



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 e arquitetura de microsserviços: uma solução IoT.**

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, agosto 2019

Camilo Esteves Mendes de Avelar

**Domótica aplicada no controle de água utilizando comunicação
MQTT e arquitetura de microserviços: 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 2019

Camilo Esteves Mendes de Avelar

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

60 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, agosto 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 agosto 2019, pela comissão avaliadora constituída pelos professores:

Filipe Augusto Santos Rocha
Orientador

Convidado

Convidado

Ouro Preto, agosto 2019

Resumo

Este trabalho tem o propósito de criar um sistema para Domótica para controle de água utilizando um sensor de fluxo de água, uma válvula solenoide como atuador e um RaspberryPi. O sistema será capaz de monitorar o uso de água através do sensor de fluxo, 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 usuários para integrar no sistema. Toda a comunicação entre o sensor de fluxo e o RaspberryPi será feita via MQTT e 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.

Palavras-chaves: iot, monitoramento, microsserviços, mqtt, node, domótica, raspberry pi

Abstract

This paper has the porpose of create a Domotic system for water control using a water flux sensor, solenoid valve as actuator and a Raspberry Pi. The system will be able to monitorize the use of the water using the flux sensor, 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 communications between the sensor and the Raspberry Pi will be done via MQTT and the communication system and the users system will be built using the microservices architeture, being two different systems that communicate with each other to do the necessary operations. The users and communication 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

Listas de ilustrações

Figura 1	Arquitetura básica de microprocessadores e microcontroladores	15
Figura 2	ESP8266	16
Figura 3	RaspberryPi	16
Figura 4	Sensor de fluxo de água YF-S201	17
Figura 5	Teclado Matricial de Membrana	18
Figura 6	Circuito do teclado	18
Figura 7	Pinagem do teclado	19
Figura 8	Válvula Solenoide	19
Figura 9	Dashboard genérico do HomeAssistant	20
Figura 10	Exemplo arquitetura de microsserviços	23
Figura 11	Exemplo da arquitetura do MQTT	24
Figura 12	Exemplo de configuração <i>headless</i>	28
Figura 13	Exemplo de configuração do IP estático	28
Figura 14	Modelo Entidade Relacionamento do PostgreSQL	30
Figura 15	Divisão dos arquivos do Sistema de Usuários	32
Figura 16	Exemplo do sistema de simulação	34
Figura 17	Página inicial do HomeAssistant acessado por um cliente conectado na mesma rede local do servidor Hassbian	35
Figura 18	Página inicial do Grafana	36
Figura 19	Configuração do gráfico no Grafana	37
Figura 20	Cadastro de usuários	38
Figura 21	Edição de tempo permitido do usuário	39
Figura 22	Edição de senha do usuário	40
Figura 23	Adicionando banhos ao usuário	40
Figura 24	Recuperar dados do usuário	41
Figura 25	Recuperar dados de banhos do usuário	41
Figura 26	Exemplo de usuário não autorizado	42
Figura 27	Exemplo de usuário autorizado	42
Figura 28	Exemplo do gráfico no Grafana para usuários diferentes	43
Figura 29	Exemplo do sensor no HomeAssistant quando ligado	43
Figura 30	Exemplo do sensor no HomeAssistant quando desligado	44
Figura 31	Exemplo de um novo sensor quando um novo usuário é cadastrado	44

Lista de tabelas

Tabela 1	Tabela de disposição dos pinos do teclado numérico	18
----------	--	----

Lista de códigos

3.1	Exemplo do código de configuração do InfluxDB	30
A.1	Exemplo do código do <i>interactor</i> de criação do usuário	50
A.2	Exemplo do código do <i>interactor</i> de edição de senha de usuários	51
A.3	Exemplo do código do <i>interactor</i> de edição de tempo de banho dos usuários .	52
A.4	Exemplo do código do <i>interactor</i> de autorização de usuários	53
A.5	Exemplo do código do <i>interactor</i> de cadastro de banhos	54
A.6	Exemplo do código do <i>interactor</i> para recuperar informações dos usuários . .	55
B.1	Exemplo do código de comunicação MQTT	56
B.2	Exemplo do código responsável por lidar com informações do tempo de banho	57
B.3	Exemplo do 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

Lista de códigos	8
1 Introdução	12
1.1 Objetivos	13
1.1.1 Objetivos específicos	13
1.2 Justificativas e Relevância	14
1.3 Organização do trabalho	14
2 Materiais e Softwares	15
2.1 Microcontroladores e Microprocessadores	15
2.1.1 ESP8266	16
2.1.2 RaspberryPi	16
2.2 Sensores e atuadores	17
2.2.1 Sensor de fluxo YF-S201	17
2.2.2 Teclado matricial de membrana	17
2.2.3 Válvula solenoide	18
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	26
3.1.2 Dados gerados	27
3.1.3 Ligar/desligar atuador	27
3.2 Configuração do RaspberryPi	27
3.3 Criação das tabelas no PostgreSQL	28
3.4 Configuração do InfluxDB	30

3.5	Microsserviços	31
3.5.1	<i>Clean Architecture</i>	31
3.5.2	Sistema de usuários	32
3.5.3	Sistema de comunicação MQTT	33
3.5.4	Sistema de simulação dos dados	33
3.6	Configuração do HomeAssistant	34
3.6.1	Sensores	35
3.7	Configuração do Grafana	36
4	Experimentos e Resultados	38
4.1	Manipulação de usuários	38
4.2	Visualização via Grafana	43
4.3	Estado do atuador via HomeAssistant	43
5	Considerações Finais	45
	Referências	46
	Apêndices	49
	APÊNDICE A Código sistema de usuários	50
	APÊNDICE B Código sistema de comunicação MQTT	56

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 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. (FERREIRA et al., 2007)

Pesquisas indicam 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 paradigma do IoT, evidenciado no parágrafo a seguir. 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 (Internet of Things ou Internet das Coisas) representa uma rede mundial de objetos 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. 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 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 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)

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 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.

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 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, incluindo 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 combinar as técnicas de 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.

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. Com o advento dos microcontroladores de 16 e 32 bits - o padrão mais difundido atualmente é o de 8bits - a capacidade de prover soluções mais complexas e maior velocidade de processamento se iguala ao do microprocessador. (CHASE; ALMEIDA, 2007).

Na Figura 1, pode-se observar algumas das diferenças entre Microprocessadores e Microcontroladores.

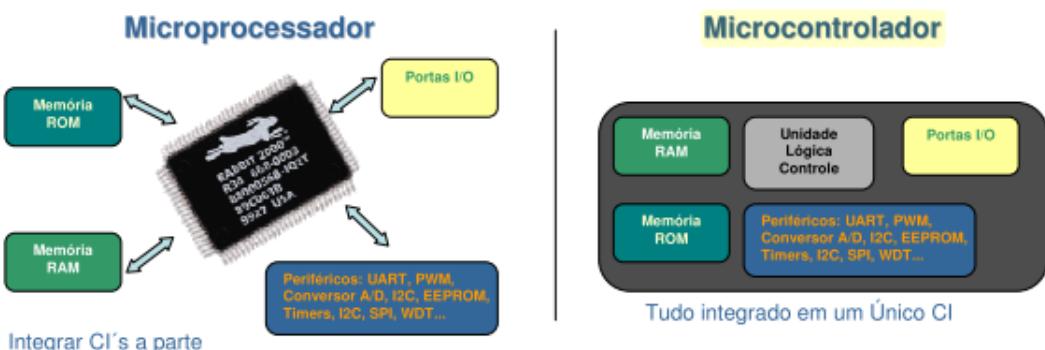


Figura 1 – Arquitetura básica de microprocessadores e microcontroladores

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)

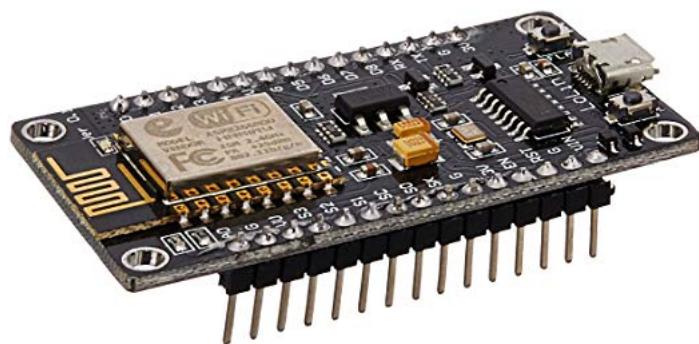


Figura 2 – ESP8266

Fonte: KODALI; SORATKAL (2017)

2.1.2 RaspberryPi

RaspberryPi (Figura 3) 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. 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

Nesta seção serão apresentados os sensores e atuadores a serem utilizados no projeto.

2.2.1 Sensor de fluxo YF-S201

O sensor YF-S201 (Figura 4) é um sensor do tipo turbina que mede 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 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 5), com 16 teclas 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.

As 16 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. O circuito do teclado está exemplificado na Figura 6.

A Tabela 1 possui as informações da distribuição dos pinos em tabelas e colunas, exemplificada na Figura 7.



Figura 5 – Teclado Matricial de Membrana

Fonte: OLIVEIRA (2018)

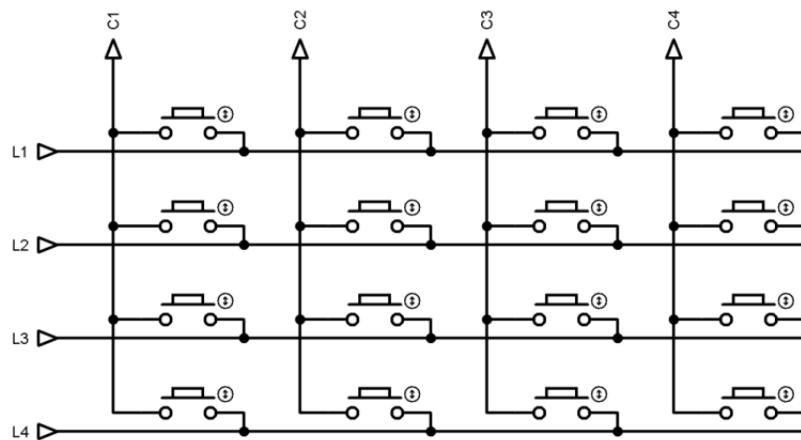


Figura 6 – Circuito do teclado

Fonte: PICORETI (2017)

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

Tabela 1 – Tabela de disposição dos pinos do teclado numérico

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,

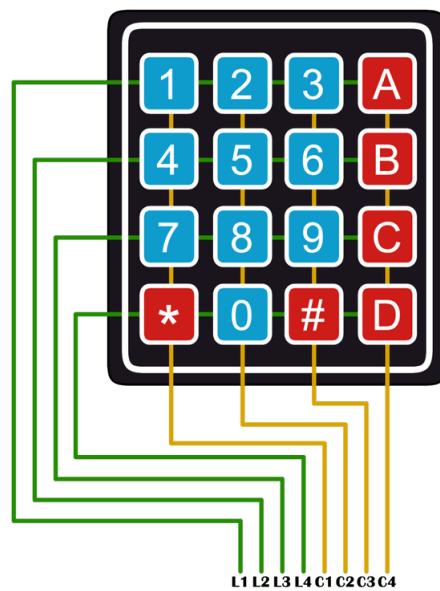


Figura 7 – Pinagem do teclado

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

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 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 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 (LUNDIGAN et al., 2017). Em seu núcleo, HomeAssistant é um protocolo de troca de mensagens que facilita 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 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, *switches* para acionamento de interruptores e iluminação.

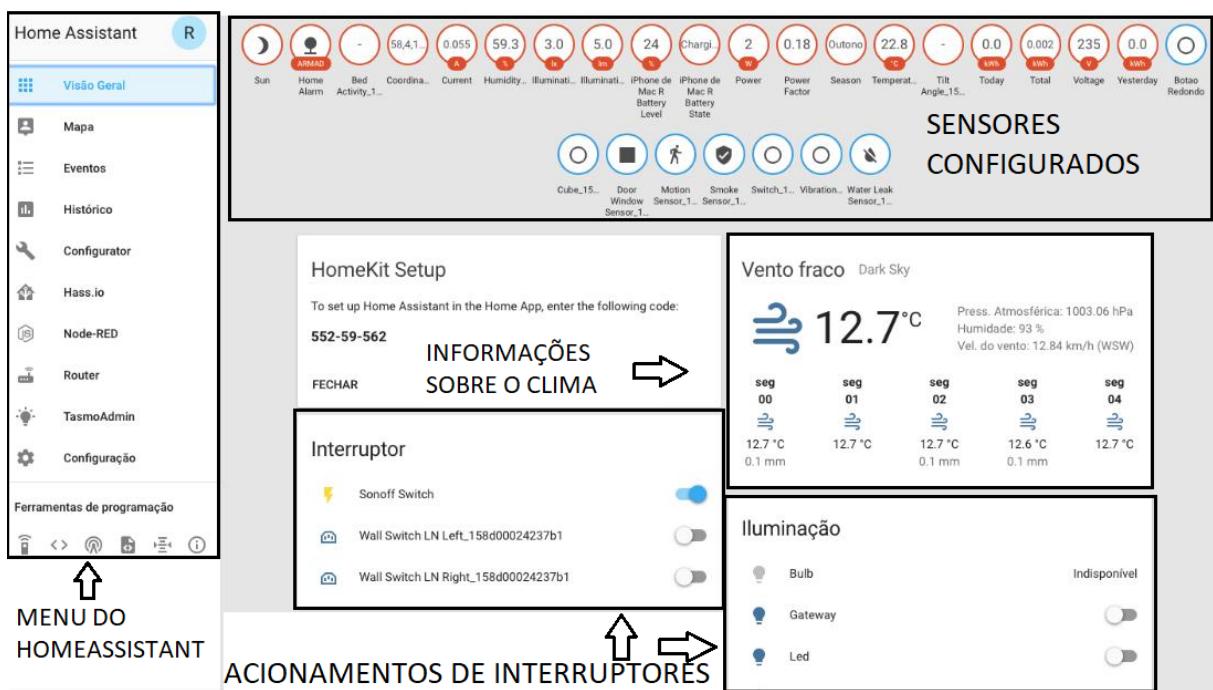


Figura 9 – Dashboard genérico do HomeAssistant

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 Home Assistant. (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 do que outros protocolos wireless.
- Sensores Wi-Fi tem integração com demais equipamentos residenciais.

O HomeAssistant pode ser instalado em qualquer sistema operacional, é muito pequeno e leve. Fato que o faz ser 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 controle que pode informar outros serviços, como o Philips Hue ou o Nest, 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 que contém traços de tempo ou séries temporais. É construído 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 um método mais eficiente. Atualmente, a maioria das empresas estão gerando uma quantidade gigante 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, Analise de Dados, etc. Alguns casos de uso incluem monitoramento de sistemas de software como máquinas virtuais, monitoramento de sistemas físicos como dispositivos conectados, o ambiente, sistemas de automação residencial, dentre outros.

2.3.2.1 InfluxDB

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. 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 gráficos 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 nomes únicos, que compõem a base de dados, podendo estar relacionada a uma ou mais tabelas. Conceitos como integridade referencial de dados – que garante sincronia de dados entre tabelas – e chaves primárias estão presentes e garantem 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

PostgreSQL é uma excelente implementação de um banco de dados relacional, de código aberto e sem custos para o uso. (STONES; MATTHEW, 2006) 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

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

2.3.5 Arquitetura de microsserviços

Seguindo a definição de NAMOT; 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 10.

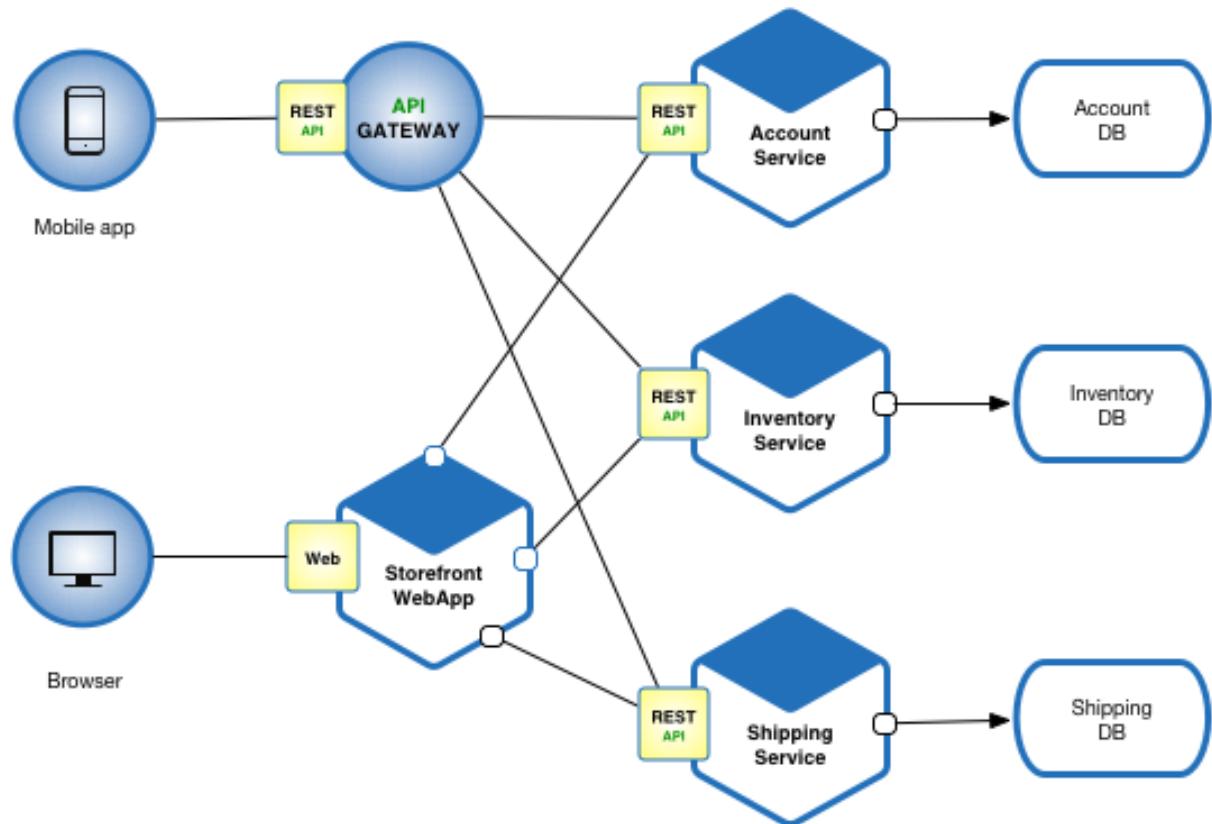


Figura 10 – Exemplo arquitetura de microsserviços

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)

2.3.6 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 largura de banda de rede.¹ O MQTT trabalha sobre 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 IBM² em 1999, sendo recentemente reconhecido como padrão pela OASIS (Organization 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 MQTT e um MQTT *broker*, 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* 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 MQTT. Ele aceita e transmite as mensagens através de múltiplos clientes conectados à ele.

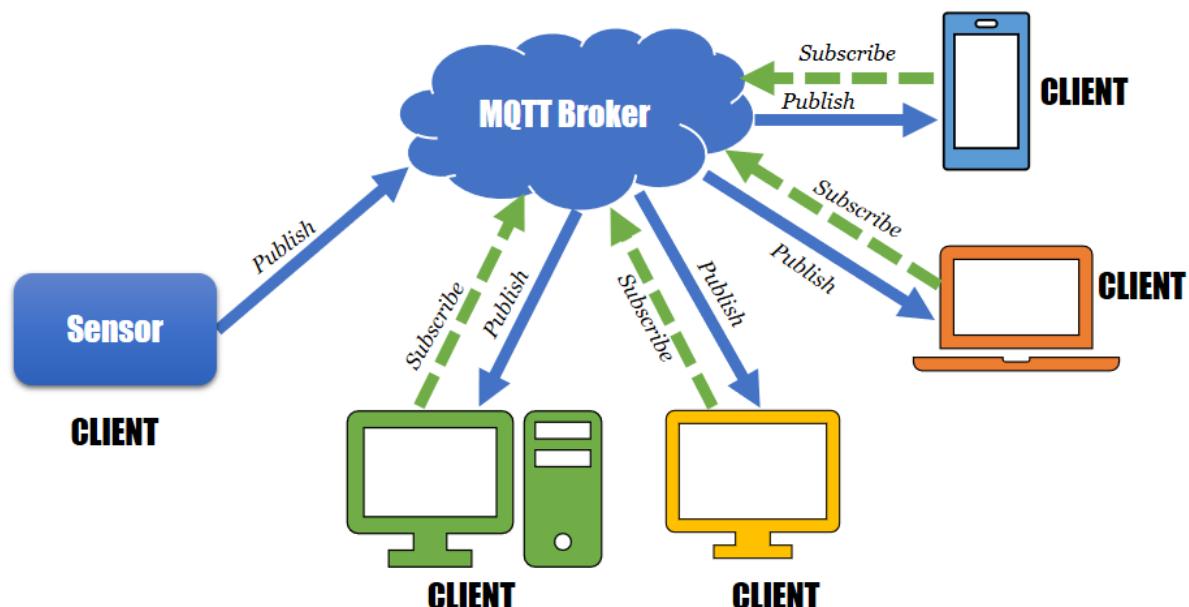


Figura 11 – Exemplo da arquitetura do MQTT

Fonte: LABORATORY (2018)

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

¹ <http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>

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>

para o *broker* MQTT.

A Figura 11 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 *publish/subscribe*, 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 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.

A validação do projeto será feita através de um sistema criado para emular os dados vindos dos sensores, atuadores e teclado. Este sistema é capaz de criar dados que simulam os sinais recebidos pelo sensor YF-S201, pelo teclado numérico e atuador.

3.1 Funcionamento do sistema

Nesta seção estão descritos o funcionamento do sistema, os parâmetros utilizados nele assim como as respostas esperadas.

3.1.1 Parâmetros de operação

Aqui são apresentados e explicados os parâmetros de operação e como eles ditam o funcionamento do sistema. 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. Cada usuário terá sua própria informação do seu sensor no HomeAssistant.

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 da quantidade de água por segundo que está passando 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 guardando-o no Banco de Dados PostgreSQL.

3.1.3 Ligar/desligar atuador

Para ligar o atuador, o sistema precisa identificar o usuário e autentica-lo. A identificação e autenticação do usuário 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 no teclado a tecla `#` seguido pelo seu id e em seguida `#`. Ao realizar a consulta, o sistema retorna o nome do usuário e requisita a senha.

Após digitar a senha, a tecla `#` deve ser pressionada para o sistema 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 16 mostra este fluxo de digitação, assim como a identificação do usuário pelo id.

3.2 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 Raspbian¹, que foi instalado no cartão SD a ser 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*², 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

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

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

um protocolo de rede criptográfico para operação de serviços de rede de forma segura³. 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 (192.168.2.60), que facilita o acesso SSH ao sistema.

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

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

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

Figura 12 – Exemplo de 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 de configuração do IP estático

3.3 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:

³ <https://www.ssh.com/ssh/>

- *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 também 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, possuindo 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 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 dados como componentes (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.

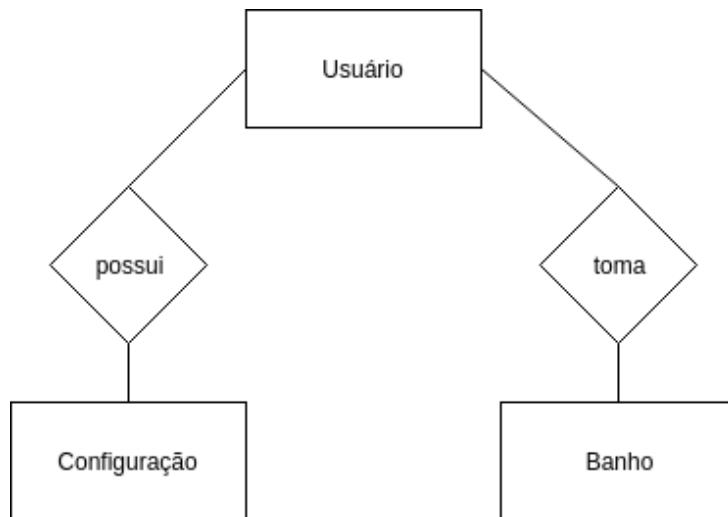


Figura 14 – Modelo Entidade Relacionamento do PostgreSQL

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. 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.

No Código 3.1 está um exemplo do código de configuração e implementação das funções do InfluxDB no sistema.

```

1 const Influx = require('influx');
2
3 class InfluxHandler {
4     constructor() {
5         this.influx;
6         this.host = 'influxdb';
7         this.database = 'water_flow_data'
8     }
9
10    connect() {
11        this.influx = new Influx.InfluxDB({
12            host: this.host,

```

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

```

13     database: this.database,
14     schema: [
15       measurement: 'flow_data',
16       fields: { flow: Influx.FieldType.INTEGER },
17       tags: ['user_id']
18     ]
19   );
20
21   this.influx.getDatabaseNames()
22   .then(names => {
23     if(!names.includes(this.database)) {
24       return this.influx.createDatabase(this.database);
25     }
26   })
27   .then(() => {
28     console.log('InfluxDB connected!');
29   })
30 }
31 }

```

Código 3.1 – 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, encapsulando e abstraindo o código para facilitar a leitura e entendimento das devidas funções de cada arquivo.

É possível ver a divisão dos arquivos utilizando a *Clean Architecture* do Sistema de Usuários na Figura 15. 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.

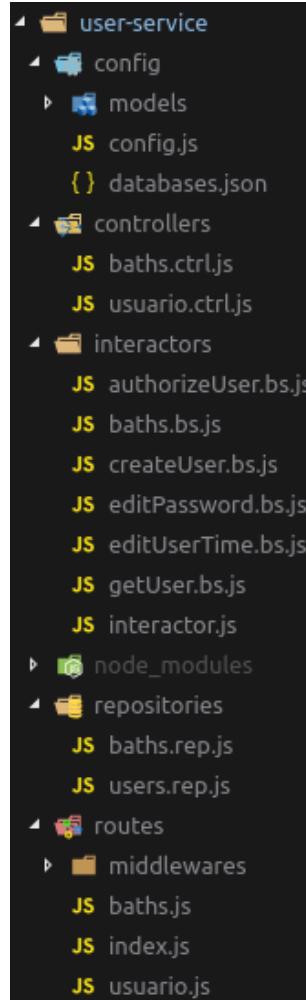


Figura 15 – Divisão dos arquivos do Sistema de Usuários

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*⁵, 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

⁵ Um *endpoint* é uma forma de comunicação entre os sistemas, o sistema disponibiliza *endpoints* que servem como endereços para acesso ao sistema, que podem receber e enviar informações, dependendo do que foi programado.

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.

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 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.

3.5.4 Sistema de simulação dos 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;
- Dados do teclado numérico: informação de qual tecla foi pressionada no teclado;
- Informações para o atuador: informação para ligar/desligar o atuador;

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 reconhece 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 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 16 mostra o sistema de simulação funcionando, com o usuário com o id 3 sendo autorizado e o sinal de ligar/desligar o atuador enviado.

```

camilo in mock-service on ✘ master via • v10.15.3
> node app.js
Press any key...
mqtt client connected to mqtt://localhost
You pressed the "*" key
{ topic: 'keys', message: '*' }
You pressed the "3" key
{ topic: 'keys', message: '3' }
You pressed the "#" key
{ topic: 'keys', message: '#' }
USUARIO {"id":3,"name":"Camilo","allowedBathTime":2}
You pressed the "1" key
{ topic: 'keys', message: '1' }
You pressed the "2" key
{ topic: 'keys', message: '2' }
You pressed the "3" key
{ topic: 'keys', message: '3' }
You pressed the "4" key
{ topic: 'keys', message: '4' }
You pressed the "#" key
{ topic: 'keys', message: '#' }
STARTING ACTUATOR, OPENING VALVULE
STOPING ACTUATOR, CLOSING VALVULE

```

Figura 16 – Exemplo do sistema de simulação

3.6 Configuração do HomeAssistant

O HomeAssistant pode ser instalado a partir de diversos métodos⁶, 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, deve-se copiar a imagem para o cartão SD que será inserido no RaspberryPi. Para isto, utilizamos o *balenaEtcher*.

⁶ <https://www.home-assistant.io/docs/installation>

Com a imagem devidamente escrita no cartão SD, basta inseri-lo no RaspberryPi, 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 deve ser acessado esta porta. No caso deste sistema, o IP do RaspberryPi foi colocado como estático *192.168.2.60*, então, para acessar a interface deve-se estar conectado na mesma rede do RaspberryPi e acessar o link *http://192.168.2.60:8123*, como podemos ver na Figura 17.

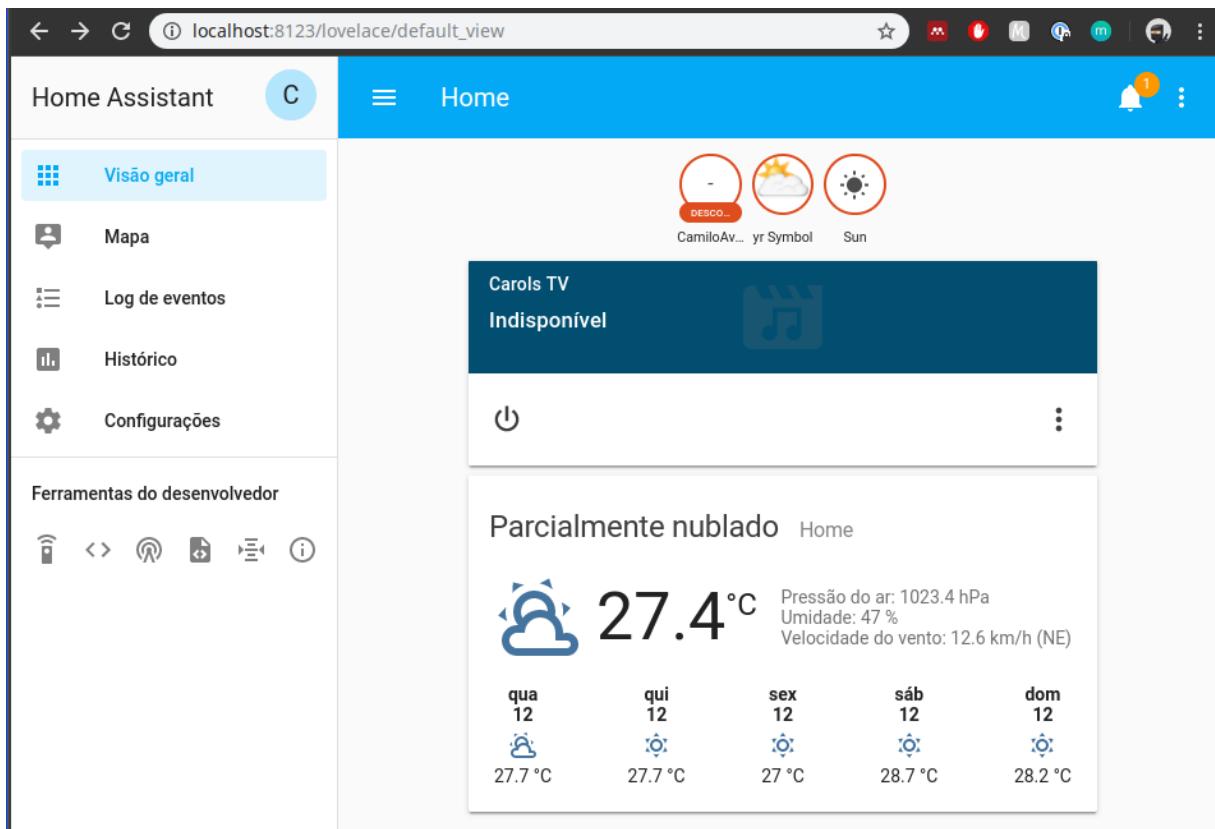


Figura 17 – Página inicial do HomeAssistant acessado por um cliente conectado na mesma rede local do servidor Hassbian

3.6.1 Sensores

A interface do HomeAssistant permite diversas configurações e inclusão de diferentes sensores, como mostra a Figura 9. 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.

3.7 Configuração do Grafana

A instalação do Grafana foi realizada através do *download* 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 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 18.

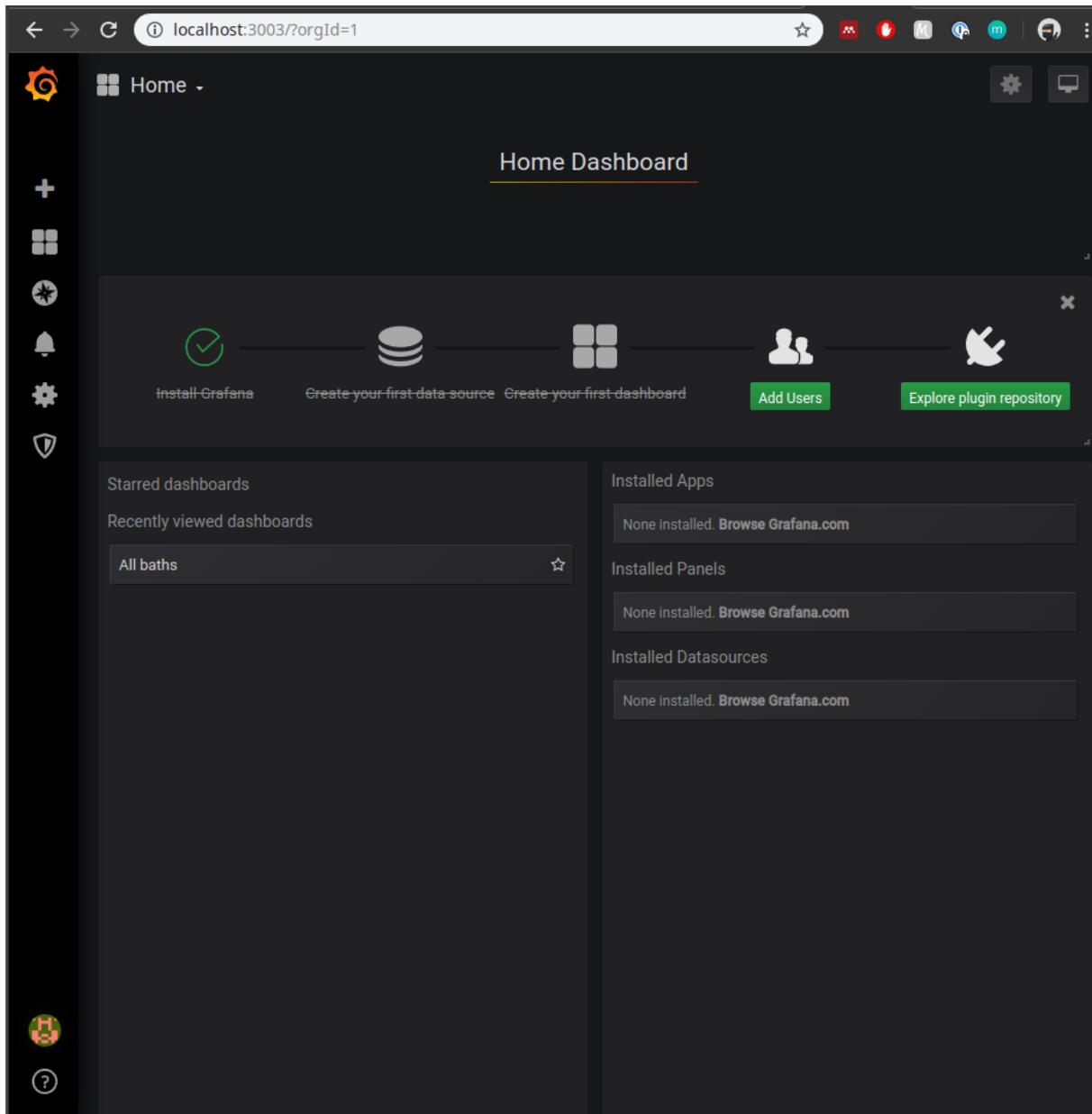


Figura 18 – Página inicial do Grafana

Ao acessar a interface do Grafana, é necessário criar um *dashboard*⁷ para a visualização dos dados do InfluxDB. Neste sistema criamos o *dashboard AllBaths*, que se conecta

⁷ Dashboards são painéis que mostram métricas e indicadores importantes para alcançar objetivos e metas traçadas de forma visual, facilitando a compreensão das informações geradas.

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 19.

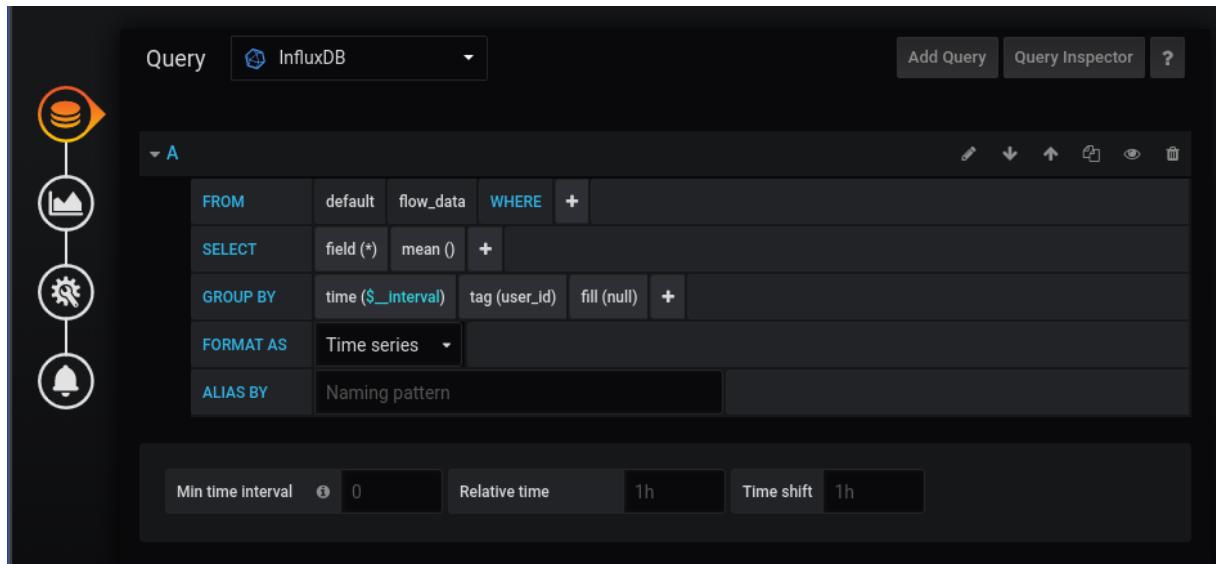


Figura 19 – Configuração do gráfico no Grafana

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.

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 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 consumir¹ o *endpoint* `http://localhost:3001/usuario/cadastrar` passando os parâmetros via *POST* no formato *JSON*, como na Figura 20.

A resposta deste *endpoint* pode ser conferida na Figura 20, e para garantir que o usuário foi incluído no banco de dados, podemos observar a Figura 20.

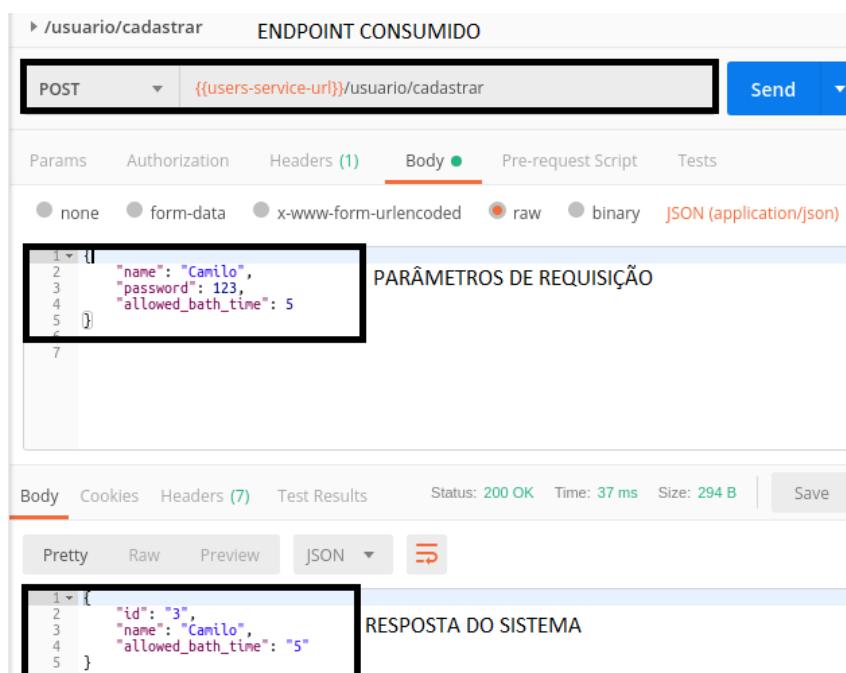


Figura 20 – Cadastro de usuários

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

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 21, para editar o tempo, e a imagem, para editar a senha.

A resposta pode ser observada na Figura 21, para o tempo, e Figura 22 para a senha.

The screenshot shows the Postman interface with the following details:

- Endpoint Consumido:** /usuario/editar-tempo
- Método:** POST
- URL:** {{users-service-url}}/usuario/editar-tempo
- Body:** JSON (application/json)


```

1 {
2   "id": 3,
3   "new_time": 2
4 }
```
- Parâmetros de Requisição:** (None, form-data, x-www-form-urlencoded, raw, binary)
- Resposta do Sistema:** Status: 200 OK, Time: 13 ms, Size: 299 B


```

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

Figura 21 – Edição de tempo permitido do usuário

Conseguimos cadastrar um banho ao consumir o *endpoint* `http://localhost:3001/banho` via *POST* com os parâmetros e resposta mostrados na Figura 23.

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 24.

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 25.

Finalmente, podemos autenticar os usuários enviando via *POST* os parâmetros para o *endpoint* `http://localhost:3001/usuario/autorizar`, obtendo como resposta a Figura 27, para usuário autenticado, e Figura 26, para usuário não autenticado, caso a senha esteja errada.

The screenshot shows a POST request to the endpoint {{users-service-url}}/usuario/editar-senha. The request body is JSON with fields id: 3 and new_password: 1234. The response status is 200 OK, message is "Usuário editado com sucesso".

```

1 {  

2   "id": 3,  

3   "new_password": 1234  

4 }

```

PARÂMETROS DE REQUISIÇÃO

```

1 {  

2   "status": 200,  

3   "message": "Usuário editado com sucesso"  

4 }

```

RESPOSTA DO SISTEMA

Figura 22 – Edição de senha do usuário

The screenshot shows a POST request to the endpoint {{users-service-url}}/banho. The request body is JSON with fields user_id: 3 and bath_time: 10. The response status is 200 OK, containing a detailed object with fields id, user_id, time, updatedAt, createdAt, and userId.

```

1 {  

2   "user_id": 3,  

3   "bath_time": 10  

4 }

```

PARÂMETROS DE REQUISIÇÃO

```

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 }

```

RESPOSTA DO SISTEMA

Figura 23 – Adicionando banhos ao usuário

The screenshot shows the Postman interface with the following details:

- URL:** {{users-service-url}}/usuario/3
- Method:** GET
- Body:** JSON (Pretty) - Returns a single user object:


```

1 {
2   "id": 3,
3   "name": "Camilo",
4   "allowedBathTime": "5"
5 }
```
- Status:** 200 OK
- Headers:** (7)
- Test Results:**
- Response Content:** RESPOSTA DO SISTEMA:
AS INFORMAÇÕES DO USUÁRIO

Figura 24 – Recuperar dados do usuário

The screenshot shows the Postman interface with the following details:

- URL:** {{users-service-url}}/banho/3
- Method:** GET
- Body:** JSON (Pretty) - Returns a list of bath objects:


```

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 ]
```
- Status:** 200 OK
- Headers:** (7)
- Test Results:**
- Response Content:** RESPOSTA DO SISTEMA:
OS BANHOS DOS USUÁRIOS

Figura 25 – Recuperar dados de banhos do usuário

The screenshot shows a POST request to the endpoint {{users-service-url}}/usuario/autorizar. The request body contains the following JSON:

```

1 {
2   "id": 3,
3   "password": 123
4 }
  
```

The response status is 200 OK, and the response body is:

```

1 [
2   "allowed": false
3 ]
  
```

Figura 26 – Exemplo de usuário não autorizado

The screenshot shows a POST request to the endpoint {{users-service-url}}/usuario/autorizar. The request body contains the following JSON:

```

1 {
2   "id": 3,
3   "password": 1234
4 }
  
```

The response status is 200 OK, and the response body is:

```

1 [
2   "allowed": true
3 ]
  
```

Figura 27 – Exemplo de usuário autorizado

4.2 Visualização via Grafana

Ao autenticar um usuário, o atuador é ativado e o sensor YF-S201 inicia a medição do fluxo, que é automaticamente enviado para o InfluxDB, podendo ser visualizado via Grafana. Os gráficos do grafana podem ser acessados via <http://localhost:3003>, como na Figura 18.

Pode ser observado na Figura 28 um gráfico do fluxo medido pelo sensor no horário de 09h30m até 09h35m, por dois usuários diferentes.

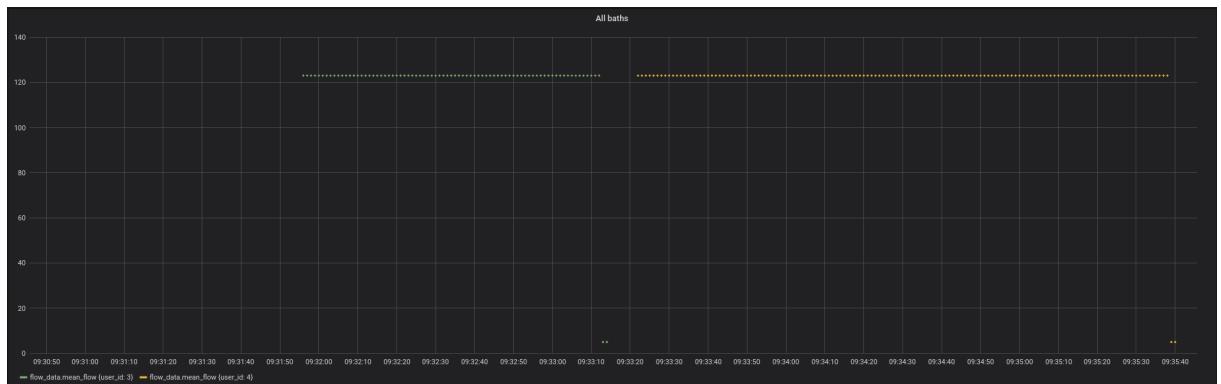


Figura 28 – Exemplo do gráfico no Grafana para usuários diferentes

4.3 Estado do atuador via HomeAssistant

O HomeAssistant é acessado via <http://localhost:8123>. Ao acessar o link, observamos os estados dos sensores dos usuários cadastrados na Figura 30, que encontram-se em estado desligado. Ao digitar corretamente a senha, o estado do sensor muda para ligado, como observado na Figura 29.

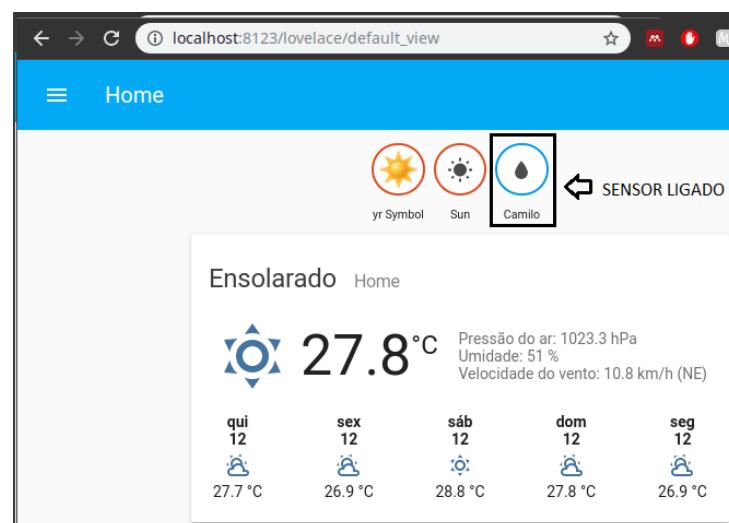


Figura 29 – Exemplo do sensor no HomeAssistant quando ligado

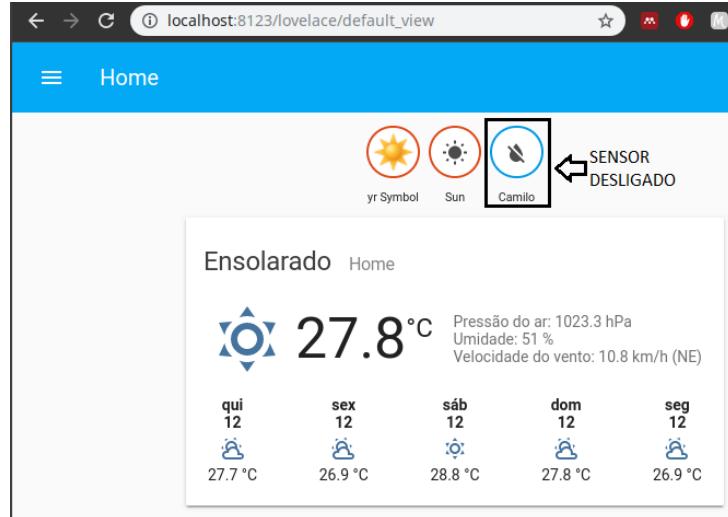


Figura 30 – Exemplo do sensor no HomeAssistant quando desligado

Ao cadastrar um usuário, o seu sensor é inserido no HomeAssistant automaticamente, Figura 31.

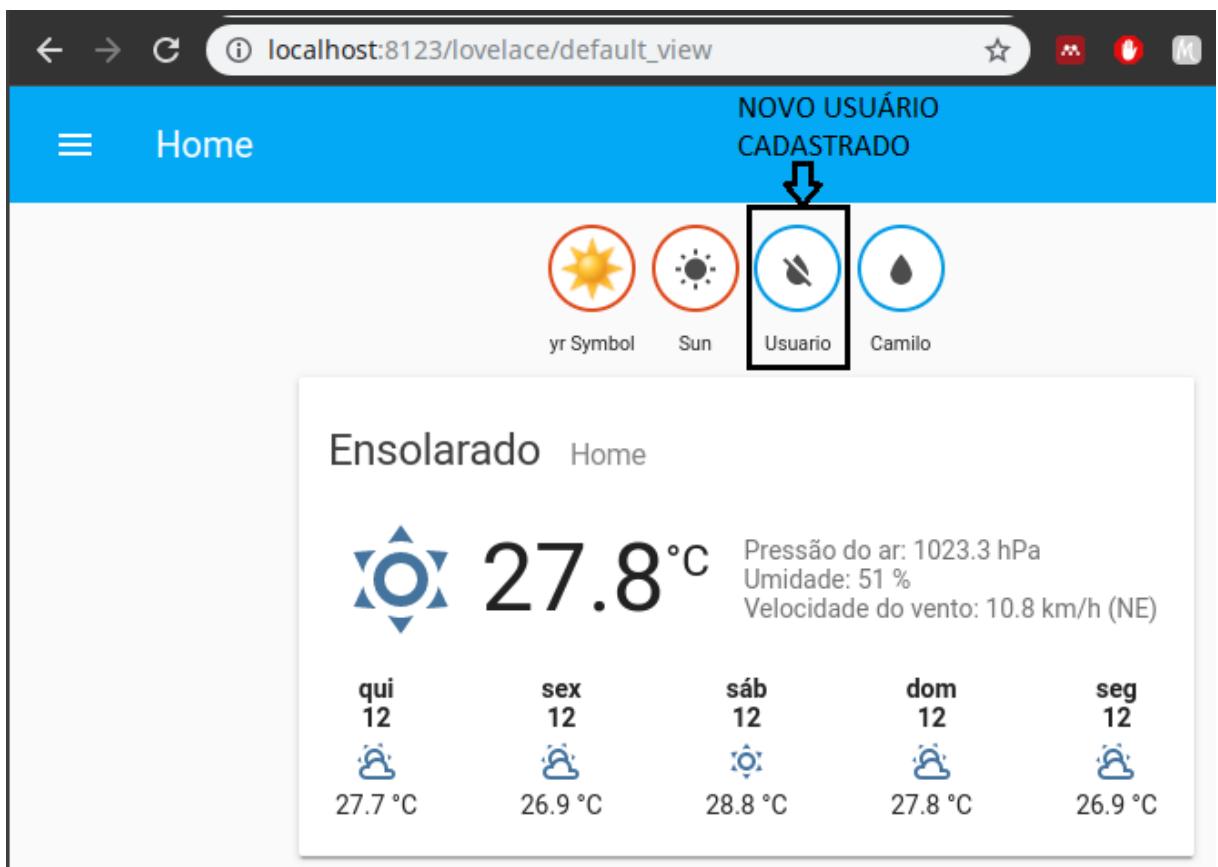


Figura 31 – Exemplo de um novo sensor quando um novo usuário é cadastrado

5 Considerações Finais

Este trabalho propõe um sistema capaz de monitorar remotamente a utilização da água em uma residência. Como mostrado, o sistema foi capaz de cadastrar, monitorar e editar usuários e banhos com gráficos e estados de sensores em tempo real remotamente.

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.

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 14.

BOSCAROLI, 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.sabajovem.org/chase>*, capturado em, 2007. v. 10, n. 11, p. 13, 2007. Citado na página 15.

CROTTI, Y. et al. Raspberry 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.; 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 17.

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 16, 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.
- LUNDIGAN, 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.
- MARTINS, N. A. Sistemas microcontrolados. *Uma abordagem com o Microcontrolador PIC 16F84*. Editora Novatec Ltda, 1^a 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 2 vezes nas páginas 17 e 18.
- OLIVEIRA, S. de. *Internet das Coisas com ESP8266, Arduino e Raspberry Pi*. [S.l.]: Novatec Editora, 2017. Citado na página 16.
- 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 3 vezes nas páginas 17, 18 e 19.
- REBOUÇAS, A. d. C. Água no brasil: abundância, desperdício e escassez. *Bahia análise & dados*, 2003. v. 13, p. 341–345, 2003. Citado na página 14.
- 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 17.
- 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.
- VERTIGO. *O que são microserviços*. 2018. <https://vertigo.com.br/o-que-sao-microservicos/>. Citado na página 23.

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.
18       saltRounds);
19
20     const user = await usersRep.createUser({ name, password: hash,
21       allowedBathTime });
22
23     if (!user) {
24       throw 'Nao foi possivel criar o usuario!';
25     }
26
27     const response = {
28       id: user.user_id,
29       name: user.user.name,
30       allowed_bath_time: user.allowed_bath_time,
31     };
32
33     return response;
34 }
```

```
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.
14       bcrypt.saltRounds);
15
16     const editedUser = await usersRep.editPassword({ id, password: hash
17       });
18
19     const response = editedUser[0] === 1 ? {
20       status: 200,
21       message: 'Usuario editado com sucesso'
22     } :
23     {
24       status: 500,
25       message: 'Erro ao editar usuario'
26     };
27
28   return response;
29 }
30 }
```

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

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

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

```
1 import Interactor from './interactor';
2 import bathsRep from '../repositories/baths.rep';
3
4 class BathsBs extends Interactor {
5   constructor(){
6     super();
7   }
8
9   async execute({ user_id, bath_time }) {
10
11     const bath = await bathsRep.createBath({
12       user_id,
13       bath_time
14     });
15
16     if(!bath) {
17       throw 'Nao foi possivel cadastrar o banho';
18     }
19
20     return bath;
21   }
22
23   async getBaths ({ user_id }) {
24
25     const baths = await bathsRep.getBaths({
26       user_id,
27     });
28
29     return baths;
30   }
31 }
32
33 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
11      to connect
12    this.password = '100200300';
13  }
14
15  connect() {
16    const self = this;
17    this.mqttClient = mqtt.connect(this.host, { username: this.username,
18      password: this.password });
19
20    this.mqttClient.on('error', (err) => {
21      console.log(err);
22      this.mqttClient.end();
23    });
24
25    this.mqttClient.on('connect', () => {
26      console.log(`mqtt client connected to ${this.host}`);
27    });
28
29    this.mqttClient.on('close', () => {
30      console.log('mqtt client disconnected');
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

```
1 const requestService = require('./requestService');
2
3 class KeysHandler {
4   constructor(mqttClient) {
5     this.keyBuffer = '';
6     this.working = false;
7     this.getPassword = false;
8     this.userId;
9     this.mqttClient = mqttClient;
10 }
11
12 async handle(message) {
13   if(message === '*') {
14     this.working = !this.working;
15     this.getPassword = false;
16     return;
17   }
18
19   if((this.working || this.getPassword) && message !== '#') {
20     this.keyBuffer += message;
21   }
22
23   if(message === '#') {
24     this.working = false;
25     try {
26       const response = await this._requestUserService();
27       console.log(response);
28       this.getPassword ? (response.allowed ? this.mqttClient.
29         sendMessage('start', 'actuator') : console.log('NAO
30           AUTORIZADO')) :
31         this.mqttClient.sendMessage(JSON.stringify(response), 'user');
32       this.getPassword = !this.getPassword;
33     } catch (err) {
34       console.log(err.message)
35     } finally {
36       this.keyBuffer = '';
37     }
38   }
39
40   async _requestUserService() {
41     const requestOptions = this.getPassword ? {
42       type: 'POST',
43       endpoint: 'usuario/autorizar',
44       body: {
45         id: this.userId,
46         password: this.keyBuffer
47       }
48     }
49   }
50 }
```

```
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