.

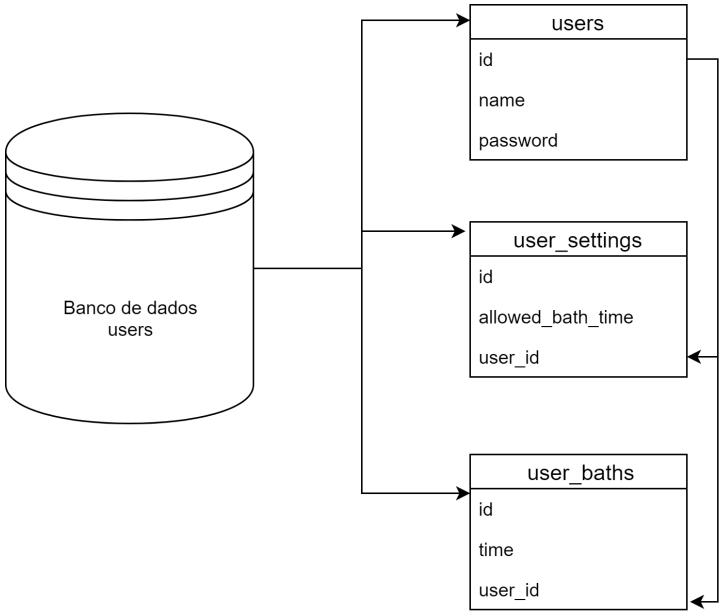

Figura 13 – Modelo das tabelas criadas no banco de dados

3.4 Configuração do InfluxDB

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

Foi criado o banco de dados *water_flow_data*, que guarda 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. Nos *fields* são guardados os dados que chegam diretamente do sensor, ou seja, o fluxo de água naquele determinado instante.

O InfluxDB salva automaticamente os *timestamps*[5] de cada dado incluido nele, o que facilita a manipulação dos dados, pois não é necessário informar a data e a hora toda vez que for incluir algum dado no banco.

No Código 3.3 está um exemplo de configuração e implementação das funções do InfluxDB no sistema.

[4] `https://www.npmjs.com/package/influx`

[5] *Timestamps* são marcas temporais, informa a data e a hora que um certo evento ocorreu.

```
const Influx = require('influx');

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

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

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

Código 3.3 – Exemplo do código de configuração do InfluxDB

3.5 Microsserviços

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

3.5.1 *Clean Architecture*

Os microsserviços desenvolvidos foram projetados para seguir a *Clean Architecture*. Esta arquitetura consiste em dividir as responsabilidades dentro de uma aplicação, encap-

sulando e abstraindo o código para facilitar a leitura e entendimento das devidas funções de cada arquivo (MARTIN, 2000).

É possível ver a divisão dos arquivos utilizando a *Clean Architecture* do Sistema de Usuários na Figura 14. Explicando esta divisão, na pasta routes encontram-se as configuração das rotas que são utilizadas no sistema, nos *controllers*, são feitas o tratamento dos *inputs* e respostas das rotas, redirecionando para os interactors, que é onde está a lógica principal da aplicação. Nos *repositories* é onde acontece a interação com o banco de dados, neste caso, com o PostgreSQL.

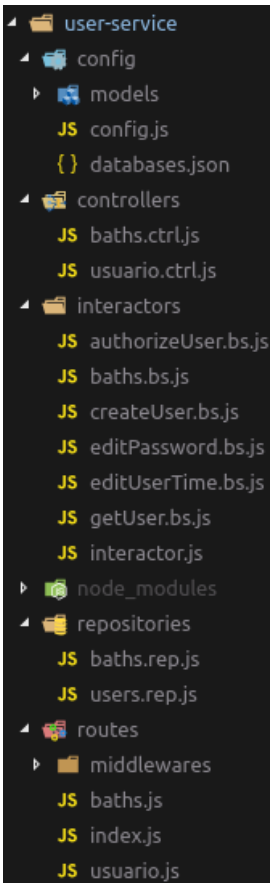

Figura 14 – Divisão dos arquivos do Sistema de Usuários

3.5.2 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 a tomar o banho.

As operações são realizadas através dos *endpoints* (Tabela 5), que é uma forma de comunicação entre os sistemas. O sistema os disponibiliza como endereços para acesso,

Tabela 5: *Endpoints* do sistema de usuários.

Endpoint	Característica
/cadastrar	possibilita cadastrar o usuário com as informações do nome, senha e tempo de banho permitido
/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

Tabela 6: Tópicos do *broker* MQTT.

Tópico	Função
keys	contém a tecla digitada no teclado numérico
actuator	envia informações do atuador, para liga-lo ou desliga-lo
user	envia informações sobre o usuário, como o nome e se ele está autorizado ou não
sensor	envia informações do sensor de fluxo

que podem receber e enviar informações, dependendo do que foi programado.

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

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

3.5.3 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 a autorização do usuário, 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 tópicos do *broker* MQTT descritos na Tabela 6 para receber e enviar as informações.

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

3.5.4 Sistema de simulação de dados

O sistema de simulação de dados emula todos os dados que podem ser capturados e enviados ao sistema físico, que são:

- Dados do fluxo de água: informação recebida pelo sensor YF-S201 (Seção 2.2.1);
- Dados do teclado numérico: informação de qual tecla foi pressionada (Seção 2.2.2);
- Informações para o atuador: informação para ligar/desligar o atuador (Seção 2.2.3);

Este sistema se inscreve nos tópicos do servidor MQTT e envia os dados simulados como se fosse o próprio sistema físico. Ao se iniciar o sistema, ele fica em estado de espera até alguma tecla for pressionada. Ao se pressionar uma tecla, ele a envia para o tópico *keys*. O Sistema de Comunicação, que se inscreve nesse tópico, reconhece a tecla e toma as devidas ações, como requisitar a senha ou ligar/desligar o atuador.

Os dados do fluxo de água são número inteiros que são passados para o tópico *sensor* do servidor MQTT, para simular estes dados envia-se um valor inteiro ao tópico em um intervalo pre-determinado de tempo (50 ms).

A Figura 15 mostra o sistema de simulação funcionando, com o usuário com o id 3 sendo autorizado e o sinal de ligar/desligar o atuador enviado.

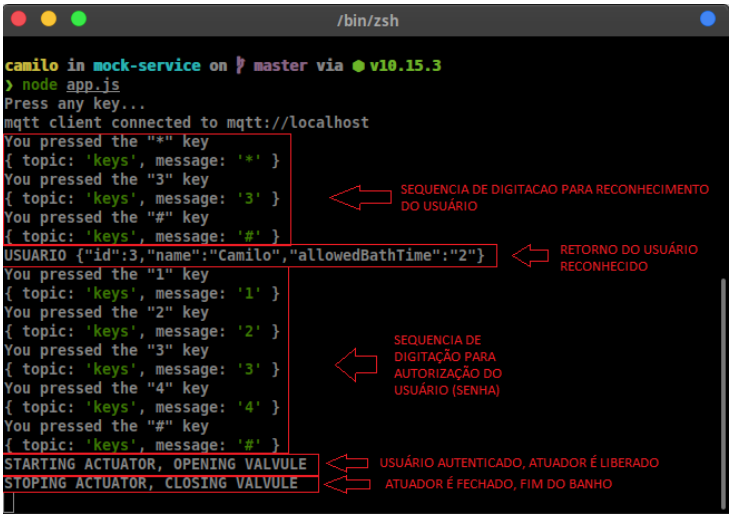

Figura 15 – Exemplo do sistema de simulação

3.6 Configuração do HomeAssistant

O HomeAssistant pode ser instalado a partir de diversos métodos[6], neste sistema, a instalação foi feita utilizando o Hassbian, que é um sistema operacional linux para o Raspberry Pi com o HomeAssistant pré-instalado.

Para configurar o HomeAssistant no Raspberry Pi é necessário realizar o download da imagem do Hassbian. Com o download concluído, deve-se copiar a imagem para o cartão SD a ser inserido no Raspberry Pi. Para isto, utilizamos o *balenaEtcher*.

Com a imagem devidamente escrita no cartão SD, basta inseri-lo no Raspberry Pi, configurar a rede como descrito na Seção 3.2, e, ao iniciar o sistema, o HomeAssistant será carregado automaticamente.

O HomeAssistant utiliza a porta 8123 e, para conseguir visualizar sua interface, o IP do servidor deve ser acessado por esta porta. No caso deste sistema, o IP do Raspberry Pi foi colocado como estático *192.168.2.60*. Então, para acessar a interface deve-se estar conectado na mesma rede do Raspberry Pi e acessar o link *http://192.168.2.60:8123*, como podemos ver na Figura 16.

Figura 16 – Página inicial do HomeAssistant acessado por um cliente conectado na mesma rede local do servidor Hassbian.

3.6.1 Sensores

A interface do HomeAssistant permite diversas configurações e inclusão de diferentes sensores, como mostra a Figura 8. No sistema desenvolvido neste trabalho, foi utilizado o sensor binário, que mostra o estado do chuveiro como ligado ou desligado. Cada usuário terá um sensor próprio, e seu estado será mostrado na interface.

[6] `https://www.home-assistant.io/docs/installation`

3.7 Configuração do Grafana

A instalação do Grafana foi realizada através do download[7] e extração do pacote para o sistema operacional Debian.

Com o pacote devidamente instalado no sistema, o Grafana utiliza a porta 3003 como interface. Para acessá-la, deve-se conectar à porta 3003 do Raspberry Pi, ou seja, acessar o link *http://192.168.2.60:3003*, como na Figura 17.

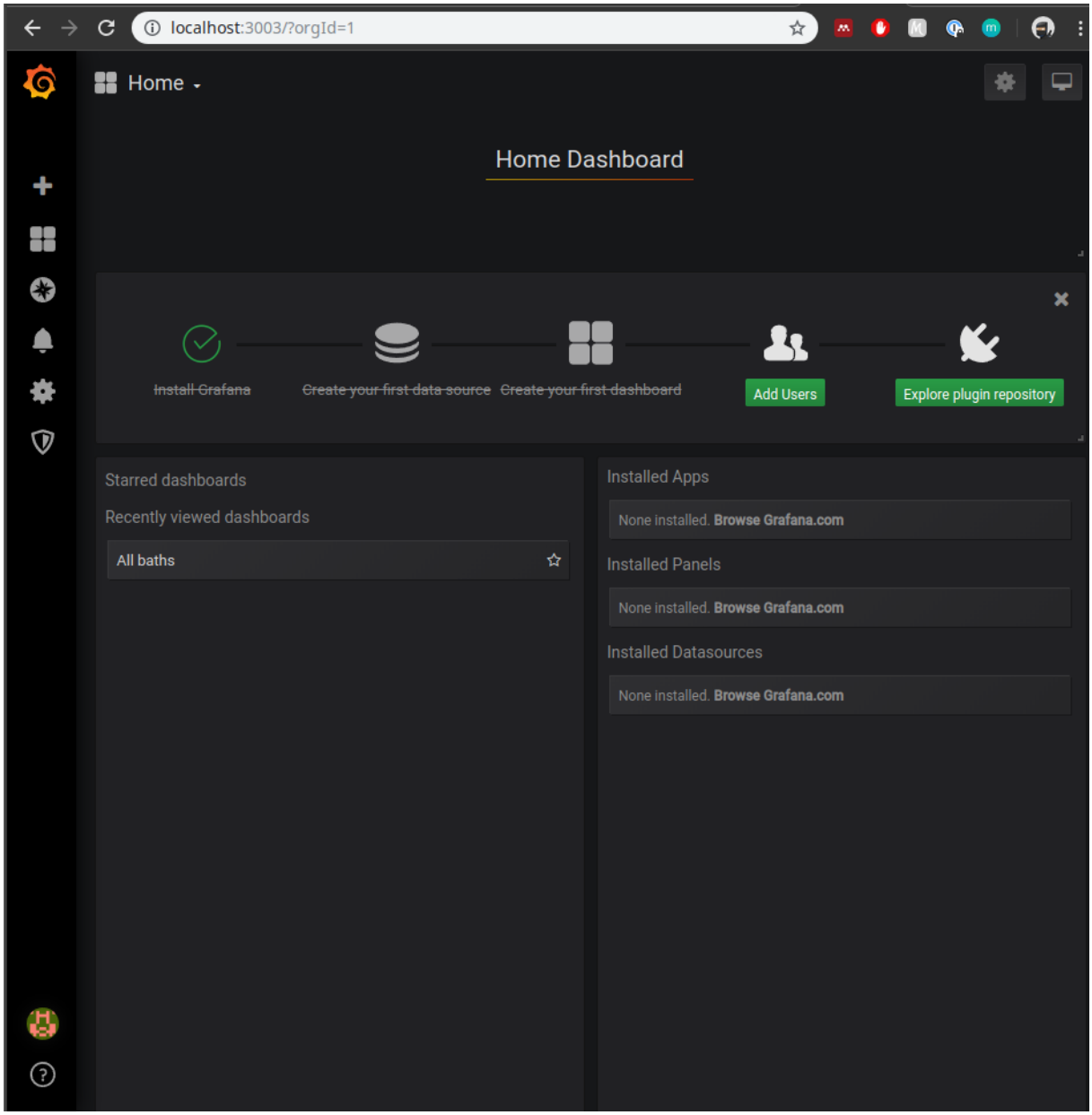

Figura 17 – Página inicial do Grafana.

[7] `https://grafana.com/get`

Ao acessar a interface do Grafana, é necessário criar um *dashboard*[8] 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 18.

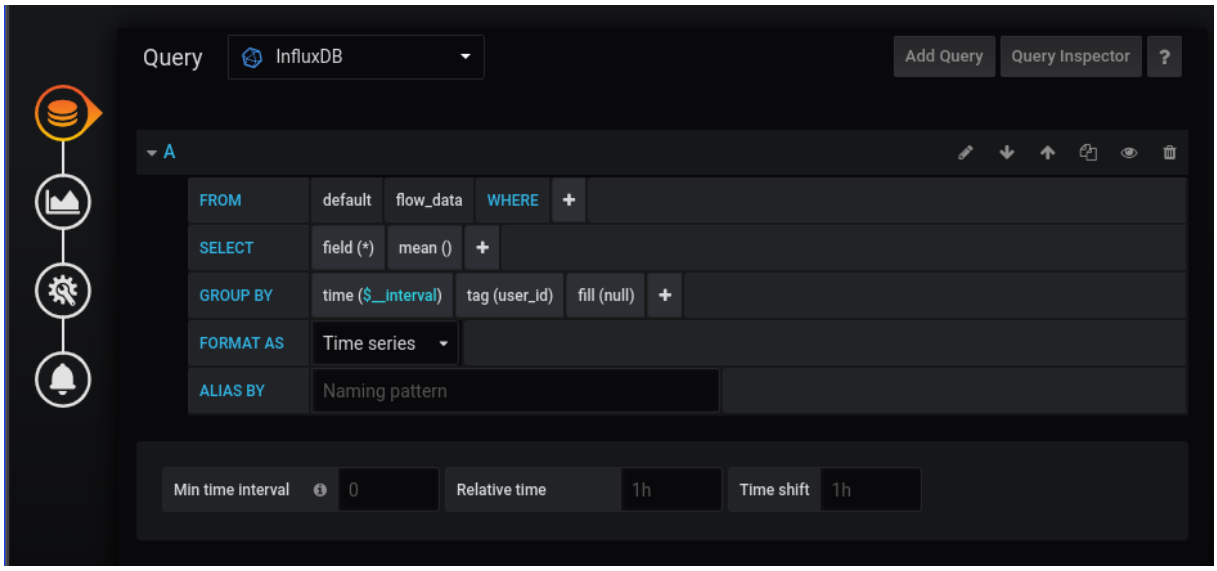

Figura 18 – Configuração do gráfico no Grafana.

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.

[8] *Dashboards* são painéis que mostram métricas e indicadores importantes para alcançar objetivos e metas traçadas de forma visual, facilitando a compreensão das informações geradas.

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.

4.1 Manipulação de usuários

O microsserviço de usuários utiliza a porta 3001. Para utiliza-lo e testá-lo, devemos utilizar esta porta juntamente com os *endpoints* disponíveis (Seção 3.5.2). Aqui realizaremos as consultas em todos estes *endpoints* e discutiremos os resultados obtidos.

Para criar um usuário devemos consumir[1] o *endpoint* http://localhost:3001/usuario/cadastrar passando os parâmetros via *POST* no formato *JSON*[2], como na Figura 19.

A resposta deste *endpoint* pode ser conferida na Figura 19 e, para garantir que o usuário foi incluído no banco de dados, podemos observar a Figura 19.

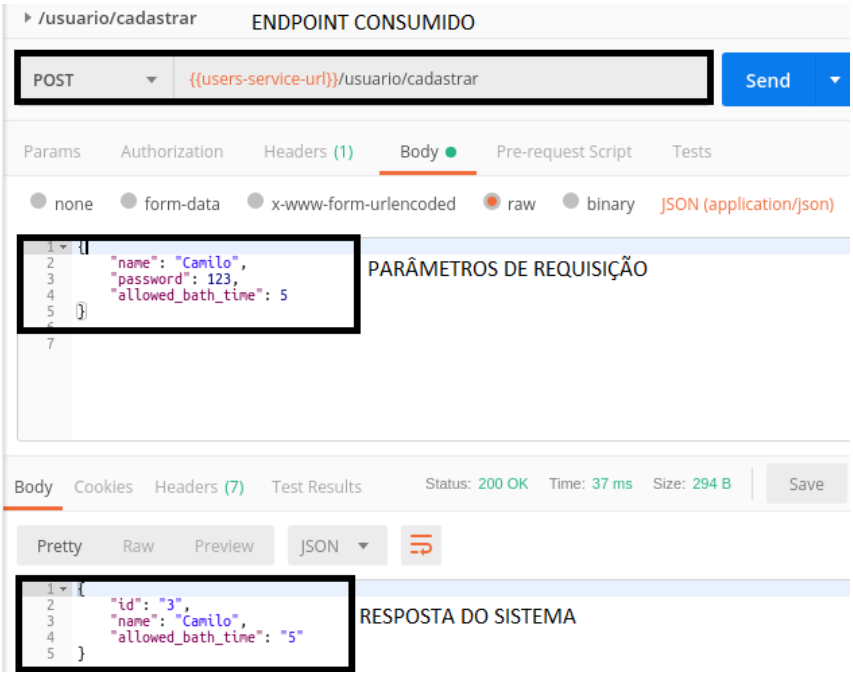

Figura 19 – Cadastro de usuários.

[1] O termo "consumir um *endpoint*" significa enviar informações para o *endpoint*, esperando algum retorno, de sucesso ou falha.

[2] Javascript Object Notation (JSON), é um formato compacto de troca de dados simples e rápida entre sistema

O *endpoint* http://localhost:3001/usuario/editar-tempo deve ser consumido para editar o tempo máximo de banho de um usuário (Figura 20). Para editar a senha do usuário, deve-se consumir o *endpoint* http://localhost:3001/usuario/editar-senha (Figura 21).

Figura 20 – Edição de tempo permitido do usuário.

Figura 21 – 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 22.

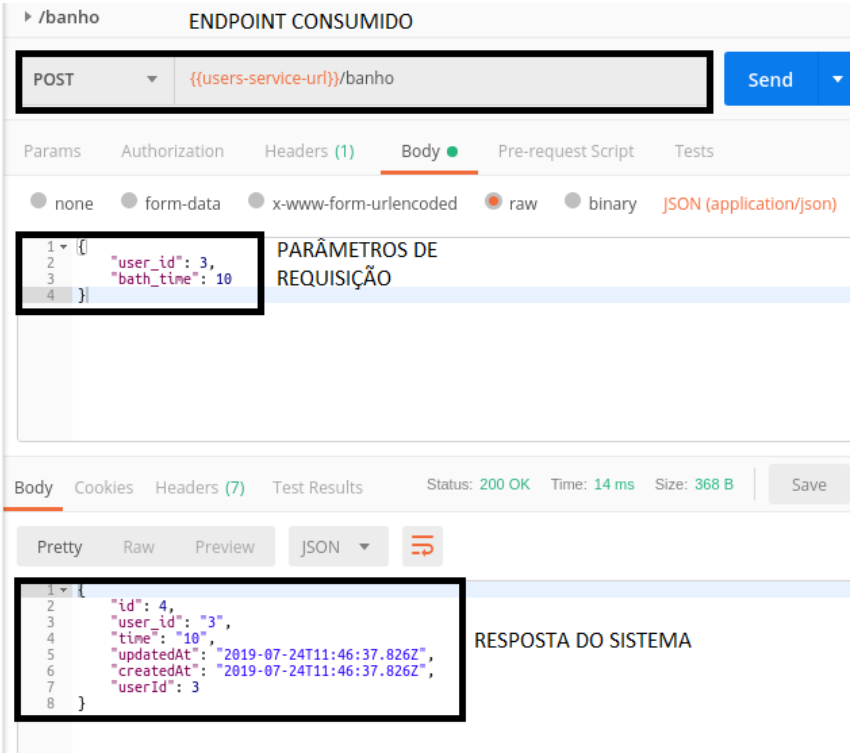

Figura 22 – Adicionando banhos ao usuário.

Para recuperar as informações de um usuário, basta consumir o *endpoint* http://localhost:3001/usuario/idDoUsuario, com o método *GET*, obtendo o resultado da Figura 23.

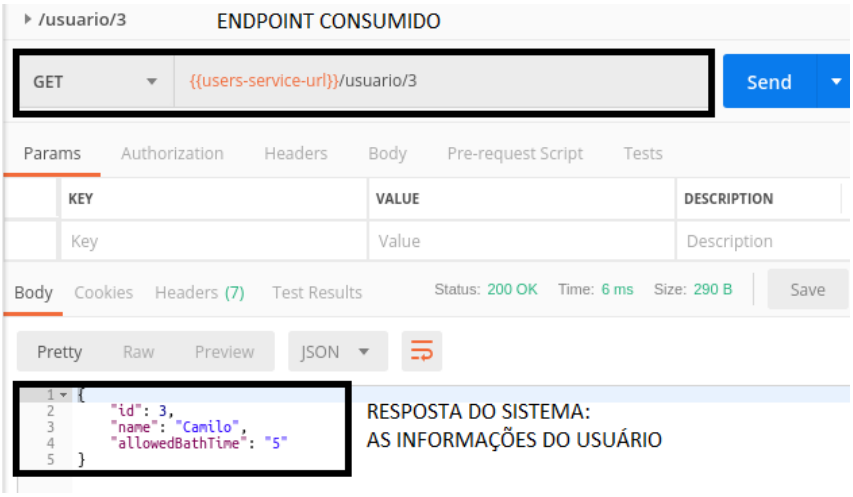

Figura 23 – Recuperar dados do usuário.

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 24.

Figura 24 – Recuperar dados de banhos do usuário.

Finalmente, podemos autenticar os usuários enviando via *POST* os parâmetros para o *endpoint* http://localhost:3001/usuario/autorizar, obtendo como resposta a Figura 26, para usuário autenticado, e Figura 25, para usuário não autenticado, caso a senha esteja errada.

Figura 25 – Exemplo de usuário não autorizado.

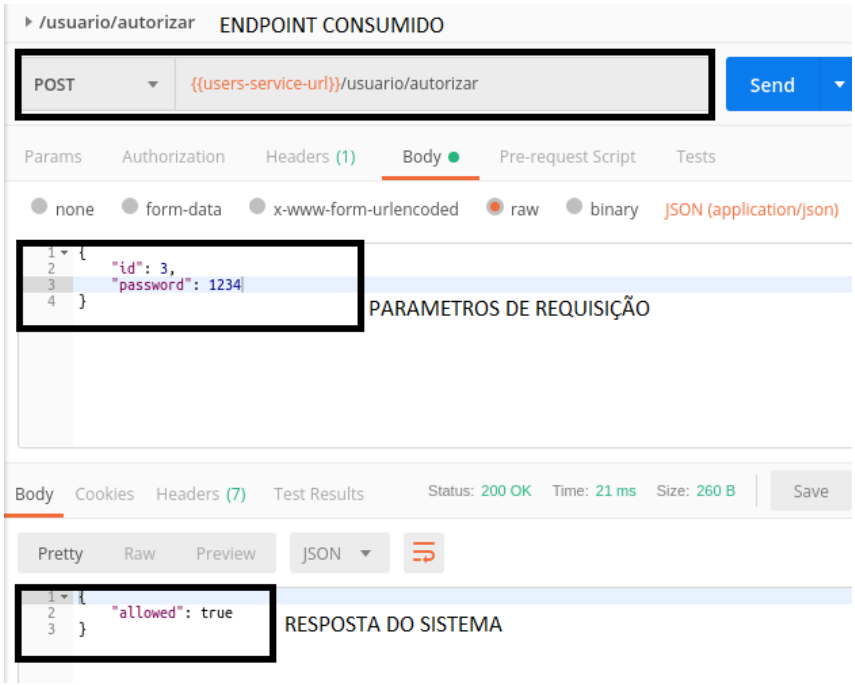

Figura 26 – Exemplo de usuário autorizado.

4.2 Visualização via Grafana

Os dados para a visualização no Grafana foram gerados pelo Sistema de simulação de dados (Seção 3.5.4).

Ao autenticar um usuário, um sinal é enviado ao tópico do atuador (Seção 2.2.3) e o sinal emulado do sensor YF-S201 (Seção 2.2.1) inicia a simulação do fluxo, que é automaticamente enviado para o InfluxDB via Sistema de comunicação MQTT (Seção 3.5.3), podendo ser visualizado no Grafana. Os gráficos do grafana podem ser acessados via http://localhost:3003, como na Figura 17.

Pode ser observado na Figura 27 um gráfico do fluxo simulado pelo sensor no horário de 16h40m até 17h00m. Existem dois gráficos com cores diferentes, cada cor é referente a um usuário.

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 29, que encontram-se em estado desligado. Ao digitar corretamente a senha, o estado do sensor muda para ligado, como observado na Figura 28.

Ao cadastrar um usuário, o seu sensor é inserido no HomeAssistant automaticamente (Figura 30).

Figura 27 – Exemplo do gráfico no Grafana para usuários diferentes.

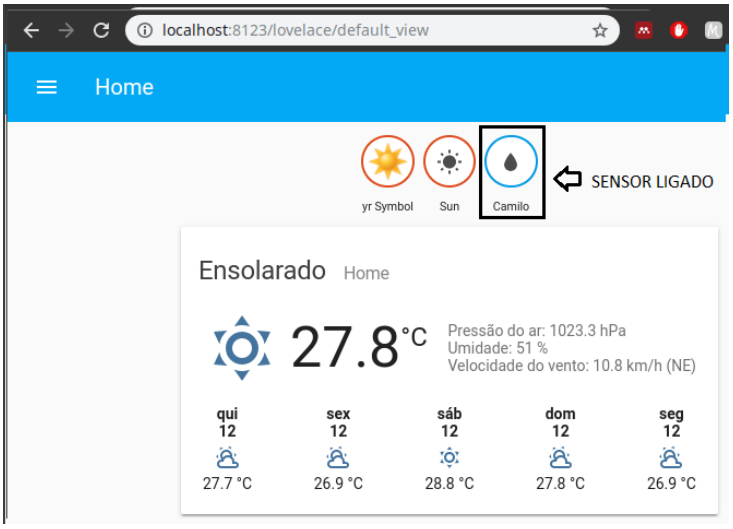

Figura 28 – Exemplo do sensor no HomeAssistant quando ligado.

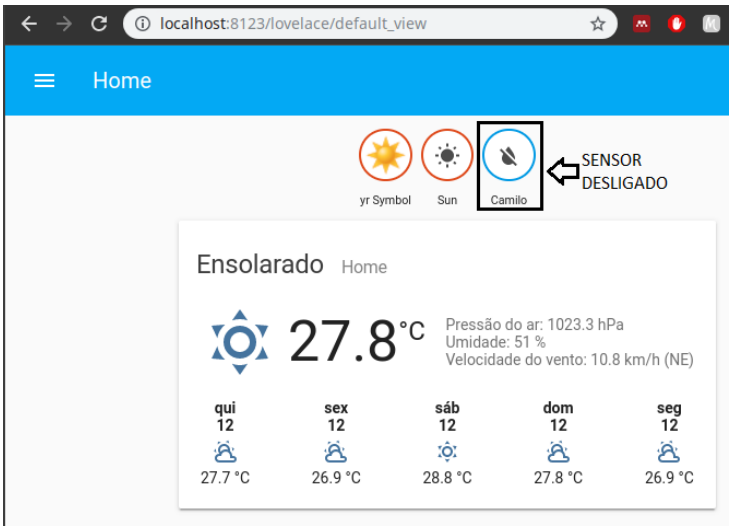

Figura 29 – Exemplo do sensor no HomeAssistant quando desligado.

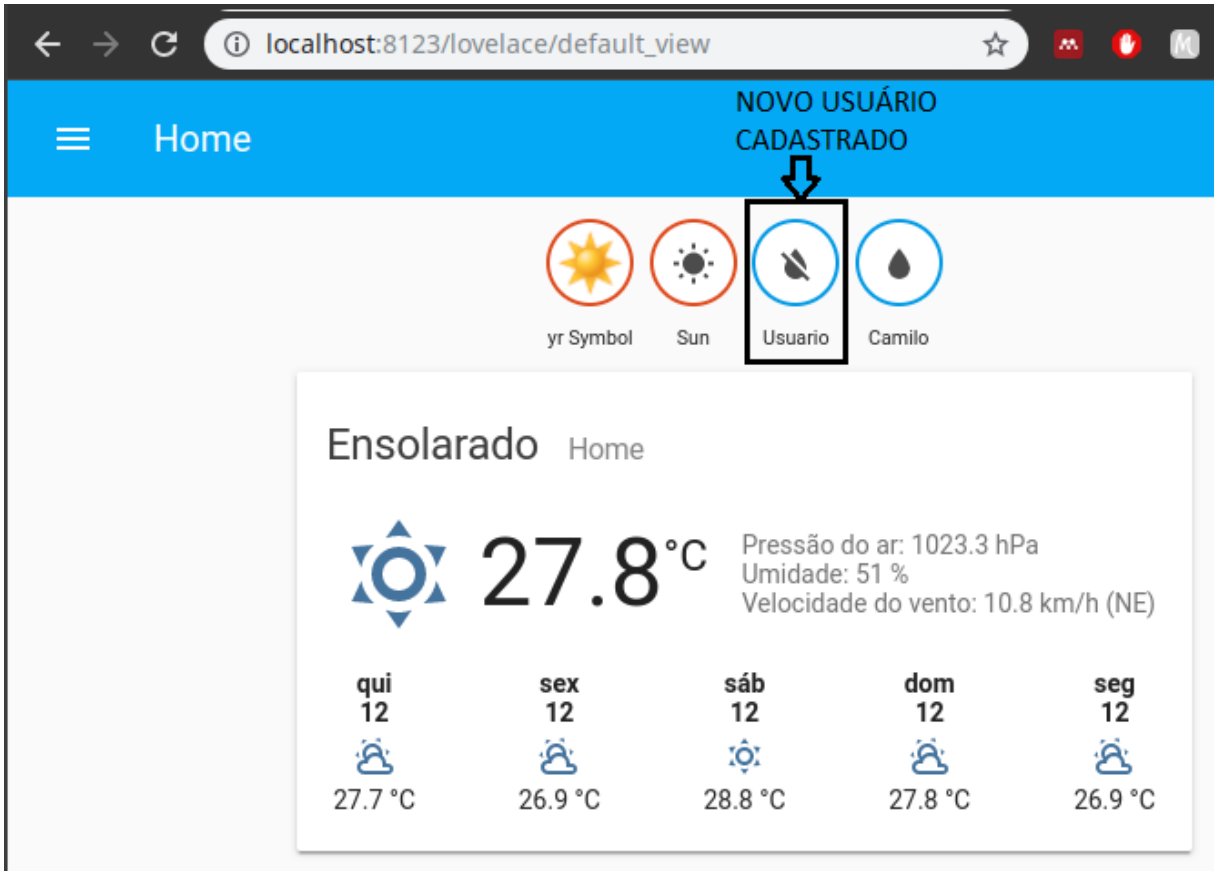

Figura 30 – Exemplo de um novo sensor quando um novo usuário é cadastrado.

5 Considerações Finais

Este trabalho propõe um sistema capaz de monitorar remotamente a utilização da água em uma residência. Os sistemas de monitoramento propostos foram implementados. Como mostrado, a implementação é capaz de cadastrar, editar e autorizar usuários. Monitorar o fluxo de água e o tempo de banhos com gráficos e estados de sensores em tempo real remotamente, comportando-se como deveria.

Ao final do desenvolvimento do projeto e dos resultados apresentados, pode-se concluir que o sistema é eficaz no monitoramento remoto auxiliando o usuário na gestão da utilização de água em chuveiros. Podendo ser um grande aliado na redução do consumo de água em toda a residência, se implementado em outros setores, como torneiras ou tanques.

Para trabalhos futuros, pode-se realizar 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 mais sistemas de monitoramento utilizando outros sensores realizando a integração com HomeAssistant implementando 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 20 e 21.

BOSCARIOLI, C. et al. *Uma reflexão sobre Banco de Dados Orientados a Objetos*. [S.l.], 2006. Citado na página 22.

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 na página 22.

CHASE, O.; ALMEIDA, F. Sistemas embarcados. *Mídia Eletrônica. Página na internet:< www. sbajovem. org/chase>, capturado em*, 2007. v. 10, n. 11, p. 13, 2007. Citado na página 15.

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 16.

DIÁRIAS, A. et al. ANÁLISE DE CONSUMO E DESPERDÍCIO DE ÁGUA EM. 2007. v. 3, p. 0–5, 2007. Citado na página 12.

FERREIRA, A. N. et al. Água subterrâneas - um recurso a ser conhecido e protegido. 2007. 2007. Citado na página 12.

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 12.

FORTY, A. *Objetos de desejo*. [S.l.]: Editora Cosac Naify, 2007. Citado na página 13.

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 21.

JÚNIOR, J. M. M. L.; ARÊAS, 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 2 vezes nas páginas 24 e 25.

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 25.

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 2 vezes nas páginas 20 e 21.

MARTIN, R. *Clean Architecture: A Craftsman's Guide to Software Structure and Design/Robert Martin*. [S.l.]: New York: Prentice Hall, 2000. Citado na página 33.

MARTINS, N. A. Sistemas microcontrolados. *Uma abordagem com o Microcontrolador PIC 16F84. Editora Novatec Ltda, 1ª edição*, 2005. 2005. Citado na página 15.

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 23.

NOOR, S. et al. *Time Series Databases and InfluxDB*. [S.l.], 2017. Citado 2 vezes nas páginas 21 e 22.

OLIVEIRA, E. *Blog Masterwalker*. 2018. `http://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-teclado-matricial-de-membrana-4x4/`. Accessed: 2019-06-02. Citado na página 17.

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 12 e 13.

PICORETI, R. *Portal Vida de Silício*. 2017. `https://portal.vidadesilicio.com.br/teclado-matricial-e-multiplexacao/`. Accessed: 2019-06-02. Citado 2 vezes nas páginas 17 e 18.

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 12.

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 19.

STONES, R.; MATTHEW, N. *Beginning databases with PostgreSQL: from novice to professional*. [S.l.]: Apress, 2006. Citado na página 22.

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 13.

Apêndices

APÊNDICE A – Código sistema de usuários

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

```
1  import Interactor from './interactor';
2  import usersRep from '../repositories/users.rep';
3  import bcrypt from 'bcrypt';
4  import config from './../config/config';
5
6  class CreateUserBs extends Interactor {
7    constructor(){
8      super();
9    }
10
11   async execute({ name, password, allowedBathTime }) {
12
13     if (!allowedBathTime) {
14       allowedBathTime = 10;
15     }
16
17     const hash = await bcrypt.hash(password.toString(), config.bcrypt.
         saltRounds);
18
19     const user = await usersRep.createUser({ name, password: hash,
         allowedBathTime });
20
21     if(!user) {
22       throw 'Nao foi possivel criar o usuario!';
23     }
24
25     const response = {
26       id: user.user_id,
27       name: user.user.name,
28       allowed_bath_time: user.allowed_bath_time,
29     };
30
31     return response;
32   }
```

```
33 }
34
35 export default CreateUserBs;
```

Código A.1 – Exemplo do código do *interactor* de criação do usuário

```
1  import Interactor from './interactor';
2  import usersRep from '../repositories/users.rep';
3  import bcrypt from 'bcrypt';
4  import config from './../config/config';
5
6  class EditPasswordBs extends Interactor {
7    constructor(){
8      super();
9    }
10
11   async execute({ id, new_password }) {
12
13     const hash = await bcrypt.hash(new_password.toString(), config.
         bcrypt.saltRounds);
14
15     const editedUser = await usersRep.editPassword({ id, password: hash
         });
16
17     const response = editedUser[0] === 1 ? {
18       status: 200,
19       message: 'Usuario editado com sucesso'
20     } :
21     {
22       status: 500,
23       message: 'Erro ao editar usuario'
24     };
25
26     return response;
27   }
28 }
```

Código A.2 – Exemplo do código do *interactor* de edição de senha de usuários

```
import Interactor from './interactor';
import usersRep from '../repositories/users.rep';

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: 'Usuario editado com sucesso'
    } :
    {
      status: 500,
      message: 'Erro ao editar usuario'
    };

    return response;

  }
}
```

Código A.3 – Exemplo do código do *interactor* de edição de tempo de banho dos usuários

```
1  import Interactor from './interactor';
2  import usersRep from '../repositories/users.rep';
3  import bcrypt from 'bcrypt';
4
5  class AuthorizeUserBs extends Interactor {
6    constructor(){
7      super();
8    }
9
10   async execute({ id, pass }) {
11     const user = await usersRep.getUser({ id });
12
13     if(!user) {
14       throw {error: 'Usuario nao encontrado'};
15     }
16
17     const match = await bcrypt.compare(pass.toString(), user.password);
18
19     return {
20       allowed: match
21     };
22   }
23 }
```

Código A.4 – Exemplo do código do *interactor* de autorização de usuários

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

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

  async execute({ user_id, bath_time }) {

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

    if(!bath) {
      throw 'Nao foi possivel cadastrar o banho';
    }

    return bath;
    }

    async getBaths ({ user_id }) {

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

    return baths;
  }
}

export default BathsBs;
```

Código A.5 – Exemplo do código do *interactor* de cadastro de banhos

```
1  import Interactor from './interactor';
2  import usersRep from '../repositories/users.rep';
3
4  class GetUserBs extends Interactor{
5    constructor(){
6      super();
7    }
8
9    async execute({ id }) {
10   const user = await usersRep.getUser({ id });
11
12   if(!user) {
13     throw {error: 'Nao foi possivel localizar o usuario!'};
14   }
15
16   const response = {
17     id: user.id,
18     name: user.name,
19     allowedBathTime: user.user_setting.allowed_bath_time
20   };
21
22   return response;
23   }
24 }
```

Código A.6 – Exemplo do código do *interactor* para recuperar informações dos usuários

APÊNDICE B – Código sistema de comunicação MQTT

Neste apêndice encontram-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`

```
1  const mqtt = require('mqtt');
2  const EventEmitter = require('events');
3
4
5  class MqttHandler extends EventEmitter {
6    constructor() {
7      super();
8      this.mqttClient = null;
9      this.host = 'mqtt://mosquitto';
10     this.username = 'mqtt_user'; // mqtt credentials if these are needed
           to connect
11     this.password = '100200300';
12   }
13
14
15   connect() {
16     const self = this;
17     this.mqttClient = mqtt.connect(this.host, { username: this.username,
           password: this.password });
18
19     this.mqttClient.on('error', (err) => {
20       console.log(err);
21       this.mqttClient.end();
22     });
23
24     this.mqttClient.on('connect', () => {
25       console.log(`mqtt client connected to ${this.host}`);
26     });
27
28     this.mqttClient.on('close', () => {
29       console.log(`mqtt client disconnected`);
30     });
31
32     this.mqttClient.on('message', (topic, message) => {
```

```
33        const builtMessage = {
34          topic,
35          message: message.toString()
36        };
37
38        self.emit('messageReceived', builtMessage);
39      })
40    }
41
42    subscribe(topic) {
43      this.mqttClient.subscribe(topic, {qos: 0});
44    }
45
46    sendMessage(message, topic) {
47      this.mqttClient.publish(topic, message);
48    }
49  }
50
51  module.exports = MqttHandler;
```

Código B.1 – Exemplo do código de comunicação MQTT

```
1   const requestService = require('./requestService');
2
3   class timeHandler {
4     constructor(mqttClient) {
5       this.mqttClient = mqttClient;
6       this.allowedTime;
7       this.interval;
8       this.nowDate;
9       this.endDate;
10      this.userId;
11    }
12
13    countTime(allowedTime, id) {
14      this.userId = id;
15      this.allowedTime = allowedTime;
16      this.nowDate = new Date();
17
18      this.interval = setInterval(() => {
19      this.mqttClient.sendMessage('stop', 'actuator');
20      this.clearBathInterval();
21      }, this.allowedTime);
22    }
23
24    async endTime() {
25      this.clearBathInterval();
26      this.endDate = new Date();
```

```
27
28      const bathTime = this.endDate - this.nowDate;
29
30      console.log('BATHTIME>>>', bathTime);
31      await this._requestUserService(bathTime);
32    }
33
34    clearBathInterval(){
35      clearInterval(this.interval);
36    }
37
38    async _requestUserService(time) {
39      const requestOptions = {
40        type: 'POST',
41        endpoint: 'banho',
42        body: {
43        user_id: this.userId,
44        bath_time: time
45        },
46      };
47
48      return await requestService.userRequest(requestOptions);
49    }
50  }
51
52  module.exports = timeHandler;
```

Código B.2 – Exemplo do código responsável por lidar com informações do tempo de banho

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

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

  async _requestUserService() {
    const requestOptions = this.gettingPassword ? {
      type: 'POST',
      endpoint: `usuario/autorizar`,
      body:{
        id: this.userId,
        password: this.keyBuffer
```

```
46        },
47        }: {
48          type: 'GET',
49          endpoint: `usuario/${this.keyBuffer}`,
50        body: null,
51      };
52
53      return requestService.userRequest(requestOptions)
54      .then((response) => {
55        this.userId = this.gettingPassword ? this.userId : response.id;
56        return response
57      }).catch(err => {throw err});
58    }
59  }
60
61  module.exports = KeysHandler;
```

Código B.3 – Exemplo do código que lida com a comunicação com o InfluxDB

```
1  const mqttHandler = require('./mqtt_handler');
2  const KeysTopicHandler = require('./keysTopicHandler');
3  const TimeHandler = require('./timeHandler');
4  const SUBSCRIBE_TOPICS = ['keys', 'actuator', 'user', 'sensor'];
5  const influx = require('./influxHandler');
6  const mqttClient = new mqttHandler();
7  const keysHandler = new KeysTopicHandler(mqttClient);
8  const timeHandler = new TimeHandler(mqttClient);
9  const influxDB = new influx();
10 influxDB.connect();
11 mqttClient.connect();
12 SUBSCRIBE_TOPICS.forEach((topic) => {
13   mqttClient.subscribe(topic);
14 })
15
16 let usuario;
17 let sensorCount = 0;
18 let actuatorOn;
19 mqttClient.on('messageReceived', (received) => {
20   switch(received.topic) {
21     case 'keys':
22       if(!actuatorOn) {
23         keysHandler.handle(received.message);
24       }
25       break;
26     case 'actuator':
27       if(received.message === 'start') {
28         timeHandler.countTime(usuario.allowedBathTime * 60000, usuario.
              id) // usuario.allowedBathTime * 60000
```

```
29          mqttClient.sendMessage('ON', `homeassistant/binary_sensor/${
                usuario.name}/state`);
30          actuatorOn = true;
31        }
32
33        if(received.message === 'stop') {
34          timeHandler.endTime();
35          if(usuario) {
36            influxDB.getPoints(usuario.id)
37              .then(/*console.log*/);
38          }
39          sensorCount = 0;
40          actuatorOn = false;
41          mqttClient.sendMessage('OFF', `homeassistant/binary_sensor/${
                usuario.name}/state`);
42        }
43        break;
44      case 'sensor':
45        if(usuario) {
46          if(received.message < 10) sensorCount++;
47
48          if(sensorCount > 10) {
49            mqttClient.sendMessage('stop', 'actuator');
50            sensorCount = 0;
51          };
52
53          influxDB.writePoints(usuario.id, received.message)
54            .then(console.log)
55            .catch(console.log);
56        } else {
57          console.log('NO USER DETECTED, STOPPING ACTUATOR');
58          mqttClient.sendMessage('stop', 'actuator')
59        };
60        break;
61      case 'user':
62        const message = JSON.parse(received.message);
63        mqttClient.sendMessage(`{"name": "${message.name}", "device_class"
              : "moisture", "state_topic": "homeassistant/binary_sensor/${
              message.name}/state"}`, `homeassistant/binary_sensor/${message.
              name}/config`);
64        usuario = message;
65        break;
66    }
67  })
```

Código B.4 – Código principal do sistema