

DANIELA SAYURI YASSUDA
GABRIELA SOUZA DE MELO
HUGO DA SILVA POSSANI
VICTOR TAKASHI HAYASHI

Hedwig - Casa Conectada

São Paulo
2017

DANIELA SAYURI YASSUDA
GABRIELA SOUZA DE MELO
HUGO DA SILVA POSSANI
VICTOR TAKASHI HAYASHI

Hedwig - Casa Conectada

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro Eletricista
com ênfase em Computação.

DANIELA SAYURI YASSUDA
GABRIELA SOUZA DE MELO
HUGO DA SILVA POSSANI
VICTOR TAKASHI HAYASHI

Hedwig - Casa Conectada

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro Eletricista
com ênfase em Computação.

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. Reginaldo Arakaki

Co-orientador:
Eng. Marcelo Pita

São Paulo
2017

AGRADECIMENTOS

À Fabio Hirotsugu Hayashi, principal responsável pela montagem e circuito eletrônico dos módulos. Sem sua ajuda, esse trabalho não seria possível.

RESUMO

O crescente desenvolvimento das tecnologias de Internet das Coisas (IoT) traz inúmeras oportunidades para a reinvenção dos nossos arredores, sendo a inteligência e eficiência peças chaves na concepção de novos produtos. Mesmo os mais básicos aparelhos, que hoje operam isoladamente, passarão a fazer parte de um sistema complexo, integrado, no qual a troca de informações é requisito básico para o funcionamento. Dessa forma, a elaboração de uma sólida infraestrutura de comunicação é vital ao processo. Juntamente com tais tecnologias, surgem conceitos atuais para suas aplicações, como os de Casas e Cidades Inteligentes - *Smart Homes* e *Smart Cities*, respectivamente -, que oferecem um novo paradigma responsável por modernizar a vivência urbana.

Com base nesse cenário, o presente trabalho tem o objetivo de desenvolver um sistema completo para casas inteligentes. Nele, são exploradas as tecnologias de comunicação e conectividade entre dispositivos, criando assim uma plataforma acessível e expansível para a automatização e monitoração de residências - tudo isso com baixo custo envolvido.

Circuitos microcontrolados, atuadores, sensores e radiotransmissores são vastamente utilizados nos módulos físicos, que ficam instalados na residência. A infraestrutura local de comunicação é formada por um protocolo para troca de mensagens e um sistema de mensageria do tipo publicação e subscrição coordenado por um servidor. O usuário final interage com o sistema por meio de aplicativos web, que utilizam-se de serviços na nuvem e conectam-se com as casas por meio de WebSockets.

Todo o protótipo desenvolvido mostrou-se viável e funcional, atendendo aos requisitos propostos e aos testes realizados. Espera-se que esta iniciativa possa ser continuada em cima da fundação atual.

Palavras-Chave – Internet das Coisas, Casas Inteligentes, Cidades Inteligentes, Infraestrutura de comunicação, Mensageria.

ABSTRACT

The increasing development of the Internet of Things (*IoT*) offers countless opportunities to reinvent our surroundings, with intelligence and efficiency being key concepts during the creation of new products. Even the most basic devices, that currently work in isolation, will become part of a complex integrated system, in which information exchange will be a basic requirement for operation. Thus, the establishment of a solid communication infrastructure is a vital part of the process. In conjunction with such technologies, modern concepts for their application are devised, such as Smart Homes and Smart Cities, offering a new paradigm responsible for the modernization of urban living.

Based on the previous scenario, this work aims to provide a complete system for Smart Houses. It explores communication technologies and connectivity between devices, creating an accessible and expandable platform for residency automation, with low production costs.

Microcontroller circuits, in addition to actuators, sensors and radio transmitters, are greatly used in the hardware modules. The local communication infrastructure is composed by a messaging exchange protocol and a publisher/subscriber messaging broker controlled by a server. The end user interacts with the system through a web client, which uses cloud services and are connected to the home via WebSockets.

All of the prototypes developed proved to be viable and functional, fulfilling the specified requirements and passing performed tests. Hopefully, this initiative will be continued and further improved on top of this underlying foundation.

Keywords – *IoT*, Smart Houses, Smart Cities, Communication infrastructure, Messaging Systems.

LISTA DE FIGURAS

1	Crescimento do número de conexões M2M por tipo de aplicação	26
2	Logotipo do projeto Hedwig	28
3	Camadas da arquitetura usada no Projeto HomeSky. As camadas em verde correspondem às bibliotecas desenvolvidas no trabalho.	33
4	Primeira versão da arquitetura do projeto Hedwig	40
5	Segunda versão da arquitetura do projeto Hedwig	41
6	Diagrama ilustrativo do módulo de Acesso ao Portão	44
7	Diagrama ilustrativo do módulo de Quarto/Sala/Cozinha	46
8	Aquário de Santo André	48
9	Diagrama ilustrativo do módulo de Aquário	49
10	Diagrama ilustrativo do módulo de Interface com Sistema de Alarmes . . .	50
11	Raspberry Pi 3 Modelo B	52
12	Comparação entre WebSockets e <i>polling</i>	55
13	Comparação entre uma aplicação monolítica (esquerda) e com micro-serviços (direita)	56
14	Diagrama de interação na autenticação por JWT	63
15	Visão alto nível da comunicação no Hedwig	65
16	Arquitetura do servidor local	72
17	Componentes e implementação na nuvem	100
18	Monitoramento do servidor na nuvem	101
19	Diagrama de funcionamento do Redux	105
20	Telas principais do aplicativo backup	111
21	Visão geral de uma planta com o aplicativo backup	111
22	Menu Principal	112

23	Teclado para digitação da senha	112
24	Página de configuração TP Link	113
25	Rotina de multiplexação de procedimentos no tempo	117
26	Tratamento de indisponibilidade de recursos	118
27	Tratamento de ataque de DoS Local	119
28	Exemplo de estado da fila de saída de mensagens MQTT do Módulo	120
29	Diagrama PCB do Módulo Base	121
30	Diagrama Elétrico do Módulo Base	122
31	Entradas Em A0	123
32	Funcionamento do Circuito de Antitravamento	124
33	Evolução do Hardware (de fevereiro/17 a setembro/17)	124
34	Evolução da Caixa de Proteção e Display	125
35	Materiais e preparação da placa padrão	125
36	Posicionamento dos Componentes	126
37	Preparação dos displays com o I2C e soldagem	126
38	Preparação das caixas para os módulos	127
39	Caixa protetora; ligação dos botões e cabos de força	127
40	Quatro Módulos Prontos	128
41	Log na página do Aplicativo Backup	132
42	Módulo com cartão SD para coleta local de dados	132
43	Módulos instalados em Santo André	133
44	Controle do Sistema de Alarme e módulo de interface de Jarinu	133
45	Módulo Básico e sensor de presença no corredor de Jarinu	134
46	Sensor de abertura das portas da cozinha (à esquerda) e da sala (à direita) .	134
47	Módulo instalado para coleta de dados do nível da caixa d'água	135
48	Relação dos Módulos e Dados Coletados	135

49	Curva Diária Santo André - Terça	138
50	Temperatura e Aquecedor - Aquário	139
51	Consolidado Diário - Aquário	140
52	Memória e loops do Corredor	141
53	Consolidado no Período - Corredor	142
54	Consolidado Diário dos Sensores - Corredor	143
55	Uso dos acionamentos no período para a lâmpada do Corredor	144
56	Uso dos acionamentos por dia para a lâmpada do Corredor	144
57	curva Diária do Módulo de Acesso	145
58	Nível da Caixa d'água - Consolidado Diário	146
59	Tempo para Encher a Caixa d'água - Período	147
60	Atividade da Porta da Sala de Jardinu	147
61	Atividade da Presença da Sala de Jardinu	148
62	Consolidado no Período - Jardinu	149
63	Resumo do Aquário	161
64	Disponibilidade Aquário - Período	162
65	Disponibilidade Aquário - Dia	162
66	Memória Livre do Aquário	163
67	Resumo do Corredor	163
68	Sensores do Corredor - Período	164
69	Uso Rele 2 Corredor - Dia	164
70	Uso Rele 2 Corredor - Período	165
71	Consolidado Diário Corredor	165
72	Resumo da Lavanderia	166
73	Disponibilidade da Lavanderia - Dia	166
74	Disponibilidade da Lavanderia - Período	167

75	Memória Livre Lavanderia	167
76	Sensores da Lavanderia - Período	168
77	Sensores da Lavanderia - Dia	168
78	Uso Rele 1 Lavanderia - Dia	169
79	Uso Rele 1 Lavanderia - Período	169
80	Uso Rele 2 Lavanderia - Dia	170
81	Uso Rele 2 Lavanderia - Período	170
82	Consolidado Diário da Lavanderia	171
83	Resumo da Sala/Cozinha	171
84	Disponibilidade da Sala/Cozinha - Período	172
85	Memória Livre da Sala/Cozinha	172
86	Sensores da Sala/Cozinha - Dia	173
87	Sensores da Sala/Cozinha - Período	173
88	Uso do Rele 1 Sala/Cozinha - Dia	174
89	Uso do Rele 1 Sala/Cozinha - Período	174
90	Uso do Rele 2 Sala/Cozinha - Dia	175
91	Uso do Rele 1 Sala/Cozinha - Período	175
92	Porta de Acesso - Dia	176
93	Porta de Acesso - Período	176
94	Resumo da Entrada	177
95	Consolidado da Entrada - Período	177
96	Memória Livre Entrada	178
97	Sensores da Entrada - Dia	178
98	Sensores da Entrada - Período	179
99	Uso Lâmpada Entrada - Dia	179
100	Uso Lâmpada Entrada - Período	180

101	Uso da Entrada - Dia	180
102	Uso da Entrada - Período	181
103	Jarinu - Consolidado por Dia da Semana	181
104	Configuração de Cores e Layout do Display do Módulo	183
105	Configurações de alertas, alarmes, reles, log e menu de ferramentas do Aplicativo Backup	184
106	Atualização de Firmware, menu de configurações avançadas e DHT	184
107	Configurações de Radio Frequência, Blynk e WiFi	184
108	Configurações de Nome e Ponto de Acesso WiFi	185

LISTA DE TABELAS

1	Riscos para o Aquário	48
2	Análise Consolidada de Disponibilidade	136
3	Análise Consolidada de Uso dos Reles	137
4	Lista de Materiais	160

LISTA DE SIGLAS

ACME	<i>Automatic Certificate Management Environment</i>
API	<i>Appliaction Programming Interface</i>
CAGR	<i>Compound Annual Growth Rate</i>
CDN	<i>Content Delivery Network</i>
CDMA	<i>Code Division Multiple Access</i>
CORS	<i>Cross-Origin Resource Sharing</i>
DoS	<i>Denial of Service</i>
E/S	Entrada / Saída
HTTP	<i>Hypertext Transfer Protocol</i>
IDE	<i>Integrated Development Environment</i>
IoC	<i>Inversion of Control</i>
IoT	<i>Internet of Things</i>
IP	<i>Internet Protocol</i>
JSON	<i>JavaScript Object Notation</i>
JVM	<i>Java Virtual Machine</i>
M2M	<i>Machine to Machine</i>
MOOC	<i>Massive Open Online Course</i>
NoSQL	<i>Not Only SQL</i>
PWA	<i>Progressive Web App</i>
QoS	<i>Quality of Service</i>
REST	<i>Representational State Transfer</i>
SOA	<i>Service-Oriented Architecture</i>
SQL	<i>Structured Query Language</i>
SSID	<i>Service Set Identifier</i>
TCP	<i>Transmission Control Protocol</i>
TLS	<i>Transport Layer Security</i>
UI	<i>User Interface</i>
URL	<i>Uniform Resource Locator</i>
UX	<i>User Experience</i>
WLAN	<i>Wireless LAN</i>
WPA	<i>Wi-Fi Protected Access</i>
XML	<i>eXtensible Markup Language</i>
YAML	<i>YAML Ain't Markup Language</i>

SUMÁRIO

1	Introdução	25
1.1	Motivação	25
1.2	Projeto Hedwig	26
1.2.1	Objetivo	26
1.2.2	Nome do Projeto	28
1.2.3	Logo	28
1.3	Aplicações	28
2	Projetos Relacionados	31
2.1	Sistemas Existentes no Mercado	31
2.1.1	Sistemas Comerciais	31
2.1.2	Sistemas Open Source	32
2.2	Projeto HomeSky	32
3	Especificação	35
3.1	Stakeholders	35
3.2	Requisitos	36
3.2.1	Requisitos Funcionais	36
3.2.2	Requisitos Não-Funcionais	37
3.2.3	Funcionalidades por Nível de Conectividade	38
4	Arquitetura	39
4.1	Visão geral	39
4.2	Evolução arquitetural	39
4.3	Criação de Módulos	42

4.3.1	Módulos Base	43
4.3.1.1	ESP8266	43
4.3.2	Módulo de Acesso	44
4.3.3	Módulo de Quarto/Sala/Cozinha	46
4.3.4	Módulo de Aquário	48
4.3.5	Módulo de Interface com Sistema de Alarmes	50
4.4	Controlador Local	51
4.4.1	Raspberry Pi	51
4.5	Servidor na nuvem	52
4.5.1	Computação em nuvem	52
4.5.2	Banco de dados não-relacional	53
4.5.3	Banco de dados em memória	53
4.5.4	WebSockets	54
4.5.5	Arquitetura de Microsserviços	55
4.5.5.1	Características	55
4.6	Cliente Web	58
4.6.1	<i>Progressive Web Apps</i>	58
4.6.1.1	Contexto	58
4.6.1.2	Conceito	59
4.6.1.3	Tecnologias e técnicas	60
4.6.1.4	Aplicações	61
4.6.2	<i>JSON Web Tokens</i>	61
4.6.2.1	Definição	61
4.6.2.2	Autenticação	62
4.7	Comunicação	62
5	Metodologia	67

5.1	Gerência do projeto	67
5.1.1	Gerência de Escopo Tempo	67
5.1.2	Gerência de Partes Interessadas Aquisição	67
5.1.3	Gerência de Processos de Software	67
5.1.4	Gerência de Partes Interessadas	68
5.1.5	Gerência de Comunicação	68
5.1.6	Gerência de Escopo	68
5.1.7	Gerência de Riscos	68
5.2	Pesquisa bibliográfica	68
5.3	Ferramentas e tecnologias	68
6	Implementação	69
6.1	Morpheus	69
6.1.1	Descrição	69
6.1.2	Plataforma	69
6.1.3	Tecnologias utilizadas	70
6.1.4	Requisitos	72
6.1.4.1	Requisitos Funcionais	72
6.1.4.2	Requisitos Não Funcionais	74
6.1.5	Especificações	74
6.1.5.1	Tópicos	74
6.1.5.2	Regras de negócio	75
6.1.5.3	Definição de interfaces	76
6.1.5.4	Definição das mensagens	76
6.1.5.5	Testes realizados da comunicação Morpheus e módulos: . . .	84
6.1.6	Comunicação entre Morpheus e Nuvem	88
6.1.7	WebSocket	89

6.1.7.1	Morpheus	89
6.1.7.2	Nuvem	89
6.1.8	Configurações	91
6.1.8.1	Configuração do <i>MQTT Mosquitto broker</i>	91
6.1.8.2	Guia de instalação (Testado com Ubuntu 16.10 x64) . . .	92
6.1.8.3	Criação dos certificados	95
6.1.8.4	Senhas	96
6.1.8.5	Casos de teste para Controle de Acesso nos Tópicos <i>MQTT</i> entre módulos e nuvem	96
6.2	Servidor na nuvem	97
6.2.1	Descrição	97
6.2.2	Requisitos	98
6.2.2.1	Requisitos funcionais	98
6.2.2.2	Requisitos não-funcionais	99
6.2.3	Tecnologias usadas	99
6.2.4	Infraestrutura	100
6.2.5	Segurança	101
6.2.6	Operação	101
6.3	Aplicativo de dashboard	102
6.3.1	Descrição	102
6.3.2	Requisitos	102
6.3.2.1	Requisitos funcionais	102
6.3.2.2	Requisitos não-funcionais	104
6.3.3	Tecnologias utilizadas	104
6.3.4	Interface	106
6.3.4.1	Identidade visual	106
6.3.5	Interações	107

6.3.5.1	Dados	107
6.3.5.2	Ações	107
6.3.5.3	Conectividade	107
6.3.5.4	Aplicativo na tela inicial do dispositivo móvel	107
6.3.6	Implantação	108
6.3.7	Segurança	108
6.3.8	Performance	108
6.4	App Backup	109
6.4.1	Requisitos	109
6.4.2	Tecnologias usadas	110
6.4.3	Navegação	110
6.4.4	Configurações	112
6.4.5	Abertura de porta do roteador	113
6.4.6	Controle remoto	114
6.4.7	Notificações	114
6.4.8	Offset Temperatura e Umidade	115
6.4.9	Hard Reset	115
6.4.10	Setup inicial	115
6.5	Módulos	116
6.5.1	Rotinas	116
6.5.1.1	Multiplexação no tempo	116
6.5.1.2	Tratamento de indisponibilidade	117
6.5.1.3	DoS Local (<i>Evil Twin</i>)	118
6.5.1.4	Comunicação por MQTT	119
6.5.2	Diagrama	120
6.5.3	Montagem	124

6.5.3.1	Montagem	128
7	Aprendizado de Máquina e Análise de Dados	131
7.1	Coleta e Análise de Dados	131
7.1.1	Coleta	131
8	Conclusões	151
	Referências	153
	Anexo A – Códigos das aplicações desenvolvidas	157
	Anexo B – Lista de materiais para montagem dos módulos	159
	Anexo C – Dados Coletados de 10/09/2017 a 13/11/2017	161
	Anexo D – Imagens de configuração para o aplicativo backup	183

1 Introdução

1.1 Motivação

Há uma expectativa de que, no ano de 2017, o número de casas inteligentes aumente cerca de 17% nos Estados Unidos (MCKINSEY&CO, 2016), onde já se tem investimentos de grandes empresas, como *Google*, *Amazon* e *Apple*. O interesse nessa área é tamanho que a *Google* investiu cerca de 5 milhões de dólares em um comercial de seu produto *Google Home* no *Super Bowl* 2017, jogo que decide o time campeão da temporada de futebol americano nos EUA (KENNEMER, 2017). É esperado que os consumidores invistam cada vez mais em casas inteligentes nos próximos anos, com previsões de que o valor total desse mercado chegue a mais de 63 bilhões de dólares em 2020 (BUSINESS WIRE, 2017).

As aplicações de automação residencial não mais se limitam aos sistemas de iluminação, controle da ventilação e temperatura de cômodos. Elas contemplam também segurança, eficiência energética e até mesmo soluções voltadas à área da saúde, com dispositivos desenvolvidos especificamente para pessoas idosas, com problemas de mobilidade ou doenças crônicas (I-SCOOP, 2017).

Os avanços das tecnologias de Internet das Coisas (*Internet of Things* ou *IoT*) apontam para um futuro no qual qualquer dispositivo da casa possa estar conectado à Internet, onde surge uma série de possibilidades de inovações em funcionalidades e integrações. É previsto que o número de conexões *Machine to Machine* (M2M) de dispositivos de casa conectada tenha uma taxa de crescimento anual composta (*Compound Annual Growth Rate* ou *CAGR*) de 18% entre 2016 e 2021 (CISCO, 2017).

Figura 1: Crescimento do número de conexões M2M por tipo de aplicação



Fonte: (CISCO, 2017)

As oportunidades trazidas pelo conceito de *IoT* à área de automação residencial são uma grande motivação para esse projeto. Também destaca-se a possibilidade de promover tais conhecimentos ao mercado nacional, com a criação de produtos e a sua adequação às necessidades dos potenciais consumidores brasileiros. Mesmo nos Estados Unidos, ainda é necessário tempo até que as casas conectadas se consolidem, de modo que há uma conjuntura propícia ao pioneirismo, com a criação de tecnologias de *IoT* à preços acessíveis, capazes de serem absorvidos pela demanda de mercados emergentes.

1.2 Projeto Hedwig

1.2.1 Objetivo

A contribuição do projeto para o avanço das tecnologias de Internet das Coisas fundamenta-se na criação de um sistema com arquitetura modularizada, e em camadas, com funcionalidades locais e de nuvem, provendo uma API que permita seu acesso por diversos clientes — como websites ou aplicativos para smartphones — que seja capaz de monitorar e agir em diversos módulos presentes na residência do usuário final do sistema. O projeto irá disponibilizar módulos físicos, prontos para serem instalados e configurados na residência, sem que sejam necessários conhecimentos avançados de eletrônica ou computação.

Para a elaboração do projeto, e o alinhamento das expectativas e requisitos que o motivaram, os seguintes conceitos desempenharam papéis relevantes:

Robustez

Com foco na robustez e disponibilidade, devem ser previstos níveis de operação para o sistema, os quais dispõem de diferentes requisitos de funcionalidades para garantir serviços essenciais, mesmo na ocorrência de problemas como a queda do servidor local, indisponibilidade de internet, falha no roteador, dentre diversas outras possibilidades. Há medidas tratativas, nos diferentes níveis, para a tentativa automática de reconexão, monitoramento de estado e manutenções preventivas e corretivas do sistema.

Modularidade

A modularidade, principalmente relacionada aos dispositivos físicos e as decisões arquiteturais, promove a independência de funcionamento entre as partes, e contribui no atendimento aos requisitos de robustez e disponibilidade. Em relação aos dispositivos, também representa diminuição nos custos de produção, e a possibilidade de que dispositivos específicos sejam desenvolvidos para aplicações diversas.

Camadas

A arquitetura do projeto é concebida em camadas e níveis, cujas responsabilidades são independentes. Cada parte do sistema exerce um conjunto de tarefas específicas (de transporte, análise, tomada de ações, etc.), e seus efeitos são traduzidos em entradas para o nível seguinte.

Machine Learning

Geração de aprendizado de máquina por meio de análise automática do uso do sistema pelos usuários, de forma a entender suas rotinas e poder atuar no conhecimento obtido, com notificações, alertas e acionamentos automáticos de funções para o usuário.

Segurança

A proteção da privacidade do usuário é tão importante, ou talvez mais, quanto a proteção física da casa. Assim, necessita-se que o sistema seja seguro, e que o fluxo de dados trocados entre as partes ocorra em meios protegidos. Foram utilizadas protocolos criptográficos modernos para a proteção da troca de mensagens entre servidor local e serviços de nuvem. Todo usuário é autenticado e autorizado por meio de *tokens* seguros, ao utilizar o aplicativo cliente, e a comunicação interna da casa é realizada em canais restritos.

1.2.2 Nome do Projeto

O nome do projeto foi escolhido em homenagem a Hedy Lamarr. Nascida Hedwig Eva Maria Kiesler (SHEARER, 2010), a atriz e inventora desenvolveu, durante a Segunda Guerra Mundial, um aparelho de interferência em rádio para despistar radares nazistas, cujos princípios estão incorporados nas tecnologias atuais de *Wi-Fi*, *CDMA* e *Bluetooth* (EFF, 1997). Baseado na ideia de um sistema de comunicação seguro, e como reconhecimento de seu trabalho, foi dado esse nome ao projeto aqui descrito.

1.2.3 Logo

O logo do projeto é uma coruja, em referência à coruja *Hedwig* do personagem *Harry Potter*, da série de livros de mesmo nome. A coruja é responsável por encaminhar mensagens de maneira segura entre os interlocutores, assim como o projeto desenvolvido promove uma maneira segura de comunicação com sua casa.

Figura 2: Logotipo do projeto Hedwig



1.3 Aplicações

Como aplicações do projeto Hedwig, destacam-se a automação no uso de eletrodomésticos e iluminação, segurança no acesso à casa, economia nas contas de energia elétrica, além de monitoramento remoto de pessoas que moram sozinhas, principalmente pessoas idosas, garantindo a tranquilidade de seus familiares e mantendo a segurança do indivíduo.

Exemplos de módulos que podem ser incluídos no sistema são: quarto (despertador, iluminação, monitoramento de temperatura e umidade); cozinha (*timer*, iluminação, mo-

nitoramento de presença e gás); acesso (controle de abertura, monitoramento de estado); externo (monitoramento de temperatura, umidade, energia elétrica e consumo de água); corredor (monitoramento de presença, iluminação); chuveiro (controle de temperatura/potência a partir do perfil de usuário e temperatura externa) e ar condicionado (controle da potência a partir do monitoramento das temperaturas internas e externas da casa).

A presença de funcionalidades de aprendizado de máquina incrementa o sistema, traz facilidades e promove maior adaptação às rotinas dos moradores. O sistema é capaz de aprender com *feedbacks* do usuário, seja pelo monitoramento dos módulos ou por respostas dadas pelo aplicativo, podendo atuar em questões de segurança (*safety*), personalizações e até mesmo em possíveis sugestões de produtos relacionados aos hábitos do cliente.

2 Projetos Relacionados

2.1 Sistemas Existentes no Mercado

2.1.1 Sistemas Comerciais

Atualmente, já existem sistemas comerciais de automação residencial — a maioria deles atuando de maneira mais forte do mercado norte-americano. Alguns dos sistemas mais populares nessa linha são o *Amazon Echo* e o *Google Home*.

O *Amazon Echo*¹ consiste em um *smart speaker* (alto-falante inteligente) conectado ao assistente pessoal Alexa, também da Amazon, que é capaz de entender comandos de voz. Inicialmente, funcionava como uma maneira de encomendar produtos, mas, atualmente, além de ser assistente pessoal, também é capaz de controlar diversos *smart devices* da casa, como um *hub* de automação residencial. Uma limitação deste produto é dependência de conexão *wireless* de Internet, não sendo capaz de operar em nenhum nível sem ela.

Uma característica interessante do Alexa é a possibilidade de adição de novas *skills* (habilidades) por desenvolvedores, que possuem acesso a uma API pública, documentada e disponibilizada *online*. Dessa forma, seu *skillset* é passível de grande expansão e personalização. Além disso, o serviço de voz desse sistema, conhecido como *Alexa Voice Service*, pode ser utilizado por qualquer dispositivo que contenha microfone e alto falante, e que consiga conectar-se a ele pela Internet.

O *Google Home*² é similar ao *Amazon Echo* em alguns aspectos, sendo também um *smart speaker*, que surgiu como expansão do aplicativo para *smartphones* Google Now, um assistente pessoal. Atualmente existe também como aplicativo para *smartphones*. Não é possível o desenvolvimento de módulos e expansões ao *Google Home* por desenvolvedores desvinculados à Google, porém ela trabalha diretamente com outras marcas e produtos para o estabelecimento de parcerias, de forma que o *Google Home* também

¹<http://www.amazon.com/oc/echo/>

²<https://madeby.google.com/home/>

consiga funcionar como *hub* de automação residencial.

2.1.2 Sistemas Open Source

Também existem diversos projetos *open source* sobre o tema, cujas documentações estão disponíveis publicamente na Internet. Alguns desses projetos, analisados para o desenvolvimento do Hedwig, foram o OpenHAB e o Home Assistant.

O OpenHAB³ possui como objetivo principal o estabelecimento de uma plataforma de integração, capaz de solucionar o problema atual de haver diversos dispositivos em uma residência que não são capazes de se comunicar, devido à falta de uma linguagem comum para a troca informações. Por ser independente de hardware específico, é extremamente flexível e personalizável, porém isso implica em maior complexidade para o usuário no momento de sua instalação. O OpenHAB apresenta interface para o usuário em cliente web e aplicativos nativos para iOS e Android.

O Home Assistant⁴ é uma plataforma de automação residencial capaz de controlar e monitorar os diversos dispositivos em uma casa, oferecendo uma plataforma web para o controle do sistema pelo usuário. O controlador local é implementado em Python, e recomenda-se instalá-lo em um Raspberry Pi. Possui diversas integrações já estabelecidas, com sistemas e serviços como o próprio Amazon Echo, Google Cast, IFTTT, Digital Ocean, entre outros, mas possibilita também a criação de novos componentes pelos próprios usuários. A personalização pelos usuários é feita por meio de um arquivo de configuração no formato YAML.

Os dois projetos apresentam como maior dificuldade a necessidade do usuário possuir conhecimentos técnicos para utilizá-los.

2.2 Projeto HomeSky

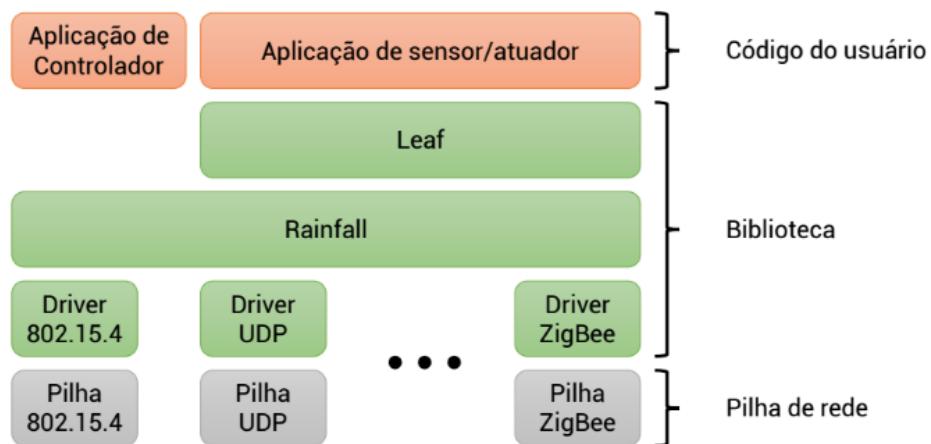
O Projeto HomeSky (MURAMATSU; RODRIGUES; GALLEG, 2016) é um Trabalho de Conclusão de Curso desenvolvido por alunos de Engenharia de Computação na Escola Politécnica da Universidade de São Paulo. Com o objetivo de fomentar iniciativas de desenvolvimento na área de casas inteligentes, o trabalho focou-se na criação do Rainfall, um protocolo em código aberto a nível de aplicação para ser usado na coordenação de uma rede de sensores. Isso permitiria aos desenvolvedores ter uma maior flexibili-

³<http://www.openhab.org/>

⁴<https://github.com/home-assistant/home-assistant>

dade em seus projetos, visto que muitas das soluções existentes são proprietárias. Por fim, também foi realizada a implementação de um algoritmo de aprendizado de máquina capaz de controlar a iluminação.

Figura 3: Camadas da arquitetura usada no Projeto HomeSky. As camadas em verde correspondem às bibliotecas desenvolvidas no trabalho.



Fonte: (MURAMATSU; RODRIGUES; GALLEGO, 2016)

No desenvolvimento do protocolo Rainfall, foram consideradas algumas hipóteses simplificadoras a respeito da conectividade e da segurança. O protocolo não trata de forma especial a fase de conexão à rede, considerando que todos os nós já estão conectados a ela, e também considera que todos os protocolos adjacentes são confiáveis, delegando as implementações de mecanismos de reconhecimento de entrega e retransmissão ao desenvolvedor. Quanto à segurança, assume-se que a infraestrutura seja segura e que nenhum nó conectado à rede tenha comportamento mal intencionado, como por exemplo espionar mensagens destinadas a outros nós ou fingir ser o controlador.

3 Especificação

3.1 Stakeholders

Um dos passos iniciais na elaboração de um projeto é a determinação das partes interessadas. Com esse conhecimento, pode-se entender as necessidades dos diferentes perfis de clientes e as expectativas desses grupos em relação ao uso do produto. Por meio da análise de aplicação, tanto do projeto desenvolvido quanto dos dispositivos existentes, pode-se destacar os seguintes grupos dentre os potenciais consumidores:

- Pessoas que moram sozinhas e seus familiares;
- Pessoas que buscam comodidade no uso e controle de dispositivos domésticos;
- Pessoas preocupadas com o consumo de água e energia elétrica.

Considerando o Censo de 2010 (IBGE, 2010), pode-se estimar as classes de consumidores para a cidade de São Paulo:

- Considerando que 1/10 da população com mais de 60 anos more sozinha e que 1/4 deles adquiriria o produto, temos uma estimativa de 33 mil consumidores. Como essa população está envelhecendo em taxas cada vez maiores (8,96% em 2000 contra 13,6% em 2016) (BIBLIOTECA VIRTUAL, 2017), a tendência é que essa classe aumente;
- Considerando que 1/100 dos domicílios ocupados tenha uma pessoa com esse perfil, temos uma estimativa de 35 mil consumidores em potencial;
- Considerando que cerca de 70% das residências reduziram o consumo com campanhas de redução de uso de água em 2015 (G1, 2015), supondo que 5% ficariam preocupados/interessados ao nível de se tornarem consumidores, temos uma estimativa de 71 mil consumidores em potencial.

3.2 Requisitos

O levantamento de requisitos funcionais e não funcionais são definidos a seguir. Para requisitos funcionais, utilizou-se o termo *RF* seguido de um número identificador — e.g. *RF-1*. Os requisitos não-funcionais levam o termo *RNF*, também seguido de um identificador numérico — e.g. *RNF-1*.

3.2.1 Requisitos Funcionais

RF-1: O sistema deve permitir o monitoramento de aparelhos do dia a dia, dentro de uma residência, de maneira independente;

RF-2: O sistema deve ser capaz de notificar o usuário sobre o estado da casa — e.g. temperatura, presença, etc;

RF-3: O sistema deve poder ser personalizável pelo usuário, o qual pode adicionar ou remover funcionalidades;

RF-4: O sistema deve permitir o acionamento de dispositivos da residência;

RF-5: O sistema deve ser capaz de aprender a respeito de cada usuário, podendo detectar padrões no seu comportamento e, a partir disso, sugerir ações a serem tomadas automaticamente — e.g. acendimento automático de lâmpadas;

RF-6: O sistema deve permitir o cadastro, remoção e atualização de usuários;

RF-7: O sistema deve permitir o cadastro, remoção e atualização de casas, para cada usuário;

RF-8: O sistema deve permitir o cadastro, remoção e atualização de dispositivos, para cada casa;

RF-9: O sistema deve permitir o seu reinício (*reset*);

RF-10: O sistema deve permitir a autenticação de usuários;

RF-11: O sistema deve permitir a configuração de dispositivos;

RF-12: O sistema deve disponibilizar uma *API* para comunicação.

3.2.2 Requisitos Não-Funcionais

O levantamento de requisitos não-funcionais foi realizado com base na norma ISO25010:2011 (ISO/IEC, 2011).

RNF-1: Os dispositivos que compõem o sistema dentro de uma residência devem ser independentes entre si, devendo obedecer a uma interface comum de integração com o *core* do projeto, para que seja facilitada a sua ampliação e extensão, com outras funcionalidades;

RNF-2: O sistema deve garantir segurança dos dados por meio de protocolos de comunicação seguros. O tráfego de informação entre as residências e os servidores externos não pode ser realizado em texto aberto. A comunicação interna da casa deve ser realizada em canais restritos;

RNF-3: Os dados de cadastro do sistema devem ser armazenados em servidores seguros (MCGRAW, 2004);

RNF-4: O sistema deve ser robusto, de modo a continuar operando, mesmo com menor nível de funcionalidades, quando há ocorrência de falhas na comunicação entre a casa e serviços externos — e.g. indisponibilidade parcial devido a problemas com servidores remotos, indisponibilidade total devido a perda da conexão com a Internet — ou falhas na rede interna, como a indisponibilidade de conexão local. A validação é realizada com interrupções na rede, desconexão e desligamentos de servidores e dispositivos, seguida da verificação dos serviços disponíveis;

RNF-5: O sistema identificar e se recuperar recuperar de travamentos em suas partes, reiniciando-as;

RNF-6: O sistema deve possuir mecanismos de proteção contra ataques de negação de serviço (*DoS*);

RNF-7: O sistema deve apresentar disponibilidade de 99,9% — cerca de 8 horas de indisponibilidade por ano. Essa medida de disponibilidade refere-se somente aos serviços remotos, não levando em consideração indisponibilidades das residências;

RNF-8: O sistema deve ser escalável a até 10 mil usuários, sem perdas de desempenho consideráveis, ou aumento na latência para as requisições serem atendidas, com

variação de até 5%. A verificação deve ser realizadas com softwares de análise de performance comerciais¹;

RNF-9: O sistema deve possuir instalação intuitiva e simplificada — a instalação é realizada em passos, seguindo o manual de instruções, sem a necessidade de conhecimentos de computação ou eletrônica;

RNF-10: O sistema deve atender e processar requisições em paralelo, tanto dos usuários quanto dos dispositivos físicos;

RNF-11: O sistema deve autenticar e autorizar as requisições recebidas, descartando as que invalidas.

3.2.3 Funcionalidades por Nível de Conectividade

Os requisitos apresentados anteriormente detalham o sistema completo. Para que o requisito *RNF-4* fosse implementado, o projeto incorporou três níveis de funcionalidade — *Online*, *Local* e *Offline*. O primeiro nível, *Online*, é o mais amplo e caracteriza o comportamento normal do sistema, quando todas as conexões estão disponíveis, e os dispositivos e serviços funcionam corretamente. O nível *Local* modela o cenário de não haver possibilidade de conexão externa com a casa, como é o caso quando não há internet disponível. O último nível, *Offline*, reflete uma situação emergencial, no caso de problemas com o servidor interno, por exemplo.

¹Um exemplo de aplicação de monitoramento de rede é desenvolvida pela empresa Dynatrace (<https://www.dynatrace.com/>)

4 Arquitetura

4.1 Visão geral

O projeto da arquitetura foi realizado com atenção aos requisitos levantados. O requisito *RNF-4* desempenhou papel fundamental nas escolhas e implementações, bem como o requisito *RNF-1*, que atenta à independência das partes e a obediência à contratos (interfaces). Com base no requisito *RNF-1*, optou-se por um projeto modular, onde os dispositivos físicos fossem agrupados, de acordo com a sua aplicação, em módulos — os quais podem ser adquiridos, instalados e aperfeiçoados independentemente.

Nas próximas seções, serão analisados os modelos de arquitetura propostos, revelando o processo de escolha com base nas vantagens e problemas de cada modelo, até a chegada da versão final, implementada.

4.2 Evolução arquitetural

O processo de definição da arquitetura foi iterativo, em um método de estabelecimento do modelo, validação e adequação. Para cada modelo foram analisadas as suas vantagens no cumprimento dos requisitos, bem como os seus pontos fracos, até a definição da arquitetura a ser implementada.

A primeira versão proposta baseava-se unicamente em microsserviços, responsáveis por toda a inteligência do projeto, o que a fazia interessante do ponto de vista da escalabilidade para um número muito grande de casas. Com uma arquitetura fundamentada dessa maneira, é possível a utilização transparente de quantas tecnologias forem necessárias ou desejáveis, para cada um dos serviços, sem efeitos colaterais ou impactos em outros serviços. Por outro lado, cria-se grande uma complexidade na integração entre os serviços disponíveis. A complexidade pode ser gerenciada por técnicas já consolidadas, como a coreografia e a orquestração (LEWIS; FOWLER, 2014).

Com o crescimento no número de microsserviços, o *overhead* para a comunicação é aumentado, visto que é cada requisição necessita de um amplo número de serviços solicitados, para que possa completar a sua tarefa. Há também uso mais intenso da infraestrutura de comunicação, já que os serviços operam por troca de mensagens, as quais sofrerão aumento proporcional ao número de chamadas. As requisições aos microsserviços devem ser autenticadas e autorizadas, conforme o requisito *RNF-11*, de modo que foi proposto um *gateway* para os serviços da nuvem, por onde passaria todas as requisições válidas, no fluxo de comunicação com a casa. A inserção do *gateway* cria um ponto único de falha, mas que pode ser evitado com técnicas de redundância e duplicação (SAVIT, 2013).

Figura 4: Primeira versão da arquitetura do projeto Hedwig



É possível observar que alguns microsserviços são classificados como sensitivos, os quais dependem de nova consulta ao serviço de autenticação e autorização para garantir a segurança. Esses serviços são todos aqueles responsáveis por tomar uma ação em relação à casa que envolva riscos, como a abertura de portões. Os microsserviços não-sensitivos utilizam a autenticação já realizada pelo *gateway* na chegada da requisição.

Quando uma requisição chega à nuvem, ela deve ser validada, para garantir a sua origem (*RNF-11*), e o método adotado faz uso de *tokens*. Caso passe nos critérios de autenticação e autorização, é retornado um *JWT* (*JSON Web Token*), necessário para os passos seguintes. O *JWT* é discutido na seção 4.6.2.

De extrema importância, e não cobertos pela arquitetura anterior, são os requisitos de disponibilidade do projeto (*RNF-4*). Se o *gateway* estiver inacessível em determinado momento, a casa não terá mais nenhuma forma de comunicação com os meios externos,

mesmo para os serviços mais básicos. Para resolver este problema, foi proposta uma segunda versão, conforme ilustra a imagem seguinte.

Figura 5: Segunda versão da arquitetura do projeto Hedwig



Nesta versão, serviços essenciais seriam duplicados dentro da casa e, no caso de haver qualquer forma de impedimento na comunicação com a nuvem, esses serviços seriam responsáveis por controlar diretamente os atuadores desejados. Entretanto, cria-se mais uma complexidade ao manter serviços duplicados na casa, e no caso destes serviços também não estarem *online* no momento necessário, também não seriam alcançados requisitos de disponibilidade. Contudo, é uma versão que chega mais próxima de obedecer às necessidades do projeto.

Essa arquitetura provê módulos sem inteligência, e todo o controle é feito pelo serviço correspondente. Ao mesmo tempo que essa escolha desfruta de benefícios como a escalabilidade, a manutenção (já que é extremamente mais simples atualizar o software nos servidores do que nas residências) e a facilidade para prover correções ou possíveis aumentos de funcionalidade. Outro ponto é que alguns módulos ficariam em lugares de difícil acesso, ou mesmo fora da casa, onde a comunicação poderia ser intermitente, ou mesmo perdida. Assim, em caso de falha de comunicação, um atuador não receberia os sinais necessários do serviço, acarretando em sérios problemas na proteção da casa. No caso de uma garagem, por exemplo, o portão permaneceria aberto indeterminadamente, ou poderia não ser aberto quando o morador chegasse em casa.

Assim, avançou-se para o desenvolvimento de um modelo arquitetural modularizado, onde cada módulo teria inteligência para realizar as tarefas necessárias e, ao mesmo tempo, pode enviar dados à nuvem, e ser avisado quando for necessário realizar uma tarefa. Além disso, no aspecto comercial, módulos inteiros poderiam ser vendidos, substituídos e aumentados.

A arquitetura projetada faz uso de microsserviços no lado da nuvem e, no lado da casa, os componentes de hardware passam a ser agrupados em módulos independentes, com responsabilidades bem estabelecidas (*RNF-1*), inteligência e autonomia para realizar todas as atividades necessárias. Os módulos se comunicarão com um servidor local, que realizará, por último, a conexão direta com os serviços não locais. Esse servidor se comunicaria com os módulos por meio de mensagens enviadas em tópicos, as quais seriam interpretadas e enviadas aos servidores remotos em canais protegidos (*RNF-2*). O requisito *RNF-8*, relativo à escalabilidade, não será verificado no projeto, e pode ser considerado em passos futuros.

Em uma eventualidade, a comunicação entre o servidor local e a nuvem pode ser perdida. O usuário, no entanto, deve ainda conseguir se comunicar com a casa, ainda que tenha ao seu dispor uma quantidade mais restrita e essencial de ações — como a liberação de acesso à casa. Quando é perdida a conexão entre a casa e os serviços externos, o servidor local armazena as mensagens vindas dos módulos, que serão transmitidas ao servidor remoto posteriormente. Como não há urgência para o processamento de tais dados (visto que não são requisições de ações, mas comunicação de estado), os quais serão utilizados para análise de comportamento e Machine Learning (*RF-5*), não há prejuízo com o eventual envio tardio. O usuário poderá acompanhar o estado da casa, com as informações vindas dos sensores, por meio de um segundo aplicativo, denominado aplicativo *backup*.

No caso mais extremo, de perda de comunicação tanto com a nuvem quanto com o servidor local, como na ocorrência de falha de hardware, os aplicativos poderão se comunicar diretamente com os módulos para terem acesso aos serviços de extrema importância.

4.3 Criação de Módulos

Para a criação dos módulos de hardware, foram escolhidos componentes de *IoT* comerciais, que possuem preços acessíveis, ampla documentação disponível e uma comunidade de desenvolvedores crescente.

A interconexão dos componentes, bem como a comunicação com o mundo externo pela

internet será intermediada por um servidor local, instalado e disponível na plataforma *Raspberry Pi*, rodando um sistema operacional Linux (*Raspbian*, baseado em *Debian*) e que dispõe da interface de hardware necessária para conexão com a rede.

Os sensores e atuadores devem ser conectados fisicamente com um módulo controlador, e para que essa limitação fosse contornada, foram utilizados dispositivos *ESP8266* — subseção 4.3.1.1 — para transmissão sem fio por meio de *Wi-Fi*. Esses módulos serão responsáveis pela transmissão das informações recebidas para o servidor local. Toda a arquitetura para essa transmissão será detalhada mais à frente. Os outros dispositivos a serem utilizados, como sensores *DHT11*, *LM555*, etc. podem ser vistos em uma lista completa no Anexo B.

Em geral, os módulos consistem do microcontrolador, relés, sensores e fontes/conversores de tensão a depender do módulo, além de um circuito para manutenção corretiva baseado no astável 555, conectados à rede *Wi-Fi* ou trabalhando como pontos de acesso. Para casos de falha de conexão, há um algoritmo de novas tentativas com tempos progressivamente maiores conforme as falhas ocorrerem, que busca deixar o módulo disponível para outras funções enquanto o serviço não está disponível. Para mitigar o travamento, um sinal de *keep alive* é monitorado, e um circuito anti-travamento deve ativar o *hard reset* (reset por hardware), ou então uma rotina de *soft reset* deve ser acionada, de modo que os requisitos *RNF-5* e *RF-9* sejam cumpridos. No entanto, observa-se que a segunda alternativa é a mais natural de se implementar, mas menos robusta, já que ainda pode não funcionar em casos de loop infinito.

4.3.1 Módulos Base

4.3.1.1 ESP8266

O *ESP8266* é um microprocessador com baixo consumo e radiotransmisor com conexão *Wi-Fi 802.11* integrada (ESPRESSIF, 2015). Pode ser programado usando a *Arduino IDE*, vastamente utilizada (THOMSEN, 2016). Opera com uma tensão de 3.3 V, suporta *WPA* e possui modo de interrupção somente por software. É amplamente usado como *shield* para conexão *Wi-Fi* de placas de desenvolvimento da plataforma Arduino. Contudo, no projeto Hedwig, o dispositivo será utilizado em modo *StandAlone* como principal processador e responsável pela conexão dos diferentes módulos de automação. Suas duas principais plataformas de desenvolvimento são *Wemos*¹ e *NodeMCU*². O projeto

¹<https://www.wemos.cc/>

²<http://nodemcu.com/>

utilizará o *Wemos D1 Mini*, versão compacta da *Wemos D1 R2*.

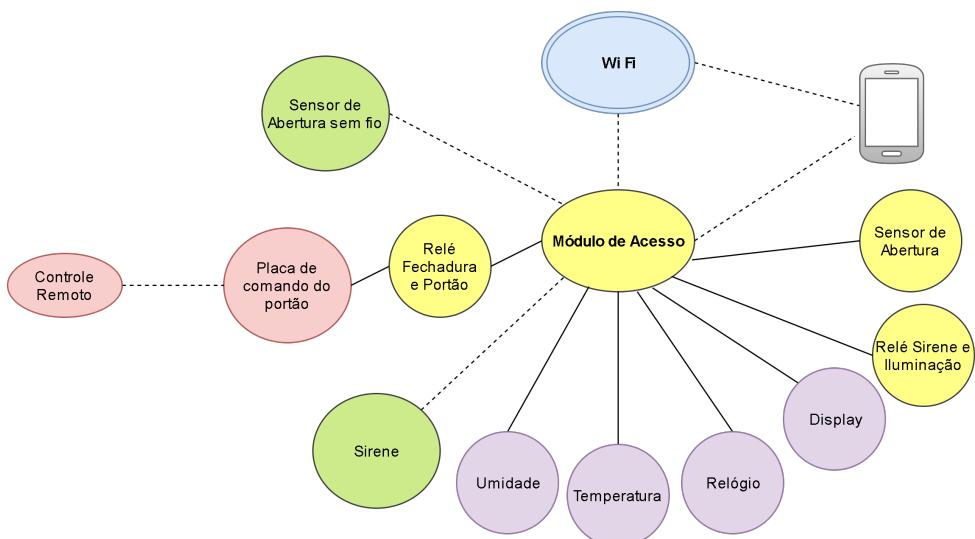
O *ESP8266* possui um modo de operação de baixa potência (*sleep mode*) em que o consumo de bateria fica muito menor — em contrapartida, o número de funcionalidades é limitado. Pode-se utilizar 7 portas de E/S digitais e uma porta de entrada analógica. Duas portas não são acessíveis, pois são utilizadas para programação e outras tarefas do sistema integrado do *ESP8266*. Alternativas para extensão de portas são:

1. Utilização três níveis de sinal análogo para detectar três tipos de acionamento, através de um circuito dedicado, com priorização de entrada;
2. Utilização interface I2C, como o usado para o display;
3. Utilização de radiofrequênci, por meio de um par receptor-transmissor integrado no módulo, controles, atuadores e sensores sem fio.

4.3.2 Módulo de Acesso

Buscando garantir mais segurança e comodidade para o acesso à residência, além do controle de abertura, o módulo de acesso atua em paralelo com uma fechadura eletrônica acionada por meio de controle remoto, que utiliza ondas de rádio para envio de dados. Assim, mesmo com falha total do sistema, o usuário poderá abrir o portão diretamente, sem a necessidade de acesso à Internet.

Figura 6: Diagrama ilustrativo do módulo de Acesso ao Portão



A Figura 6 ilustra, em vermelho, dispositivos já existentes no mercado, como o controle remoto. O sensor e a sirene sem fio adicionais são mostrados em verde — dispositivos

externos ao módulo, que se comunicam por ondas de rádio. O próprio módulo de acesso, com um buzzer embutido, e sua conexão com a rede local Wi-Fi ou sua conexão direta com o celular (quando o módulo opera como um ponto de acesso de rede), em amarelo. As funcionalidades adicionais são marcadas em roxo.

A comodidade, no exemplo em questão, está em abrir o portão por meio do celular, ao utilizar o aplicativo web ou o aplicativo local (de emergência), sem a necessidade de carregar uma chave ou controle.

Entretanto, é necessário que a realização do controle de acesso seja feita de maneira segura. Assim, é empregado um algoritmo de rotação de teclas, para evitar que pessoas mal intencionadas possam:

1. Olhar e copiar a senha que o usuário digita em seu celular;
2. Copiar os dados de abertura e usá-los mais tarde (*middle man*).

Na alternativa alternativa 2, a cada acesso, um novo mapeamento de teclas é gerado e enviado ao usuário. Mesmo que haja cópia das credenciais, ela não funcionará devido ao mapeamento ter mudado. Observe ainda que a fechadura eletrônica, por si só, já estava vulnerável a este tipo de ataque — há, inclusive, dispositivos copiadores de senhas comercializados.

Outro aspecto de segurança é a preocupação dos usuários em esquecer a porta ou portão abertos. Para mitigar esse perigo, o módulo deve monitorar, por meio de um sensor, o estado vigente (aberto/fechado), conforme o requisito *RF-1*, e alertar localmente (por meio de *buzzer*) e remotamente (e.g. por email ou notificação no *smartphone*) o usuário, conforme o requisito *RF-2*. Essa e outras configurações (como de rede) são acessadas por uma senha diferente daquela de abertura, de modo que a interface básica seja simples para uso.

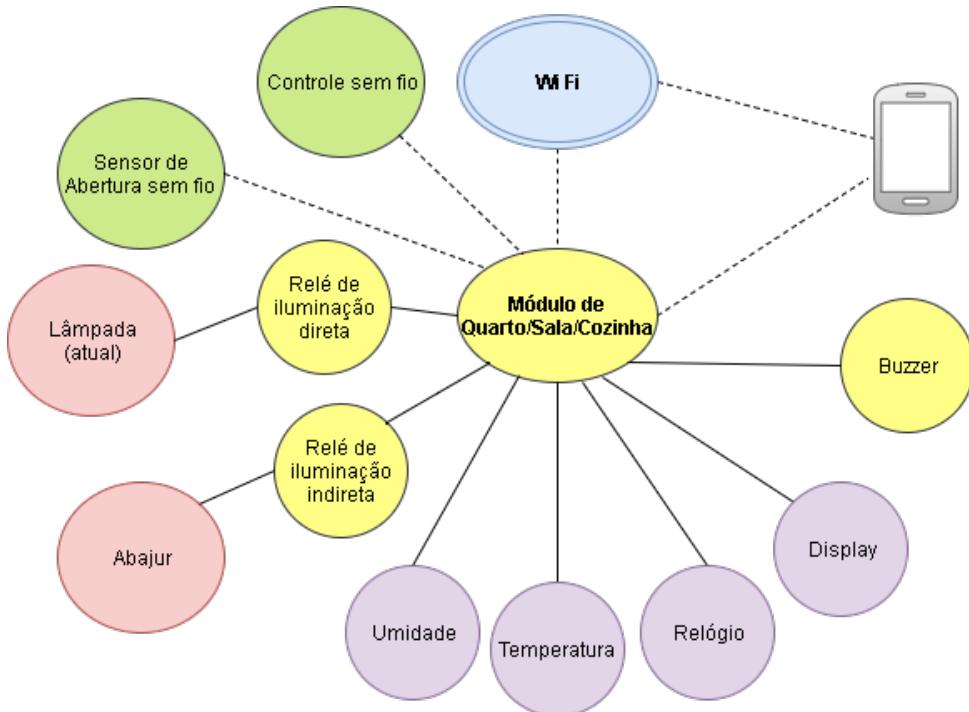
Para o caso de falha de envio de notificação (e.g. servidor fora do ar, ou indisponibilidade na conexão), há um algoritmo de novas tentativas com tempos progressivamente maiores conforme as falhas ocorrerem, buscando deixar o módulo disponível para outras funções. Tratamento análogo é realizado no servidor local, e no sistema de mensageria, de modo a evitar perdas de mensagens, mesmo em situações desfavoráveis. Para o caso de falta de conexão à internet, o módulo não seria controlável pela nuvem, com o aplicativo web, mas sim com o aplicativo emergencial, com a ativação do *Access Point*, desenvolvido para operar diretamente com os módulos, sem intermédio do servidor local e dos serviços remotos.

Por meio das credenciais disponíveis no sistema, é possível saber qual dos usuários que solicitou a abertura do portão. A persistência destes acessos pode ser analisada e, utilizando-se técnicas de *Machine Learning*, perfis de acesso podem ser determinados, e evoluir até o sistema saber quando houver um acesso em horário inesperado e notificar o usuário remotamente, conforme o requisito *RF-5*. O aprendizado de máquina é fundamental aqui para descobrir comportamentos que podem ser entendidos como suspeitos. Um exemplo prático de caso de uso seria um usuário que costuma chegar em um horário semelhante todos os dias, e realizar certo conjunto de tarefas na casa. Uma tentativa de acesso que não se enquadre em tais padrões pode ser produto de atividade suspeita, a qual pode ser informada pela casa para uma central, que acionaria a polícia caso não seja um falso positivo.

4.3.3 Módulo de Quarto/Sala/Cozinha

Um dos módulos com muitas opções de implementação e uso é o módulo de quarto, pois também pode ser usado no controle de iluminação para corredores, salas e ambientes externos.

Figura 7: Diagrama ilustrativo do módulo de Quarto/Sala/Cozinha



O diagrama ilustra equipamentos externos ao sistema em vermelho (lâmpada e abajur), enquanto o controle e sensor de abertura possuem comunicação sem fio. Em roxo, representa-se os equipamentos opcionais.

Como principais funcionalidades, tem-se o despertador (configurado pelo usuário, que também pode receber recomendações baseadas na informação gerada pelo monitoramento de seus ciclos de sono); monitoramento de temperatura e umidade do ambiente (que podem ser notificadas ao usuário, caso informem valores fora de determinados intervalos); controle de iluminação (da luz direta, que é a lâmpada central do ambiente, com maior potência, e da luz indireta, que é usualmente um abajur ou uma lâmpada com menor potência, usada para leitura); e estado da janela, para verificar remotamente se a janela está fechada ou não (por notificação ou visualização no aplicativo, útil para dias chuvosos). Além disso, se o módulo for instalado em ambientes internos e externos, o usuário pode usufruir de dados de temperatura e umidade, que podem ser usados para escolha de vestimenta, uso de guarda-chuva na ída para o trabalho ou se é mais vantajoso deixar roupas secando dentro ou fora de casa, e em que períodos.

O módulo de quarto pode ser acoplado ao sistema existente (fisicamente, é instalado no mesmo lugar do interruptor), e possui estados para o despertador. No estado inicial, somente a luz indireta é ligada. Após determinado tempo (programável pelo usuário), há avisos sonoros periódicos. No terceiro estado, os períodos são menores. Finalmente, no quarto estado a luz direta é ligada e os avisos sonoros são ininterruptos. Até o terceiro estado, o alarme pode ser desarmado (apertar duas vezes) ou entrar em estado soneca (apertar única vez) diretamente no módulo. Já no estado 4, a critério anterior do usuário, o alarme pode ser desarmado somente fisicamente em outro módulo presente em um segundo aposento — por exemplo, na sala. Esse módulo pode variar de dia para dia, caso o usuário assim desejar. O sistema desarma o alarme após 40 minutos.

O display possui iluminação automática para não apresentar brilho muito intenso quando todas as luzes estiverem desligadas (por meio de circuito baseado em *LDR*). Para o controle da iluminação, há diferentes tempos para desligamento. Por exemplo, quando ocorre controle manual (pelo botão presente no módulo), o tempo pode ser maior. Já pelo modo automático, quando a luz já foi ligada pelo próprio módulo, o tempo para desligamento pode ser menor (por exemplo, 4 minutos).

Com o monitoramento da presença, há um *preset* da contagem para desligamento sempre que houver presença detectada, de forma a inibir acionamentos desnecessários do relé. Outra aplicação para o monitoramento da presença é a descoberta de comportamento anormal. Por exemplo, se o usuário sempre toma café entre 8 e 10 horas, e não apresentar presença na casa até às 15 horas, o sistema pode notificar emails de parentes cadastrados.

4.3.4 Módulo de Aquário

Devido a altos custos de compra, implantação e manutenção de um aquário, que pode ser de água doce ou salgada, e até abrigar espécies raras, é desejável que uma série de riscos sejam mitigados. Dentre tais riscos, destacam-se:

Tabela 1: Riscos para o Aquário

Perigo/Necessidade	Origem	Consequência
Aquecimento accidental	Ajuste errado da temperatura do termostato	Superaquecimento; risco de mortes (peixes e plantas)
Falta de água	Vazamento ou evaporação natural	Mal funcionamento ou queima da bomba submersa (à longo prazo, falta de oxigenação da água)
Falta de circulação de água	Entupimento do tubo de circulação ou mal funcionamento da bomba	Falta de oxigenação da água, ocasionando em risco de mortes (peixes)
Iluminação adequada	Existência de plantas e/ou iluminação natural insuficiente no ambiente do aquário	Ambiente nocivo para os peixes, risco de morte das plantas (principalmente durante períodos de esquecimento/viagens)

Sobre o caso específico da residência onde foram executados os testes de campo, considere um aquário com 50 litros de água, com uma bomba reserva (não a responsável pela circulação de água, que não muda o volume interno) e um aquecedor, com muitos peixes dentro, que são sensíveis à variação brusca de temperatura.

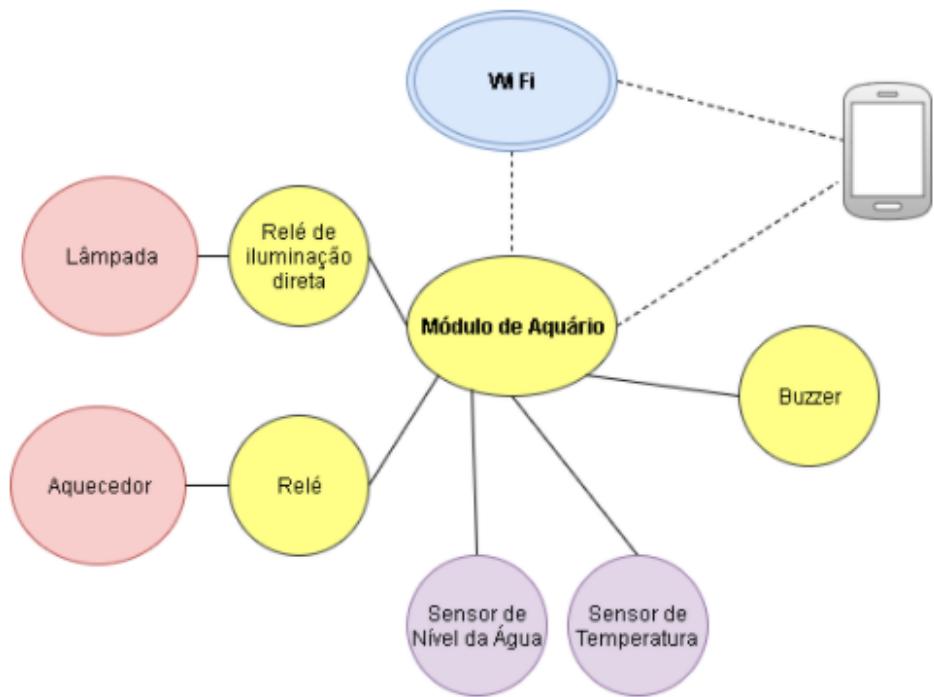
Figura 8: Aquário de Santo André



Na ocasião de troca de água do aquário, cerca de 20% do volume total é substituído. A saída de água se dá por um funil (simplificado, para se obter uma vazão de saída constante), e a adição é realizada pela bomba auxiliar, que possui água limpa, sem cloro e com certeza menos amônia e outros compostos nocivos aos peixes (que justificam essa

troca periódica de água). O monitoramento do nível da água pode também mitigar o risco de esquecimento — o que levaria o nível da água a tender a zero.

Figura 9: Diagrama ilustrativo do módulo de Aquário



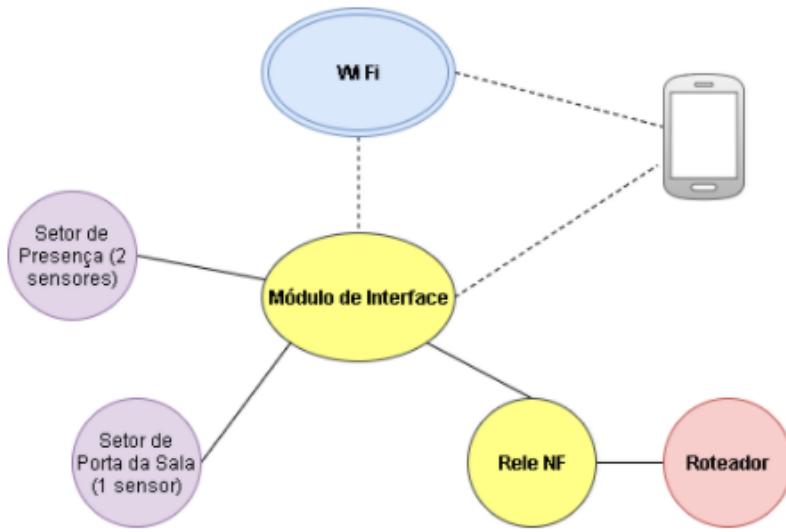
Para mitigar os riscos descritos anteriormente, foi desenvolvido o módulo de aquário, que permite:

1. Controle de horários em que a lâmpada fica acesa, fornecendo uma iluminação adequada para as plantas e peixes do aquário em períodos curtos de viagem;
2. Monitoramento do nível de água do aquário principal, alerta pelo aplicativo quando não estiver no nível esperado (pode ocorrer por muita evaporação, vazamento ou problema com a bomba);
3. Monitoramento da temperatura do aquário, e bloqueamento do aquecedor caso a água já esteja numa temperatura desejável (evitar superaquecimento devido a mal funcionamento do termostato), feito por meio de ligação em série com o termostato do aquário;
4. Nos casos de perigo acima descritos, alerta sonoro também localmente (por meio do buzzer).

4.3.5 Módulo de Interface com Sistema de Alarmes

O módulo de interface com o sistema de alarmes monitora dois setores específicos, um relativo a presença, que possui sensores no corredor e na sala da residência, e outro relativo à porta da sala (sensor único na porta da residência). O módulo implementado foi instalado em uma residência em Jarinu - SP.

Figura 10: Diagrama ilustrativo do módulo de Interface com Sistema de Alarmes



Um problema recorrente é a conexão com a internet, que apesar da indicação de estado válido e conectado, não fornecia acesso à sites, tampouco acesso externo por meio de abertura de porta no roteador. Para contornar esse problema, e permitir que o módulo esteja disponível para coleta de dados e persistência em cartão SD, foi instalado também um relê *NF* (normalmente fechado) em série com a alimentação do roteador. Em caso do módulo estar desligado, o roteador ficará ligado.

Já quando há erro persistente (maior que 10 vezes em intervalos de 2 a 3 minutos), ao realizar a operação de *ping* com sites ou servidores conhecidos, o módulo reinicia a conexão, atuando diretamente no roteador. Ocorre a execução deste procedimento em intervalos cada vez mais espaçados, de forma a não executar muitas vezes o reinício do roteador sem que a conexão seja reestabelecida com sucesso (nas primeiras tentativas, tem-se um intervalo de 3 minutos até a nova tentativa, a partir da terceira vez um intervalo de 10 minutos, e assim por diante).

Outra dificuldade encontrada em sua instalação em campo foi o fato de obtenção de endereço IP dinâmico (onde outros dispositivos, tais como celulares e tablets também obtinham endereços IP dinâmicos), gerando indisponibilidades do módulo. Com a mudança

do endereço IP para fixo, em outro intervalo de endereçamento, o módulo passou a ter alta disponibilidade, ficando até semanas sem reiniciar.

4.4 Controlador Local

Para a intercomunicação entre os módulos e a nuvem, há a presença do servidor local, Morpheus, responsável por introduzir mais uma camada de segurança na troca de mensagens. Para isso, foi desenvolvida uma plataforma, com a utilização de sistemas de mensageria, e definido um protocolo de comunicação entre os serviços de nuvem e os módulos. Assim, quando um usuário realiza determinada operação por meio do cliente web, uma mensagem é enviada, interpretada pelo servidor local e, em seguida, encaminhada para o destino por meio do protocolo *MQTT* com o broker Mosquitto. O Morpheus é visto em detalhe na Seção 6.1.

4.4.1 Raspberry Pi

O *Raspberry Pi* é um computador integrado em um único chip, do tamanho de um cartão de crédito. Foi desenvolvido com o objetivo de promover o ensino de computação básica, e possui funcionalidades tais como um Computador Pessoal (PC): navegação na Internet, reprodução de vídeo, processamento de texto, dentre outros. No projeto, será utilizado como servidor local (gerenciador de módulos local da casa), exatamente pelas funcionalidades compatíveis com a de um computador desktop.

A versão 3 possui uma CPU 1.2 Ghz 64-bit quad-core ARMv8, conexão 802.11n Wireless LAN, Bluetooth 4.1, suporte a Bluetooth Low Energy (BLE), 1GB RAM, 4 portas USB, 40 pinos GPIO, porta HDMI, porta Ethernet, interface para câmera, display e cartão SD. Para projetos que necessitem de baixo consumo energético, os modelos mais indicados são Pi Zero ou A+ (RASPBERRY PI, 2017).

Figura 11: Raspberry Pi 3 Modelo B



Fonte: (RASPBERRY PI, 2017)

4.5 Servidor na nuvem

O servidor na nuvem tem a responsabilidade de realizar a comunicação entre as aplicações cliente disponíveis para o usuário final e as casas inteligentes, de armazenar os dados coletados pelos sensores e de realizar processamentos que sejam muito onerosos para a capacidade de processamento dos dispositivos físicos locais. A seguir, são explorados os conceitos que orientaram o design da arquitetura e a implementação do servidor do Hedwig.

4.5.1 Computação em nuvem

O projeto Hedwig opta por uma solução voltada à nuvem. Os componentes do servidor são hospedados na nuvem pelas seguintes vantagens (VISWANATHAN, 2017):

Custo - O investimento em servidores próprios geralmente possui um alto custo. Em contrapartida, os fornecedores de *Infrastructure as a Service* — *IaaS*, oferecem várias modalidades de precificação que vão desde assinaturas periódicas até pacotes que impõem limites de requisições.

Escalabilidade - Existe uma grande facilidade em escalar os recursos de forma rápida. Os serviços de nuvem permitem manipular características como capacidade de disco, tamanho de memória e tipo de processador das instâncias que rodam os programas e aplicações. Também é possível seguir o caminho da escalabilidade horizontal e simplesmente replicar instâncias ao invés de melhorar suas especificações técnicas.

Alocação de recursos eficiente - Com a especificação flexível e a facilidade em escalar, é possível aproveitar melhor a capacidade de processamento disponível e evitar desperdício com recursos ociosos.

Backup e recuperação de dados - Os serviços de nuvem já providenciam funcionalidades de backup e restauração de dados, que podem ser tarefas arduosas para realizar em dispositivos físicos.

4.5.2 Banco de dados não-relacional

Bancos de dados não-relacionais são modelados em forma alternativa às tabelas relacionais dos sistemas SQL. São muitas vezes chamados de bancos NoSQL, que adquiriu o significado de *Not Only SQL* (NOSQL, 2009). Algumas das características predominantes são a facilidade de escalabilidade horizontal, por meio de replicação e “clusterização”, e a priorização da disponibilidade ao invés da consistência. Esse último ponto pode ser sintetizado no conceito de BASE — *Basically Available, Soft state, Eventual consistency* —, que é colocado em contraposição às garantias popularmente oferecidas pelos bancos relacionais: Atomicidade, Consistência, Isolamento, Durabilidade (*Atomicity, Consistency, Isolation, Durability* - ACID). Apesar disso, alguns dos bancos NoSQL possuem características compatíveis com ACID. Esse tipo de banco de dados é bastante utilizado em aplicações web de tempo real e de Big Data (PEREIRA, 2016).

Os bancos de dados NoSQL podem usar vários esquemas para modelar os dados que armazenam: colunas, documentos, pares chave-valor, grafos ou uma mistura dos anteriores. Dentre eles, destacam-se os documentos, também chamados de dados semi-estruturados. Documentos são dados codificados em XML, JSON, YAML ou em outros formatos ou códigos, incluindo até mesmo binários. Geralmente, não seguem nenhum esquema rígido, o que torna o desenvolvimento mais flexível e facilita a incrementação dos modelos com novos dados.

4.5.3 Banco de dados em memória

Bancos de dados em memória usam a memória principal do computador para realizar o armazenamento de dados ao invés de dispositivos de armazenamento em disco (RAIMA, 2013). Como o tempo de acesso em disco é muito maior, esse tipo de banco de dados é capaz de atingir altos níveis de performance, proporcionando latências menores e mais previsíveis.

É um método de armazenamento mais caro (MULLINS, 2017), visto que a unidade de espaço em RAM é mais cara a de que disco. Assim, muitos projetos optam por uma arquitetura híbrida que usa tanto esse tipo de banco de dados quanto os tradicionais de acesso em disco. Devido ao fato da memória RAM ser volátil, esse tipo de abordagem também é usado para evitar perdas de dados em casos de falhas.

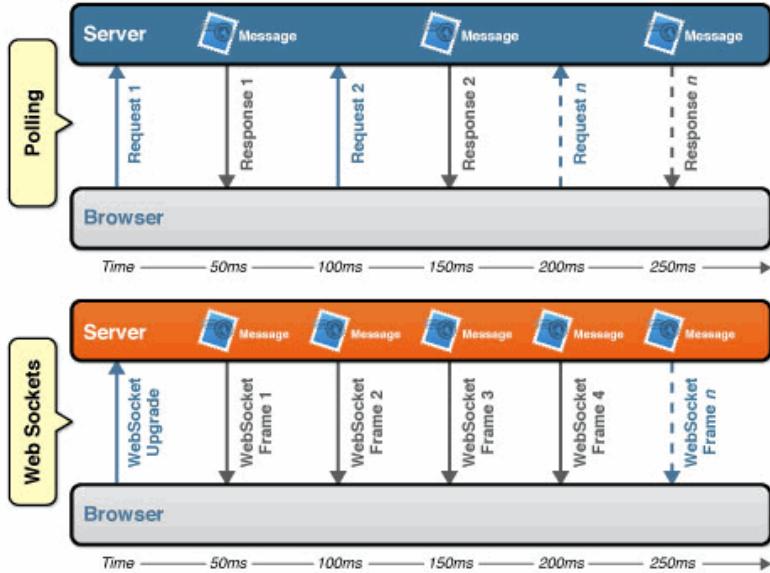
4.5.4 WebSockets

WebSocket é um protocolo de comunicação *full-duplex* sobre conexões TCP (IETF, 2011). Ele possibilita a comunicação interativa entre cliente e servidor sem a necessidade de disparar múltiplas requisições HTTP, permitindo também que o servidor envie conteúdo ao cliente sem que este tenha que requisitá-lo.

É um protocolo da camada de aplicação compatível com HTTP: a transição entre esses dois protocolos é feita por meio de um handshake que usa o cabeçalho de HTTP Upgrade, como observado na figura a seguir.

A sua arquitetura diminui as latências de comunicação, fazendo-o ser um protocolo popular entre aplicações de tempo real. A compatibilidade com HTTP permite que toda a troca de dados ocorra nas portas 80 ou 443, mitigando os problemas de incompatibilidade com ambientes que possuem firewalls bloqueando certas portas TCP. A maioria dos navegadores modernos implementa o protocolo de WebSockets, possibilitando o funcionamento de aplicações de chat e de notificações.

Figura 12: Comparação entre WebSockets e *polling*



Fonte: (LUBBERS; GRECO, 2016)

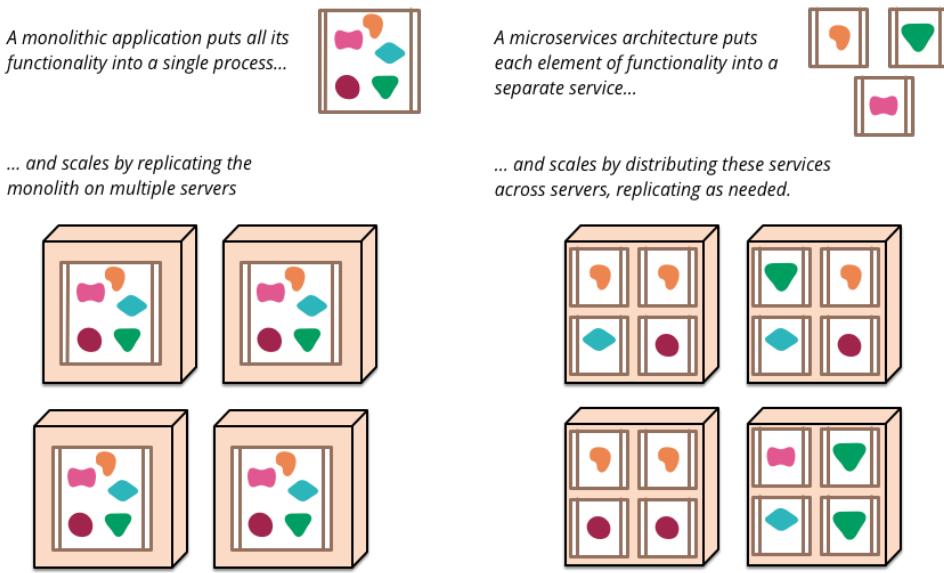
4.5.5 Arquitetura de Microsserviços

4.5.5.1 Características

A arquitetura de microsserviços é um modelo que comprehende a estruturação de uma aplicação em um conjunto de serviços com baixo grau de acoplamento que se comunicam por meio de protocolos de comunicação leves.

Para melhor compreender essa arquitetura, pode-se compará-la à arquitetura monolítica. Uma aplicação monolítica está contida em uma única unidade, que geralmente é dividida em camadas de funcionalidade tecnológica como interface web, camada de negócios *server-side* e camada de persistência de dados. A escalabilidade desse modelo é dada por meio do aumento do número de servidores, máquinas virtuais ou contêineres juntamente a um *load balancer* — é a chamada escalabilidade horizontal. Uma alteração em uma pequena parte da aplicação significa que toda a aplicação deverá passar por um processo de *build* e *deploy*. Já a arquitetura de microsserviços divide as funcionalidades em serviços autônomos, muitas vezes usando as regras de negócios para realizar essa divisão. Cada serviço tem seu próprio ciclo de desenvolvimento e pode ser atualizado independentemente. A escalabilidade também é tratada serviço a serviço.

Figura 13: Comparação entre uma aplicação monolítica (esquerda) e com microsserviços (direita)



Fonte: (LEWIS; FOWLER, 2014)

É difícil delimitar uma definição formal para arquitetura de microsserviços, pois não existe consenso a respeito de sua definição formal. Contudo, existe uma série de características que projetos usando essa arquitetura compartilham. Detalha-se a seguir alguns atributos e aspectos dos microsserviços. Nem todos os projetos possuem rigorosamente todas as características, mas a maioria deles possui um perfil similar ao descrito aqui.

Serviços são processos - Pode-se fazer um mapeamento de um processo para um serviço, porém isso é apenas uma aproximação, podendo um serviço ser constituído por uma aplicação de múltiplos processos;

Serviços comunicam-se por protocolos leves - Geralmente, são usados protocolos como o HTTP;

Serviços implementam capacidades do negócio - Isto é, a divisão de serviços é baseada nas regras de negócio e nas funcionalidades que o produto deverá suprir;

Serviços são facilmente substituíveis - Por serem pequenos e independentes;

Cada serviço tem um ciclo de vida independente - Isso inclui o desenvolvimento e os processos de *deploy*. Um microsserviço pode ser implementado e atualizado independentemente dos outros.

As vantagens da arquitetura de microsserviços giram em torno da modularidade e autonomia dos serviços que é natural à sua estrutura. Com isso, pode-se ter uma heterogeneidade de tecnologias, isto é, cada serviço pode ser desenvolvido usando diferentes linguagens, *frameworks* e ferramentas de acordo com seus requisitos. A independência entre serviços também possibilita o deploy automatizado e o uso de práticas de integração contínua. Também há benefícios de aspecto gerencial: como cada serviço tem como escopo uma capacidade do negócio que envolve interfaces de interação com usuário, código em várias camadas que implementa as funcionalidades necessárias e persistência em bancos de dados, é possível criar pequenas equipes multidisciplinares para cada microsserviço.

Existem trade-offs que devem ser considerados ao decidir pela arquitetura de microsserviços. A comunicação entre serviços por meio de uma rede possui maior latência e exige maior processamento do que mensagens trocadas a nível de processos. Por isso, é muito importante analisar as fronteiras dos serviços e a alocação de responsabilidades durante do projeto. A descentralização de dados entre microsserviços traz também a necessidade de métodos para manter a consistência das informações. Outro ponto crítico são sistemas com alta granularidade de microsserviços, causando overhead tanto de comunicação como de código além de uma fragmentação lógica que causa mais impactos negativos na complexidade e performance do que benefícios — tal caso de antipadrão foi chamado de nanosserviço (ROTEM-GAL-OZ, 2014).

Os microsserviços podem ser vistos como um estilo específico de arquitetura orientada a serviços (*Service-oriented architecture* — SOA), visto que existem várias características compartilhadas entre os dois. Contudo, o termo arquitetura orientada a serviços é muito amplo, e muitas de suas implementações podem não seguir certos pontos apresentados como aspectos dos microsserviços, como por exemplo, o uso de grande inteligência no mecanismo de comunicação de dados ao invés de delegar tal complexidade aos endpoints do serviço (JAMES, 2013). Esse e outros problemas conhecidos das experiências passadas de sistemas estruturados em SOA fazem com que muitos encarem os microsserviços como uma modernização da arquitetura orientada a serviços.

Apesar do termo microsserviço ter surgido por volta de 2011 (JAMES, 2013), as ideias por trás desse estilo arquitetural não são recentes. O aumento da discussão em torno dos microsserviços nos últimos anos pode ser creditada a avanços tecnológicos tais como a disseminação dos serviços de nuvem, o crescimento de ferramentas de automatização de implantação, a consolidação dos conceitos de *DevOps*, entre outros.

4.6 Cliente Web

A arquitetura do Hedwig permite a criação de vários aplicativos web ou mobile independentes para monitorar e controlar os dispositivos conectados de uma casa. Como toda a comunicação desses clientes é realizada através do servidor na nuvem por meio de WebSockets e da API REST, é possível realizar integrações nas mais diversas plataformas, seja navegadores ou sistemas operacionais nativos de smartphones.

A fim de demonstrar como o usuário final poderia interagir com o sistema em sua totalidade, optamos por desenvolver uma aplicação web. Esse tipo de aplicação tem o desenvolvimento facilitado por uma vasta quantidade de bibliotecas, frameworks, ferramentas e IDEs. Outro fator favorável é a evolução dos navegadores modernos, que possuem funcionalidades de depuração e integrações com os ambientes de dispositivos móveis cada vez melhores. Tais melhorias possibilitaram que aplicativos web pudessem ter uma aparência e percepção mais próxima aos aplicativos nativos.

4.6.1 *Progressive Web Apps*

4.6.1.1 Contexto

Durante a ascensão dos smartphones no mercado, os aplicativos nativos predominaram por serem mais rápidos e possuírem maior suporte para acessar funções do hardware, alcançando assim um padrão mais alto de experiência de usuário do que aplicativos web. Em 2007, Steve Jobs chegou a afirmar que sua visão para o iPhone era de que todos os aplicativos de terceiros fossem web apps (9 TO 5 MAC, 2011). Contudo, apesar desse incentivo, o panorama de Jobs não foi recebido com muita empolgação, e a App Store foi ao ar em 2008 com 500 aplicativos (RICKER, 2008). Em janeiro de 2009, já estavam disponíveis mais de 15 mil aplicativos, com um total de download que superava 500 milhões (MYSLEWSKI, 2009).

Desde então, ocorreram grandes avanços no desenvolvimento web com o amadurecimento do HTML5, CSS3 e JavaScript, a criação de novas bibliotecas e ferramentas, surgimento de mais metodologias para design responsivo e a evolução dos navegadores para cumprir os padrões e especificações mais recentes.

Assim, houve o crescimento do número de aplicativos híbridos, que combinam as técnicas de desenvolvimento web com benefícios dos aplicativos nativos como o suporte para usar funções do hardware. Pode-se dividir os aplicativos híbridos em dois grandes

grupos (RUDOLPH, 2014): aplicativos que usam WebView, uma espécie de navegador interno que é envolvido por uma aplicação nativa, permitindo que algumas APIs nativas sejam acessíveis por JavaScript, e aplicativos híbridos compilados, que são escritos em uma linguagem não nativa e então compilados para várias plataformas de dispositivos móveis. Isso permite obter versões para várias plataformas com o mesmo código, mas para obter tal resultado há limitações durante o desenvolvimento.

Hoje, pesquisas indicam que os aplicativos nativos vêm perdendo força. O número de downloads de aplicativos no Estados Unidos diminui 20% ano a ano (BENSON, 2016). Analisando os dados da *Google Play*, descobriu-se que o aplicativo médio perde aproximadamente 77% dos usuários após 3 dias da instalação (CHEN, 2015) - o que demonstra a dificuldade de se alcançar um bom nível de engajamento. Logo, pedir que o usuário baixe um aplicativo para continuar desfrutando dos serviços de um site pode acarretar em evasão de visitantes. Esses fatores somam-se ao fato de que as tecnologias de desenvolvimento web continuam progredindo e permitindo experiências de usuário cada vez mais ricas - há um crescente suporte na forma de bibliotecas e metodologias para criar interfaces responsivas, transições e animações fluidas e novos tipos de interação. Nesse contexto, surge o conceito de *Progressive Web Apps*, aplicações web que, de fato, podem oferecer uma experiência compatível à de uma aplicação nativa.

4.6.1.2 Conceito

O conceito de *Progressive Web Apps* ou *PWAs* é recente - o termo foi usado pela primeira vez em 2015 pelo designer Frances Berriman e pelo Engenheiro do Google Chrome Alex Russell (RUSELL, 2015). A ideia dessa classe de aplicativos é ir além das aplicações web tradicionais, aproveitando o máximo das funcionalidades mais modernas dos últimos navegadores lançados e combinando-as à navegação mobile para oferecer uma melhor experiência ao usuário.

De acordo com o *Google Developers* (GOOGLE DEVELOPERS, 2017b), os *PWAs* devem ser:

- **Confiáveis** - devem carregar de forma instantânea, independentemente das condições de conectividade, sem prejudicar a experiência com erros e falhas na aplicação
- **Rápidos** - devem responder rapidamente às interações do usuário, com animações e renderizações suaves

- **Envolventes** - devem oferecer uma experiência imersiva, que se assemelhe à de um aplicativo nativo

Além desses três principais aspectos, várias outras características para definir PWAs mais a fundo também são exploradas pelas documentações do Google Developers (GOOGLE DEVELOPERS, 2017a) e por outros desenvolvedores web. Abaixo estão algumas delas:

- **Responsivos** - adaptação aos mais variados tipos de dispositivos e plataformas: desktops, smartphones, tablets, *smart TVs*, entre outros.
- **Atualizados** - realizar a atualização automática do conteúdo
- **Seguros** - uso de medidas para evitar a adulteração de conteúdo
- **Descobríveis** - podem ser encontrados por mecanismos de pesquisa e identificados como aplicativos
- **Linkáveis** - o seu conteúdo é compartilhável por URL

As ideias em torno do desenvolvimento dos PWAs são fortemente relacionadas ao conceito de *progressive enhancement* ou melhoria progressiva, que propõe que camadas de interface e funcionalidades sejam progressivamente adicionadas à aplicação à medida que a conexão e navegador do usuário permitam (CHAMPEON, 2003). Dessa forma, usuários com dificuldades de conectividade e dispositivos mais antigos podem acessar o conteúdo básico, e aqueles que possuem mais banda e navegadores mais modernos podem acessar a uma versão mais completa.

4.6.1.3 Tecnologias e técnicas

Manifesto

O manifesto é um arquivo de texto que oferece informações básicas sobre um aplicativo, como nome, autor, ícone, e descrição. Ele permite que os usuários adicionem o aplicativo à tela inicial de seus aparelhos para acessá-lo mais rapidamente.

Service Workers

Service workers são scripts executados em segundo plano pelo navegador que realizam tarefas que não necessitam de uma página web ou de interações imediatas com o usuário. Aplicações populares para *service workers* são as notificações push e a sincronização em segundo plano.

4.6.1.4 Aplicações

Com o uso de notificações push, a eXtra Electronics, comércio de eletrônicos e eletrodomésticos da Arábia Saudita, obteve um grande aumento de conversão na sua loja virtual. Com uma taxa de cliques nas notificações de 12%, os usuários que optaram por ativar essa funcionalidade retornavam 4 vezes mais ao site e o total das receitas de suas compras aumentou em 100% (GOOGLE DEVELOPERS, 2016b).

Outro caso de sucesso na área de e-commerce é o da AliExpress, que focou na performance e nas funcionalidades offline para obter um aumento de 104% na conversão vinda de usuários novos (GOOGLE DEVELOPERS, 2016a).

O Twitter PWA Lite conseguiu reduzir o uso de dados em até 70% usando imagens otimizadas e se aproveitando ao máximo das informações no cache. Houve um aumento de 75% da quantidade de tweets enviados (GOOGLE DEVELOPERS, 2016c).

4.6.2 JSON *Web Tokens*

4.6.2.1 Definição

O cliente web realiza a autenticação de usuário por meio de JSON *Web Tokens*. JSON *Web Tokens*, ou JWTs, foram definidas para possibilitar a troca de informações de uma forma segura, autônoma e compacta usando objetos JSON (IETF, 2015). A segurança se dá pela assinatura digital das tokens usando o algoritmo HMAC com um segredo ou com criptografia RSA usando pares de chaves pública e privada. JWTs são autônomas no sentido de que o conteúdo das tokens contém toda a informação sobre o usuário, evitando transações adicionais no banco de dados. Por fim, o tamanho compacto das tokens permite que elas sejam enviadas em URLs, parâmetros e cabeçalhos HTTP sem grande ônus ao tempo de transmissão.

Uma token é uma string composta por três partes separadas por pontos. As três partes são: Cabeçalho, Corpo e Assinatura. O Cabeçalho típico contém o tipo da token - ou seja, JWT - e o algoritmo de *hashing* usado, como por exemplo HMAC, SHA256 ou RSA. O Corpo é constituido por *claims* (afirmações) sobre a entidade, geralmente o usuário. As *claims* podem ser de três tipos: reservadas, que geralmente são informações úteis predefinidas como o tempo de expiração, públicas e privadas. O Cabeçalho e o Corpo são codificados usando Base64Url e são usados para criar a Assinatura com o algoritmo definido previamente. A token final é, então, a concatenação do Cabeçalho e do Corpo codificados e da Assinatura.

4.6.2.2 Autenticação

Para realizar a autenticação com JWTs, as tokens são geradas na nuvem durante o cadastro ou login do usuário e então são enviadas ao navegador. A partir desse momento, todas as requisições ao servidor da nuvem irão conter a JWT no campo de Authentication do cabeçalho das requisições HTTP. Somente requisições contendo tal token, e cuja token seja válida, são aceitas no back-end da aplicação. Essa estratégia de implementação é amplamente utilizada para desenvolver a funcionalidade de *single sign-on*.

Um exemplo de cabeçalho HTTP que usa JWT para autenticação é:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    ↪ eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
    ↪ .TJVA950rM7E2cBab30RMhrHDcEfxjoYZgeFONFh7HgQ
```

No Hedwig, o servidor na nuvem, ao receber um pedido de autenticação e validá-lo, gera a JWT e a manda para o aplicativo cliente, que a armazena no *local storage*. O *local storage* é uma forma de armazenamento no navegador que permite que aplicações guardem dados que persistam além da duração de uma sessão. É considerada uma alternativa aos cookies com maior capacidade e segurança.

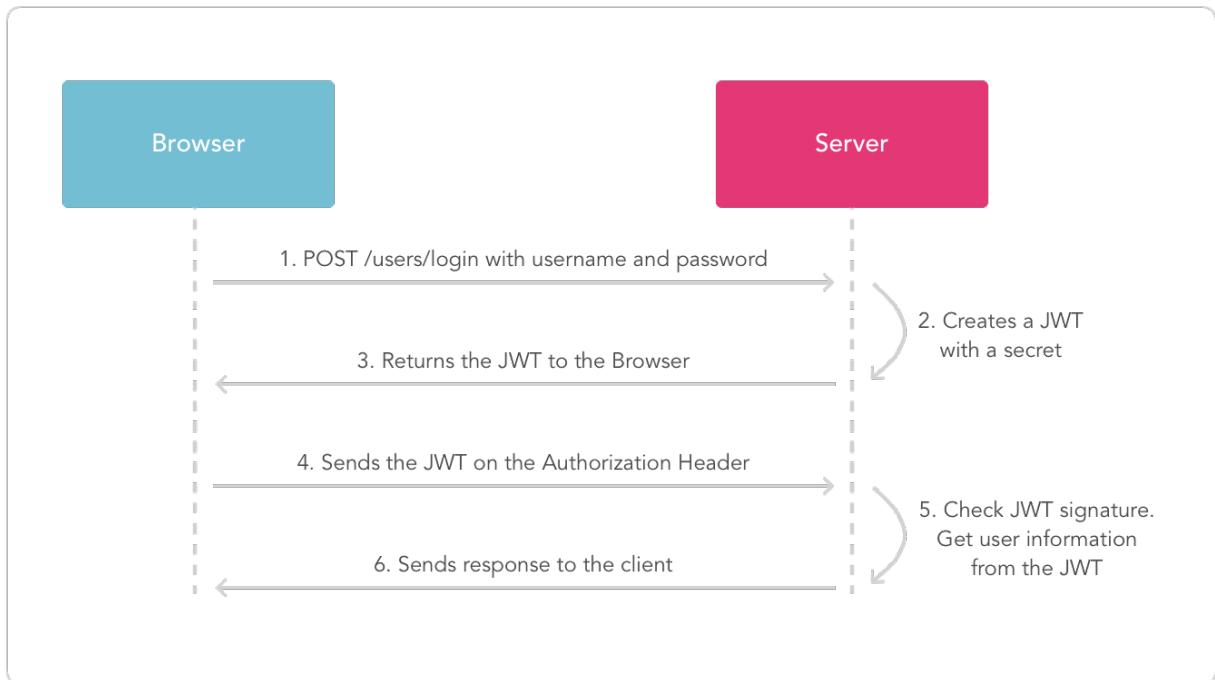
Os navegadores em geral podem armazenar em torno de 300 cookies, com um limite de aproximadamente 20 por domínio, cada um com um tamanho máximo de 4kB (COOKIE LIMITS, 2016). Já os limites para o local storage geralmente são de, no mínimo, 5MB por domínio (GOOGLE CHROME, 2017).

A autenticação com JWT é *stateless*, pois não há necessidade de guardar o estado de autenticação do usuário no banco de dados. Ao contrário dos cookies, as tokens podem ser compartilhadas por vários domínios sem as limitações de *Cross-Origin Resource Sharing* (CORS). Isso possibilita que uma única token possa ser repassada serviço a serviço para completar uma transação que necessita de autenticação em um sistema mais complexo, como é o caso da arquitetura de microsserviços.

4.7 Comunicação

Conforme explicado anteriormente, neste projeto utilizamos tanto protocolos de comunicação próprios quanto os elaborados comercialmente. A arquitetura desenvolvida aqui busca viabilizar a robustez do sistema, trabalhando em um nível local e outro nível remoto, onde o usuário terá o controle de sua casa por meio do *smartphone* ou computador

Figura 14: Diagrama de interação na autenticação por JWT



Fonte: (JWT, 2016)

pessoal.

Nosso serviço em nuvem recebe as requisições do usuário por meio de um cliente web ou nativo. Esse servidor processa as requisições, aplicando os filtros de segurança necessários, de modo a consultar a autenticidade do pedido e verificar se aquele usuário possui as permissões necessárias para o serviço que deseja operar. Os serviços da nuvem se comunicam com o servidor local da casa requisitada, o qual também aplica os filtros de segurança necessários, e realiza a comunicação com os módulos.

A infraestrutura de comunicação entre a nuvem e o servidor local, e o servidor local e os sensores e atuadores utiliza o protocolo de aplicação *MQTT*, referência em aplicações *IoT* no mundo. O protocolo *MQTT* é estabelecido em cima dos protocolos TCP/IP (nas camadas inferiores) e é orientado à sessão, diferentemente do protocolo HTTP, de mesma camada.

O protocolo *MQTT* é do tipo Pub/Sub (de *publisher/subscriber*) e é estritamente orientado à tópicos. Assim, um *subscriber* se inscreve a um tópico de seu interesse, e recebe todas as publicações que um *publisher* realizar. Os tópicos são organizados com estrutura semelhante a de um sistema de arquivos Unix, com níveis hierárquicos separados por barras, de modo que o subscriber pode se inscrever para tópicos utilizando *wildcards* (* e +, os quais são válidos para mais de um nível e um único nível, respectivamente).

Para interconectar os tópicos, com *publishers* e *subscribers*, é necessário um agente que realiza a transmissão das mensagens, e que garante a segurança e confiabilidade. Esse agente é conhecido como Broker (em versões anteriores) ou Server (na versão atual, V3.1.1). O *broker* irá permitir ou negar a subscrição ou a publicação a determinado tópico.

A segurança da troca de mensagens é realizada por meio do protocolo TLS (*Transport Layer Security*) que encripta os segmentos na camada de transporte. Toda a parte de segurança e criptografia será detalhada no momento oportuno, bem como a organização dos tópicos implementados.

Além disso, o protocolo *MQTT* oferece três tipos de QoS (*Quality of Service*), possibilitando: diminuir o overhead ao máximo, enviando a mensagem uma única vez, na configuração mais simples; garantir que a mensagem seja entregue no mínimo uma vez, na configuração de segundo nível; garantir que a mensagem seja entregue exatamente uma vez, no terceiro nível, o que aumenta o overhead, consequentemente.

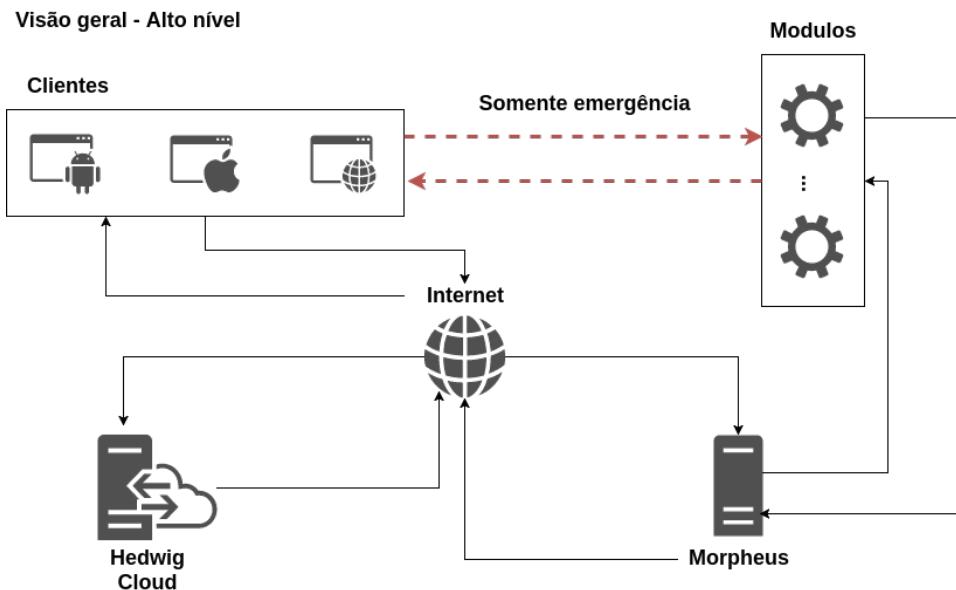
As mensagens são transmitidas em texto puro, e é necessário estabelecer um protocolo para a sua utilização. Utilizaremos aqui o protocolo que define a configurações das mensagens, desenvolvido no projeto HomeSky.

O *broker* Mosquitto³ será utilizado, e foi escolhido por ser amplamente adotado em projetos de *IoT*, além de ser open source e com licença abrangente (MIT). Entretanto, há diversas possibilidades, como o HiveMQ, adotado no projeto HomeSky, e com grande uso em aplicações enterprise.

A arquitetura de comunicação é representada pela Figura 15, com um alto nível de abstração, cujos detalhes serão vistos no momento oportuno, com granularidade menor.

³<https://mosquitto.org/>

Figura 15: Visão alto nível da comunicação no Hedwig



5 Metodologia

5.1 Gerência do projeto

Para realizar a gerência do projeto Hedwig, foram usadas as diretrizes do Guia PM-BOK (PMI, 2004) e da norma ISO/IEC 12207 (ISO/IEC-IEEE, 2008) como referência para coordenar os processos.

Para gerenciar as tarefas, estudos e pesquisas necessárias para a realização do projeto, foi utilizado o Trello¹ - sistema online para organização de ideias e projetos, que permite listagem e acompanhamento de tarefas a serem realizadas, com deadlines, responsáveis e categorização em diversos tipos de tarefas.

5.1.1 Gerência de Escopo Tempo

5.1.2 Gerência de Partes Interessadas Aquisição

5.1.3 Gerência de Processos de Software

Para gerenciar o código-fonte e permitir o trabalho da equipe em múltiplas partes do projeto ao mesmo tempo, foi utilizado o Git, um sistema de controle de versão distribuído. Para publicação do código, foi escolhido o GitHub, onde está a organização do projeto Hedwig² e os repositórios de código dos módulos associados ao sistema. A preferência pelo GitHub se deu pelas suas funcionalidades de gerenciamento e colaboração como a notificação de bugs, acompanhamento do progresso de tarefas e criação de wikis, além de ser uma plataforma conhecida por abrigar grandes projetos open-source que chegam a ter centenas ou milhares de contribuidores (GITHUB, 2016).

Para o fluxo de trabalho nesses repositórios, foi utilizado o fluxo conhecido como *Feature Branch Workflow* (ATLASSIAN, 2017), caracterizado pela criação de *branches*

¹Pode ser acessado gratuitamente em <https://trello.com/>

²<https://github.com/hedwig-project>

(ramificações) para o desenvolvimento de cada nova funcionalidade. Ao final do desenvolvimento de cada funcionalidade, é feito um pedido para mesclar o código desenvolvido em tal ramificação com o da ramificação principal (*master branch*).

5.1.4 Gerência de Partes Interessadas

5.1.5 Gerência de Comunicação

5.1.6 Gerência de Escopo

5.1.7 Gerência de Riscos

5.2 Pesquisa bibliográfica

O estudo dos tópicos relacionados a aprendizagem de máquina foi realizado com auxílio do curso Aprendizagem Automática do Professor Andrew Ng³, oferecido pela Universidade de Stanford e disponibilizado no Coursera, uma plataforma de MOOCs (*Massive Open Online Courses*) que oferece cursos abertos e especializações.

Os cursos da especialização em *Data Science* da Universidade Johns Hopkins⁴, também disponíveis no Coursera, foram usados como referência e treinamento para realizar a coleta de dados de maneira metódica. Por esse motivo, foi dada maior atenção ao curso *Getting and Cleaning Data*. Contudo, também foi aproveitado conteúdo do curso *Practical Machine Learning*.

5.3 Ferramentas e tecnologias

Para aprender a utilizar a biblioteca React para o desenvolvimento do front-end, foi usada como referência a documentação oficial⁵ oferecida pelo Facebook e o curso *React for Beginners* de Wes Bos⁶. O aprendizado de Redux foi auxiliado pelo curso *Learn Redux*⁷, do mesmo autor.

³<https://www.coursera.org/learn/machine-learning>

⁴<https://www.coursera.org/specializations/jhu-data-science>

⁵<https://facebook.github.io/react/docs/hello-world.html>

⁶<https://reactforbeginners.com/>

⁷<https://learnredux.com>

6 Implementação

6.1 Morpheus

6.1.1 Descrição

Morpheus é o servidor local responsável pela interconexão da casa inteligente com os serviços de nuvem. O nome tem sua origem na mitologia grega, onde o Deus dos sonhos, Morpheus, era responsável pelo envio de mensagens entre dois mundos diferentes, o dos deuses e o dos mortais [TODO add source]. Sua principal atribuição é garantir que a troca de mensagem entre os módulos e a nuvem seja realizada com segurança e confiabilidade, munindo-se de soluções robustas para desempenhar o seu papel.

6.1.2 Plataforma

O Morpheus tem seu desenvolvimento realizado em Java. Conforme será detalhado em seguida, tal escolha foi realizada com base na portabilidade que a máquina virtual Java (JVM) oferece, bem como na disponibilidade de bibliotecas e serviços largamente utilizados em aplicações comerciais. O servidor foi construído utilizando-se o Spring Boot Framework, com a utilização de seu container de Inversão de Controle (*IoC - Inversion of Control*), para injeção de dependências. Essa técnica diminui o acoplamento entre classes, e permite a evolução e implementação de novas funcionalidades de maneira mais facilitada.

Para se comunicar com os módulos, o Morpheus utiliza-se da conexão com um *broker MQTT*. O *broker* que utilizamos é o Mosquitto, por ser uma solução Open Source largamente utilizada em projetos de Internet das Coisas. Conforme detalhado a frente, configurações de segurança específicas para nosso projeto foram registradas no *broker*. Para a conexão com os serviços na nuvem, é utilizado um canal WebSocket, aberto pelo Morpheus (cliente) e aceito pela nuvem (servidor). Esta solução veio a partir de uma discussão em relação à segurança, a qual está documentada aqui. Na figura a seguir, é

possível verificar a arquitetura desenvolvida.

6.1.3 Tecnologias utilizadas

Toda a implementação do Morpheus foi realizada na linguagem Java. Desde o começo do projeto, decidiu-se que a escolha de tecnologias para implementação das diversas camadas deveria ter por base os seus benefícios, e não necessitaria ser rígida ou uniforme. Assim, o principal esforço foi sempre no planejamento das interfaces de comunicação entre as partes, que poderiam ser desenvolvidas em linguagens complementamente diferentes. Os sistemas de nuvem, por exemplo, foram implementados em Node.js. O aplicativo web, também em JavaScript, com utilização da biblioteca React. Os módulos de hardware foram programados em C e, como comentado acima, o controlador local em Java. Essa flexibilidade permitiu que fossem utilizados os recursos e tecnologias que fossem melhores integrados com os requisitos propostos.

Um requisito essencial para o controlador local é a sua robustez. Em um cenário em que este controlador não esteja disponível, a casa passa a funcionar em estado de emergência, onde os comandos são reduzidos, e não permitem acesso remoto. Entretanto, há inúmeras possibilidades e eventos que poderiam causar a queda deste controlador, muitas das quais referem-se a situações fora de nosso alcance. A falta de energia na residência, ou de Internet, interrompe o seu funcionamento, e não é possível ter controle sobre tal situação. O mesmo ocorre no evento de problemas de hardware, na plataforma que o sistema estiver rodando. Além do fato de que tais situações estão fora de nosso controle, os planos para contenção dos seus efeitos são complexos, custosos e fogem do escopo deste projeto, como seria o caso de se implementar duplicações, banco de baterias e tecnologia celular para comunicação secundária.

Há, entretanto, problemas no software que poderiam afetar o funcionamento do controlador. Por meio de testes, muitos desses problemas podem ser evitados, ainda em tempo de desenvolvimento. A utilização de tecnologias que facilitam o desenvolvimento seguro da aplicação é uma vantagem para este caso, já que ferramentas estão disponíveis para que haja maior controle sobre o código desenvolvido, e pode-se detectar erros mais facilmente, ainda em tempo de compilação, por exemplo.

O controlador local também precisa lidar com as requisições assíncronamente. Parte desta tarefa é facilitada com a utilização do sistema de mensageria *MQTT*, operado pelo *broker* Mosquitto. Com sua utilização, mensagens podem ser enviadas, e mesmo que o controlador não consiga recebê-las, elas não serão perdidas. Entretanto, devido

às características do sistema proposto, as mensagens precisam ser operadas sem maiores demoras. O controlador deve receber e processar as mensagens paralelamente, e não esperar o processamento de uma mensagem inteira, para assim processar a próxima, de modo que paralelismo deve ser parte essencial da arquitetura.

Ainda, para a integração com os serviços da nuvem, é necessário a utilização de JSON, para a serialização das mensagens, em um formato que pode ser desserializado posteriormente, independentemente da plataforma. Para a utilização de WebSockets, é necessário o uso de bibliotecas disponíveis, de modo que o desenvolvimento seja facilitado. Por último, é necessário gerenciar eficientemente todas essas dependências. Atualizá-las quando necessário, ou substituí-las, se desejado, deve ser uma tarefa simples.

A arquitetura oferecida pelo Java mostra ser efetiva para as necessidades levantadas acima. Com a utilização de uma IDE avançada, inúmeros recursos estão disponíveis para limpeza, refatoração, organização do código, etc. O *framework* Spring Boot¹ foi utilizado para o desenvolvimento, por oferecer diversos recursos que facilitam o desenvolvimento, com configurações de ambiente e o oferecimento de um *container* para inversão de controle e injeção de dependência². Além disso, o gerenciador de dependências Gradle³ foi também utilizado, por oferecer um poderoso ambiente para configurar, construir e distribuir aplicações. Gradle faz uso de Groovy⁴, tecnologia que também roda na *Java Virtual Machine* (JVM). Por outro lado, o uso de tais ferramentas e plataformas necessita de hardware mais robusto, para que funcione, sendo uma desvantagem. Entretanto, frente aos benefícios, ainda é vantajoso a utilização de Java, neste caso.

Java é utilizado em vasta gama de aplicações, desde complexos softwares comerciais como a IDE Eclipse, até software embutidos, como controladores de BlueRay⁵. O padrão de inversão de controle, com injeção de dependências, foi utilizado para o desenvolvimento, de modo a diminuir o acoplamento entre as partes, e facilitar futuras modificações. Assim, cada módulo recebe, em seu construtor, todas as dependências que serão utilizadas. A responsabilidade da construção de tais dependências passa, então, a ser responsabilidade do gerenciador de contexto, e não mais do módulo. A escolha do Java 8 foi decidida para que o desenvolvimento possa utilizar certos recursos de paradigmas funcionais, como o conceito de Streams de dados e Funções Lambdas.

Internamente, o Morpheus é dividido em pacotes, que são responsáveis pela modela-

¹<https://projects.spring.io/spring-boot/>

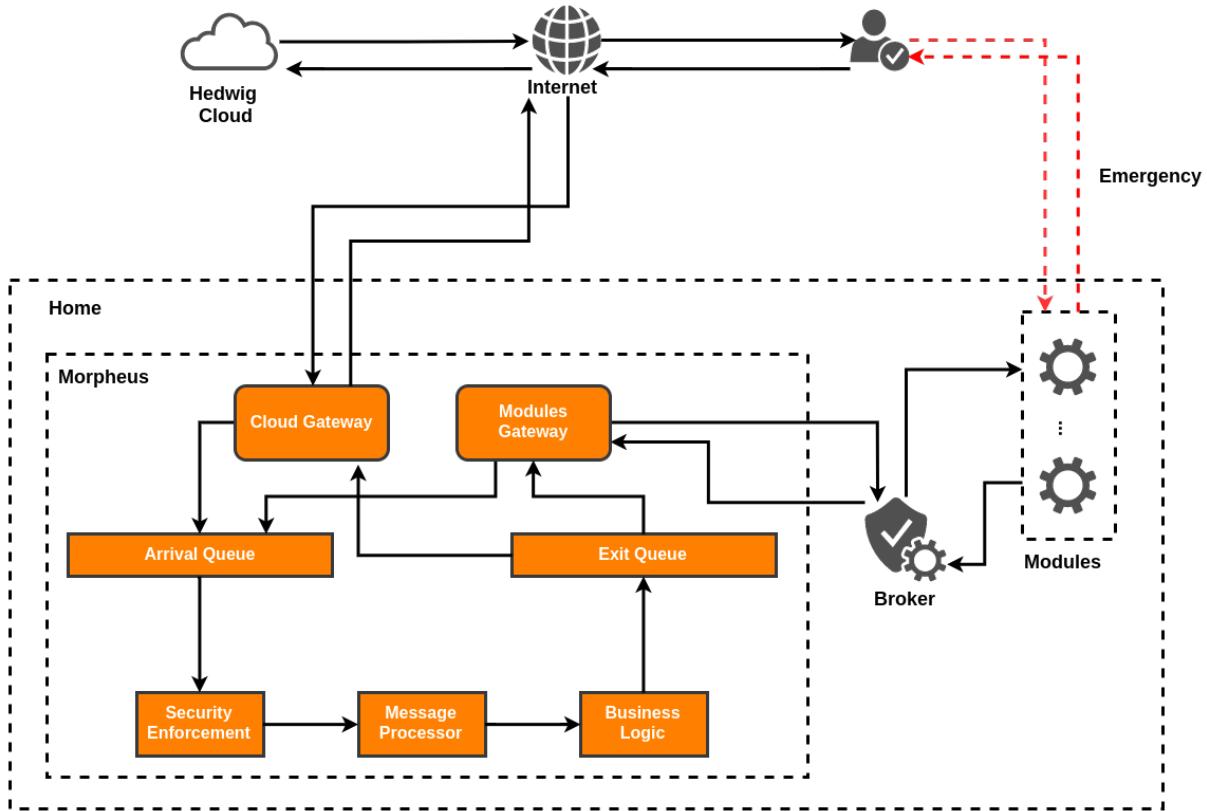
²<https://martinfowler.com/articles/injection.html>

³<https://gradle.org/>

⁴<http://groovy-lang.org/>

⁵<http://www.oracle.com/technetwork/articles/javame/bluray-142687.html>

Figura 16: Arquitetura do servidor local



gem do problema. Há classes que modelam o domínio, que executam as regras de negócio, que fazem a interface entre outros sistemas (*MQTT Mosquitto Broker* e nuvem), e que fazem a execução de tarefas como backup de mensagens, e serviços de conversão. Assim que uma mensagem chega ao Morpheus, ela é reconhecida e é realizado o seu parsing, para as estruturas de domínio internas. Caso haja algum erro, para mensagens vindas da nuvem, há o envio de relatório com os problemas encontrados. A partir daí, a mensagem é colocada em uma fila interna, onde outro módulo será responsável por capturá-la e realizar o processamento necessário.

6.1.4 Requisitos

Para a concepção do servidor local, foram discutidos os seus requisitos funcionais e não funcionais, bem com suas prioridades na implementação.

6.1.4.1 Requisitos Funcionais

Configuração dos módulos físicos

De acordo com as regras e interfaces estabelecidas, documentadas aqui, os módulos podem ser configurados por meio de mensagens. Os serviços da nuvem enviarão os parâmetros de configuração de cada módulo ao Morpheus, que os transmitirá ao módulo.

Conexão de emergência

Quando não há conexão da casa com a nuvem, deverá haver um canal para comunicação local entre o aplicativo e alguns dos módulos, com funcionalidades limitadas, apenas para serviços essenciais.

Envio de dados para a nuvem

Dados provenientes de sensores são enviados para a nuvem, para que sejam tratados de acordo com as regras de Business Intelligence, e utilizados em algoritmos de Machine Learning.

Persistência de dados

Quando não houver conexão, o servidor deverá armazenar os dados localmente e, quando solicitado, enviá-los à nuvem.

Tentativas de reenvio

Quando uma mensagem não é enviada com sucesso à nuvem, o Morpheus deverá tentar novamente, por um número configurável de vezes, em um curto espaço de tempo. Isso ocorre pois se determinada mensagem não pode ser enviada em uma janela temporal, ela perde o seu sentido, bem como por questões de segurança.

Envio de e-mail

Se o Morpheus estiver desconectado da nuvem por um período, configurável, deverá avisar o usuário, por meio de uma mensagem de e-mail.

Verificação do time-stamp

Quando uma nova mensagem chegar, seu timestamp deverá ser verificado, e a mensagem tomará curso somente se não for obsoleta.

Tomadas de ação

Quando o usuário requisitar uma tomada de ação, esta deverá ser enviada por meio de uma mensagem ao Morpheus, o qual a transmitirá ao módulo.

Configuração em arquivo

As configurações básicas do Morpheus devem ser registradas em um arquivo YAML, que será lido durante a inicialização.

Listeners para diferentes tipos de mensagens

Deverão haver *listeners* para todos os tipos de mensagens, definidos aqui, que serão recebidos da nuvem e dos módulos.

6.1.4.2 Requisitos Não Funcionais

Processamento concorrente

Toda a infraestrutura do Morpheus deverá permitir o processamento de mensagens de maneira concorrente. Não deve ser permitido esperar o processamento completo de uma mensagem para que outra comece a ser processada.

Utilização de criptografia na troca de mensagens com a nuvem

Os dados que trafegam entre a nuvem e o servidor local não devem ser codificados em texto puro. Devem estar protegidos contra *sniffers*.

Conversão de mensagens

As mensagens enviadas à nuvem devem estar em formato JSON, e não no protocolo definido aqui, que refere-se apenas à troca de mensagens entre os módulos e o Morpheus.

Serialização das configurações

O servidor deverá serializar e persistir as configurações relativas aos módulos que foram configurados, e carregá-las em sua inicialização.

Destruição de pools de threads

Ao ser desligado, todos os *pools* de *threads* criados devem ser destruídos.

6.1.5 Especificações

6.1.5.1 Tópicos

Todos os tópicos devem seguir o formato especificado abaixo. Com essa formatação, é possível garantir que:

1. Somente o Morpheus conseguirá publicar em qualquer tópico, ou ser um *subscriber* de qualquer tópico.
2. Cada módulo somente consiga publicar no tópico determinado para ele, que será garantido com as credenciais (usuário e senha) fornecidos pelo tópico.
3. Caso um módulo malicioso seja implantado, com o roubo das credenciais de um módulo legítimo, o impacto será unicamente concentrado naquele tópico, não atingindo outros módulos.

Temos as seguintes regras:

hw/<ID do módulo>/s2m (Server to Module - o módulo deve ser *subscriber* desse tópico. O servidor deve ser *publisher* desse tópico).

hw/<ID do módulo>/m2s (Module to Server - o servidor deve ser *subscriber* desse tópico. O módulo deve ser *publisher* desse tópico).

6.1.5.2 Regras de negócio

O servidor foi desenvolvido com base nas regras de negócio seguintes.

1. Após a compra de cada módulo, o usuário deverá registrar online a aquisição. O servidor da nuvem enviará para o servidor local, da casa, a requisição para configurar o módulo.
2. Cada módulo enviará mensagens para o servidor local com os seus dados, por meio do *MQTT*.
3. Para a troca de senha do *WiFi*, o usuário cadastra no site a nova senha. O servidor na nuvem faz a requisição para o servidor local, o qual enviará um arquivo de configuração com a nova senha para cada um dos módulos registrados. Após a configuração de todos os módulos, o servidor local envia resposta de sucesso para a nuvem, a qual indica ao usuário que a troca de senha já pode ser feita com sucesso.
4. Todo módulo sai de fábrica configurado com o tópico que deve se inscrever e publicar, com base no seu ID, o qual será o seu usuário, e também haverá a senha para se autenticar junto ao *broker MQTT*.

6.1.5.3 Definição de interfaces

- Há três tipos de mensagens que vão do Morpheus para os módulos:
 - Configuração (configuration)
 - Requisição de Ação (action_request)
 - Requisição de dados (data_transmission)
- Há três tipos de mensagens que chegam dos módulos:
 - Confirmação (confirmation)
 - Envio de Dados (data_transmission)
 - Data Request (data_request)

6.1.5.4 Definição das mensagens

Configuração (configuration)

- Sentido: Morpheus para módulo
- Uso: Envio de parâmetros para configuração dos módulos.

Configuração de hora:

```
#configuration
$ts: <timestamp>
$ty: time_config
@
updated_ntp: <segundos desde 0h de 1 de Janeiro de 1970, 64 bits>
@
```

Configuração de nome:

```
#configuration
$ts:<timestamp>
$ty:name_config
@
new_name: <string>
new_rele1name: <string>
new_rele2name: <string>
@
```

Configuração de comunicação:

```
#configuration
$ts: <timestamp>
$ty: communication_config
@
new_ssid: <string>
new_password: <string>
ip_local: <string>
ap_mod: <0|1> // 0= sempre ativo; 1 = automatico
ap_name: <string> //para acesso direto
ap_password: <string>
@
```

Configuração de RF:

```
#configuration
$ts: <timestamp>
$ty: rf_config
@
<sensor_abre|sensor_fecha|rele1_rf|rele2_rf>: <store|clear>
@
```

Configuração de display:

```
#configuration
$ts: <timestamp>
$ty: display_config
@
displaytype: <1|2|3>
backlight: <0|1|2>
@
```

0 = desligado, 1=ligado, 2 = automático

Requisição de ação (action_request)

- Sentido: Servidor para módulo

- Uso: Quando um usuário faz a requisição de uma ação por meio do aplicativo. Por exemplo, quando deseja-se acender uma luz, o aplicativo envia uma requisição para o servidor local, o qual enviará uma mensagem de action_request para o módulo correspondente.

Requisição de acionamento:

```
#action_request
$ts:<timestamp>
$ty:rele1_action
@
rele1: <0|1>
@
```

Essa mensagem fica sem efeito no caso do módulo de acesso, que usa mensagens com senha para o acionamento o Rele 1. Rele1: 0 = desligar; 1 = ligar

```
#action_request
$ts:<timestamp>
$ty:rele2_action
@
rele2: <0|1>
@
```

Rele2: 0 = desligar; 1 = ligar

Requisição de SW Restart:

```
#action_request
$ts:<timestamp>
$ty:sw_reset
@
swreset: <0|1>
@
```

0 = não; 1 = confirmar reinicio

Requisição de Teste de Auto Reset:

```
#action_request
$ts:<timestamp>
```

```
$ty: autoreset_test
@
autoreset: <0|1>
@
```

0 = não; 1 = confirmar reinicio

Confirmação (confirmation)

- Sentido: Do módulo para o servidor
- Uso: Confirmação de uma configuração, patch ou requisição de ação, vindas do servidor.

Confirmação de hora

```
#confirmation
$ts: <timestamp>
$ty: time_confirm
@
ntp: <segundos desde 0h de 1 de Janeiro de 1970, 64 bits>
@
```

Confirmação de nome

```
#confirmation
$ts: <timestamp>
$ty: name_confirm
@
name: <nome>
rele1name: <string>
rele2name: <string>
@
```

Confirmação de comunicação

```
#confirmation
$ts: <timestamp>
$ty: communication_confirm
@
ssid: <novo ssid>
```

```

password: <nova senha>
ip local: <novo ip local fixo>
ap mod: <sempre ativo|automatico>
ap name: <nome do ap para acesso direto>
ap password: <senha do ap para acesso direto>
@
```

Confirmação de RF:

```

#confirmation
$ts: <timestamp>
$ty: rf_confirm
@
<nome_do_[sensor | controle | funcao]: <valor_gravado>
@
```

Configuração de Display:

```

#confirmation
$ts: <timestamp>
$ty: display_confirm
@
displaytype: <1 | 2 | 3>
backlight: <0 | 1>
@
```

Confirmação de SW Restart:

```

#confirmation
$ts: <timestamp>
$ty: sw_reset_confirm
@
swreset: <0 | 1>
@
```

Confirmação de Teste de Auto Reset:

```

#confirmation
$ts: <timestamp>
$ty: autoreset_test_confirm
```

```
@
autoreset: <0 | 1>
@
```

Transmissão de dados (data_transmission)

- Sentido: Do módulo para o servidor
- Uso: Envio de dados de sensores para o servidor

Transmissão de Umidade, Temperatura, Presença e Reles

```
#data_transmission
$ts: <timestamp>
$ty: temp_umi_pres
@
s1:umidade
v11:<int de 0 a 100>
s2:temperatura
v12:<int>
s3:presenca
v13:<0|1>
s4:r11
v14:<0|1>
s5:r12
v15:<0|1>
s6: abertura
v16: <0|1>
s7: luz
v17: <int de 0 a 1000>
@
```

Requisição de dados (data_request)

- Sentido: Do módulo para o servidor
- Protocolo: *MQTT*

- Uso: Requisição de alguma informação do servidor. Ex.: Atualização de hora

Mensagens próprias para o Módulo de Acesso Configuração de Alarme

```
#configuration
$ts:<unix time>
$ty:alarm_config
@
alarme: <0|1>
alarme_tempo: <int do tempo em segundos>
@
```

Configuração de Luz Automática

```
#configuration
$ts:<timestamp>
$ty:auto1_config
@
initial_time1: <int de 0 a 23>
final_time1: <int de 0 a 23>
time_keepon1: <tempo em minutos>
time_deslmanual1: <tempo em minutos>
sensorauto1: <sensor1|sensor2|nao>
@

#configuration
$ts:<timestamp>
$ty:auto2_config
@
initial_time2: <int de 0 a 23>
final_time2: <int de 0 a 23>
time_keepon2: <tempo em minutos>
time_deslmanual2: <tempo em minutos>
sensorauto2: <sensor1|sensor2|nao>
@
```

Configuração de Senha

```
#configuration
$ts:<timestamp>
```

```
$ty:password_config
@
old_password: <string>
new_password: <string>
@
```

Abertura de Portão

```
#action_request
$ts:<timestamp>
$ty:abertura_portao
@
password: <string>
@
```

Confirmação de Alarme

```
#confirmation
$ts:<timestamp>
$ty:alarm_confirm
@
alarme: <0|1>
alarme_tempo: <tempo em minutos>
@
```

Confirmação de Luz Automática

```
#confirmation
$ts:<timestamp>
$ty:auto1_confirm
@
initial_time1: <integer de 0 a 23>
final_time1: <integer de 0 a 23>
time_kepon1: <tempo em minutos>
time_deslmanual1: <tempo em minutos>
sensorauto1: <sensor1|sensor2|nao>
@

#confirmation
$ts:<timestamp>
```

```
$ty:auto2_confirm
@
initial_time2: <integer de 0 a 23>
final_time2: <integer de 0 a 23>
time_keepon2: <tempo em minutos>
time_deslmanual2: <tempo em minutos>
sensorauto2: <sensor1|sensor2|nao>
@
@
```

Confirmação de Senha

```
#confirmation
$ts:<timestamp>
$ty:password_confirm
@
password: <string>
@
@
```

Transmissão de Estado de Acesso

```
#data_transmission
$ts:<timestamp>
$ty:acesso_estado
@
s1:aberto
vl:<0|1> // 0 = fechado; 1 = aberto
s2:alarme
vl:<0|1> // 0 = desligado; 1 = ativado
s3:tempo_alarme
vl:<int>
@
@
```

tempo alarme: tempo em minutos em que o sensor de abertura está aberto.

6.1.5.5 Testes realizados da comunicação Morpheus e módulos:

Para que fosse simulado o envio de mensagens, o aplicativo *MQTT Fx*⁶ foi utilizado. Com o uso deste software, é possível se inscrever em determinado tópico (enviando

⁶<http://mqqtfx.jensd.de/>

as credenciais para o *broker*, tanto em forma de usuário e senha, quanto em forma de certificados), bem como publicar no tópico desejado.

Requisição de acionamento 1:

```
#action_request
$ts:<timestamp>
$ty: rele1_action
@
rele1: 0
@
```

Esperado: 0 no serial do Arduino, indicando recebimento

Resultado: De acordo

Requisição de acionamento 2:

```
#action request
$ts: <timestamp>
$ty: rele2_action
@
rele2: 1
@
```

Esperado: 1 no serial do Arduino, indicando recebimento

Resultado: De acordo

Requisição e confirmação de SW Restart:

```
#action_request
$ts: <timestamp>
$ty: sw_reset
@
swreset: 1
@
```

Esperado: Confirmação de SW Restart no tópico MQTT m2s

Resultado: De acordo

Requisição e confirmação de Teste de Auto Reset:

```
#action request
```

```
$ts: <timestamp>
$ty: autoreset_test
@
autoreset: 1
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Configuração e confirmação de hora:

```
#configuration
$ts: 293029
$ty: time_config
@
updated_ntp: 293029
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Configuração e confirmação de nome:

```
#configuration
$ts: 432524
$ty: name_config
@
new_name: NovoNome
new_rele1name: Portal1
new_rele2name: Portal2
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Configuração e confirmação de comunicação:

```
#configuration
$ts: 5349545
$ty: communication_config
```

```
@
new_ssid: Novossid
new_password: novaSenha
ip_local: 192.168.0.32
ap_mod: automatico
ap_name: AcessoDiretoAP
ap_password: 1234
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Configuração e confirmação de RF:

```
#configuration
$ts: 4839434
$ty: rf_config
@
Janela4: 01234
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Configuração e confirmação de display:

```
#configuration
$ts: 543242
$ty: display_config
@
displaytype: 369
backlight: 1
@
```

Esperado: Confirmação no tópico MQTT m2s

Resultado: De acordo

Transmissão de Umidade Temperatura e Presença e Reles

```
messageToSend = UmiTempPresReles(0,80,25,1,1,0);
```

```
//UmiTempPresReles(unsigned long ts, int umidade, float temp, bool pres,
→ bool rele1, bool rele2)
```

Esperado: Mensagem no Tópico MQTTS m2s Resultado: De acordo

6.1.6 Comunicação entre Morpheus e Nuvem

Inicialmente, foi proposto um modelo arquitetural onde, para a comunicação com a nuvem, haveriam *endpoints* para requisições HTTP tanto do lado da casa quanto do lado da nuvem. Assim, quando o Morpheus precisasse enviar uma mensagem, seria necessário que fosse realizada uma chamada ao *endpoint* correspondente. Neste sentido (Morpheus para nuvem), não há nenhum problema, pois é possível garantir configurações avançadas de segurança, bem como a utilização de load balancers e servidores terceiros (como Akamai⁷), para lidar com ataques do tipo DoS (*Denial of Service*).

O problema, no entanto, está em garantir a segurança e usabilidade do lado da casa. Primeiramente, os IPs residenciais não são fixos, e são trocados a cada nova conexão. Assim, se a conexão com a Internet for perdida, por exemplo, um novo IP será atribuído àquela residência. Dessa forma, após essa troca, a não ser que o Morpheus atualize a nuvem, não será possível receber as mensagens que chegariam dos serviços remotos. Esta questão, no entanto, é contornável, por meio de um serviço de watchdog, que seria responsável por analisar o IP e notificar a nuvem sobre a troca, sempre que esta ocorrer. Há, ainda, um problema mais grave e mais difícil de ser contornado. Com essa arquitetura, o Morpheus também será um servidor, do ponto de vista da nuvem, e qualquer dispositivo pode tentar fazer uma requisição em um dos *endpoints* disponíveis. Mesmo que sejam checados os dados da requisição, para garantir que esta é válida, temos ainda uma grave ameaça de segurança, em relação à negação de serviço. Para que este risco fosse minimizado, seriam necessárias configurações avançadas no roteador local, e mesmo assim, este não seria suficiente para processar um grande número de requisições, deixando a casa vulnerável.

Foi discutida, então, uma mudança arquitetural na forma de comunicação entre a nuvem e a casa. A solução para o problema se encontra no uso de WebSockets. Assim, o Morpheus se comporta como um cliente em relação à nuvem, e é sempre ele que abre uma conexão. Assim, já não há mais a vulnerabilidade local, de estar exposto às negativas de serviço. Além disso, a conexão se mantém aberta, e forma um caminho *full duplex*, de

⁷<https://www.akamai.com/>

modo que é possível receber as mensagens da nuvem a qualquer momento também. Com essa arquitetura, os desafios relativos à segurança recaem aos servidores, e não mais à casa, de modo que é possível gerenciar esses riscos, como o fazem grandes empresas, de forma transparente ao cliente final.

Por fim, somente restou um *endpoint* no Morpheus, que seria o de emergência. Este *endpoint* somente aceita requisições vindas do *localhost*, e não mais de fora.

6.1.7 WebSocket

Com a utilização do canal de comunicação por WebSocket, foram utilizados eventos, que são recebidos e enviados, para a comunicação. São eles os descritos abaixo.

6.1.7.1 Morpheus

O Morpheus ouvirá os seguintes eventos, vindos da nuvem.

- configurationMessage
- actionRequest
- dataTransmission (Requisitar informações sobre módulo, e.g. se portão está aberto ou não).

6.1.7.2 Nuvem

A nuvem ouvirá os seguintes eventos, vindos do Morpheus.

- confirmation
- configuration
- data

Definição de mensagens entre Nuvem e Morpheus

```
configuration =
{
    "configurationId": <configurationId> ,
    "timestamp": <timestamp> ,
```

```

    "morpheusConfiguration": <morpheusConfiguration> ,
    "modulesConfiguration": <modulesConfiguration>
}

<morpheusConfiguration> =
{
    "register": [<eachModuleRegistration> ] ,
    "requestSendingPersistedMessages": <true | false>
}

<eachModuleRegistration> =
{
    "moduleId": <moduleId> ,
    "moduleName": <moduleName> ,
    "moduleTopic": <moduleTopic> ,
    "receiveMessagesAtMostEvery": <time> ,
    "qos": <qosLevel>
}

```

Requisitos:

O campo receiveMessagesAtMostEvery deve estar no formato “<time>:<unit>” A unidade deve ser “s” para segundos, “m” para minutos ou “h” para horas. O valor padrão é 60 segundos.

Ex: Requisição de mensagens persistidas e configuração do Morpheus

Configuração de módulo

A seção de configuração de módulo será um objeto com duas partes. A primeira identifica o módulo dentro do Morpheus e, a segunda, envia as mensagens que serão interpretadas pelo módulo.

```

{
    "moduleId": <moduleId> ,
    "moduleName": <moduleName> ,
    "moduleTopic": <moduleTopic> ,
    "unregister": <true|false>,
    "messages": [<message> ]
}
```

```

<message> =
"controlParameters":
{
  "parameter": <name> ,
  "value": <value>
},
"payload": [
  <key>: <value>
]
}

```

Ex.: Unregister a module and configure another

Action Request Messages As mensagens de action_request seguem o mesmo protocolo de mensagens, estabelecido anteriormente.

Data Transmission Messages As mensagens de data_transmission também seguem o mesmo protocolo de mensagens, estabelecido anteriormente.

6.1.8 Configurações

6.1.8.1 Configuração do *MQTT Mosquitto broker*

Em situações reais, cada casa terá uma instância do *Mosquitto Broker* rodando, independente de todas as outras, e aceitando conexões locais, somente. Entretanto, para que fossem realizados testes e simulações, a instalação e execução de uma instância em cada máquina diferente, localmente, seria inviável. Para tanto, foi configurada uma instalação em uma máquina remota - em servidor da Digital Ocean⁸ -, mas com configurações diferentes, uma para cada residência simulada. São executadas instâncias como processos *Daemon*, e vinculadas à portas diferentes - a partir da porta 8883 (conexão com criptografia), para Morpheus, e 1883 para módulos. Também foi criado um script em *bash* para iniciar o processo e ativar as portas no *firewall*. São necessárias as seguintes configurações, para a máquina remota.

1. Habilitar a restrição de tópicos na instância⁹. A restrição deve levar em conta

⁸Digital Ocean: <https://www.digitalocean.com/>

⁹Mosquitto Configuration <http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/>

as credenciais do dispositivo logado no momento (para definição do formato dos tópicos).

2. Os tópicos que finalizam em *s2m* devem ser exclusivamente restritos ao Morpheus. Nenhum outro dispositivo deve conseguir publicar nestes tópicos. O Morpheus pode publicar e ouvir todos os tópicos.
3. Os tópicos que finalizam em *m2s* são exclusivos de cada módulo. O *broker* saberá se um módulo pode se inscrever ou publicar no tópico de acordo com o seu número serial.
4. Para cada casa, os módulos devem se conectar a partir da porta 1883 (e.g. primeira casa → 1883; segunda casa → 1884). Essa porta não exige criptografia, mas deve exigir somente usuário e senha (que estarão vulneráveis).
5. O Morpheus será obrigado a se conectar a partir da porta 8883 (e.g. primeira casa → 8884; segunda casa → 8884), passando suas credenciais encriptadas.

6.1.8.2 Guia de instalação (Testado com Ubuntu 16.10 x64)

1. Utilizar o terminal para fornecer os seguintes comandos.

```
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
sudo systemctl enable mosquitto
```

2. Criar pastas para cada casa, em */etc/mosquitto/conf.d*, com os nomes *home<Número>.conf*. Deve se adicionar os arquivos *acl_list*, *m_home_<Número>.conf*, *passwd*. O conteúdo de cada um desses arquivos é mostrado abaixo (relativos à casa de número 1).

acl_list

```
# General section

# User specific section
## Morpheus
user adf654wae84fea5d8ea6
topic readwrite hw/#
```

```
# Client section
```

```
## Modules can write only to the topic with their username in
→ the m2s version
pattern write hw/%u/m2s
```

```
## Modules can only read to the topic with their username in the
→ s2m version
pattern read hw/%u/s2m
```

m_home_1.conf

```
password_file /etc/mosquitto/conf.d/home1/passwd
allow_anonymous false
acl_file /etc/mosquitto/conf.d/home1/acl_list

# General Listener
# When running in production, this should bind to localhost
port 1883
require_certificate false
use_username_as_clientid true

# Morpheus Listener
# When running in production, this should bind to localhost
listener 8883
cafile /etc/mosquitto/ca_certificates/ca.crt
keyfile /etc/mosquitto/certs/mosquitto.key
certfile /etc/mosquitto/certs/mosquitto.crt
require_certificate true
```

passwd

```
0002D3D7:135876
01344682:374028
000750A1:524708
001A1B07:321115
0014BB3E:147203
asd561asd5asd984faee:852456987
```

3. Para execução do *script*, basta utilizar o comando seguinte (na pasta onde o arquivo se localiza).

```
. start.sh
```

O conteúdo do *script* é mostrado na listagem seguinte.

```
#!/bin/bash

NUMBER_OF_HOMES=4
BASE_PORT_MORPHEUS=8882
BASE_PORT_MODULES=1882

echo "Olá, $USER! Bem-vindo ao MQTT - Hedwig"
echo $'-----\n'

echo 'Desativando o firewall...'
'sudo ufw disable > /dev/null'
echo 'Firewall desativado!'
echo $'-----\n'

echo 'Parando os processos do mosquitto...'
for each_instance_pid in `pgrep mosquitto`; do
'sudo kill -9 $each_instance_pid'
echo "Instância com PID $each_instance_pid eliminada"
done

echo 'Todos os processos do mosquitto parados'
echo $'-----\n'

echo 'Iniciando os processos do mosquitto...'
for count in `seq 1 $NUMBER_OF_HOMES`; do

echo "Iniciando Casa $count..."
'sudo mosquitto -c conf.d/home"$count"/m_home_"$count".conf -d'
echo "Casa $count iniciada!"
```

```

morpheus_port_number=$((BASE_PORT_MORPHEUS+count))
modules_port_number=$((BASE_PORT_MODULES+count))

echo "Adicionando $count ao firewall..."

'sudo ufw allow "$morpheus_port_number"/tcp > /dev/null'
'sudo ufw allow "$modules_port_number"/tcp > /dev/null'
echo "Adicionando $count adicionada ao firewall!"
echo ''
done

echo 'Todos os processos do mosquitto iniciados!'
echo $'-----\n'

echo 'Ativando firewall...'

'yes | sudo ufw enable > /dev/null'

echo 'Firewall ativado!'
echo $'-----\n'

echo 'Tudo pronto, divirta-se!'

```

6.1.8.3 Criação dos certificados

Por fim, deve-se criar certificados válidos, tanto para o *Broker Mosquitto*, quanto para as instâncias do Morpheus. Neste projeto, os certificados são gerados e auto-assinados. Entretanto, em um ambiente de produção, deve-se haver uma autoridade certificadora independente, para garantia da validade e segurança.

1. Criação da autoridade certificadora (key e certificado). Para a versão atual, a senha é hedwig123

```
openssl req -new -x509 -extensions v3_ca -keyout ca.key -out ca.crt
```

2. Mosquitto Key e Certificado. Foi adicionado o IP do servidor. O Common Name deve ser o IP do servidor

```
openssl genrsa -out mosquitto.key 2048
openssl req -new -key mosquitto.key -out mosquitto.csr
openssl x509 -req -in mosquitto.csr -CA ./ca.crt -CAkey ./ca.key -
    ↪ CAcreateserial -out mosquitto.crt -days 3650 -sha256
```

3. Morpheus Key e Certificado. Common Name será localhost

```
openssl genrsa -out morpheus.key 2048
openssl req -new -key morpheus.key -out morpheus.csr
openssl x509 -req -in morpheus.csr -CA ./ca.crt -CAkey ./ca.key -
    ↪ CAcreateserial -out morpheus.crt -days 3650 -sha256 -addtrust
    ↪ clientAuth
openssl x509 -in morpheus.crt -outform der -out morpheus.der
```

6.1.8.4 Senhas

Conforme mostrado anteriormente, no guia de instalação, 6.1.8.2, o arquivo de senha deve ser criado no formato `usuario:senha`. Deve-se, então rodar o comando seguinte, para que a senha não fique exposta em formato de texto:

```
sudo mosquitto_passwd -U passwd
```

6.1.8.5 Casos de teste para Controle de Acesso nos Tópicos *MQTT* entre módulos e nuvem

1. Conectar na porta 1883 sem usuário e senha.

Esperado: Falha de conexão

Resultado: Bem sucedido.

2. Conectar na porta 1883 com usuário e senha corretos.

Esperado: Permissão de conexão

Resultado: Bem sucedido.

3. Conectar com credenciais corretas e tentar publicar em tópico que não pertence ao seu usuário

Esperado: Não publicação

Resultado: Bem sucedido.

4. Conectar com credenciais corretas e tentar publicar em tópico que pertence ao seu usuário

Esperado: Publicação

Resultado: Bem sucedido.

5. Conectar com credenciais corretas e tentar ouvir de um tópico que não pertence ao seu usuário

Esperado: Não receber dados

Resultado: Bem sucedido.

6. Conectar com credenciais corretas e tentar ouvir de um tópico que pertence ao seu usuário

Esperado: Receber dados

Resultado: Bem sucedido.

7. Conectar com credenciais referentes ao Morpheus e tentar publicar ou ouvir qualquer tópico começando com hw.

Esperado: Publicação ou subscrição com sucesso

Resultado: Bem sucedido.

Para a realização das configurações acima, foram consultados materiais úteis, conforme a nota.¹⁰

6.2 Servidor na nuvem

6.2.1 Descrição

O servidor na nuvem é composto por um servidor WebSocket para comunicação entre clientes e casas, um banco de dados em memória para armazenar informações sobre conexões WebSocket ativas, uma API REST para acesso aos dados persistidos, um banco de dados não-relacional para armazenar dados dos sensores, módulos, Morpheus e usuários, um proxy reverso e um firewall.

¹⁰Topic Restriction:

[\(http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/\)](http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/)

Security Mechanisms:

[\(http://www.steves-internet-guide.com/mqtt-security-mechanisms/\)](http://www.steves-internet-guide.com/mqtt-security-mechanisms/)

Pub/Sub Client:

[\(https://pubsubclient.knolleary.net/index.html\)](https://pubsubclient.knolleary.net/index.html)

Para fins de prova de conceito, optou-se por prosseguir com uma arquitetura monolítica. A implementação da arquitetura de microsserviços aumentaria consideravelmente a complexidade do projeto, e seus principais benefícios não seriam tão bem aproveitados, visto que o sistema não seria colocado a provas de carga real no momento. Contudo, ressalta-se que o monolito que compõe o servidor na nuvem poderia sim ser implementado como um conjunto de microsserviços, o que seria uma evolução natural à medida que o sistema escala.

6.2.2 Requisitos

6.2.2.1 Requisitos funcionais

Comunicação

- O servidor deve permitir que Morpheus e aplicativos clientes se comuniquem via WebSocket.
- Morpheus podem enviar as mensagens provenientes dos módulos físicos, que são: `configuration`, `confirmation` e `data`. Também podem receber mensagens destinadas aos módulos físicos, que são: `action`, `configuration` e `data`.
- Morpheus podem receber mensagens de registro e remoção de módulo para definir quais dispositivos ele gerencia.
- Morpheus podem enviar mensagens de `report`.
- Os aplicativos cliente podem enviar as mensagens correspondentes a interações do usuário, que são: `action` e `configuration`. Também podem receber mensagens provenientes dos módulos físicos, que são: `confirmation` e `data`.
- Aplicativos cliente podem receber mensagens de `report`.
- Aplicativos cliente podem receber o status de conectividade dos Morpheus e receber notificações quando um Morpheus for desconectado.

Persistência de dados

- O servidor deve persistir dados de usuário, de configurações de Morpheus e módulo e de mensagens de dados (`data` e `report`).

Gerenciamento de dados

- O servidor deve oferecer uma API REST para leitura e escrita de dados de usuário, configurações de Morpheus e de módulos.

6.2.2.2 Requisitos não-funcionais

- O servidor deve permitir o estabelecimento de conexões HTTPS seguras e criptografadas.
- O firewall deve bloquear conexões em portas que não estão sendo utilizadas.

6.2.3 Tecnologias usadas

O servidor foi desenvolvido usando Node.js¹¹, um ambiente em tempo de execução para código em JavaScript. Sua arquitetura usa um modelo orientado a eventos e realiza a execução de comandos concorrentemente sem bloquear o servidor. Assim, servidores em Node.js conseguem alcançar uma melhor escalabilidade, suportando múltiplas conexões simultâneas sem impactos de performance.

Para a persistência de dados, foi escolhido o MongoDB¹², banco de dados não-relacional baseado em documentos. A facilidade de integração com JavaScript e Node.js, a similaridade dos documentos com objetos JSON e a natureza dos dados de sensores foram as motivações para sua escolha como banco de dados principal.

Contudo, não são apenas informações sobre usuários e dispositivos e dados coletados pelos sensores que precisam ser armazenados. Para gerenciar quais Morpheus estão conectados à nuvem e a quais aplicativos suas informações em tempo real devem ser enviadas, é usado o Redis¹³, banco de dados em memória. Redis é popularmente usado para fins como cache, mensageria e implementação de filas. No caso do Hedwig, ele é utilizado para armazenar informações de sessão, que são temporárias e requerem baixas latências para leitura e escrita.

Para implementar a comunicação entre aplicativos e casas, foi escolhida a biblioteca Socket.io¹⁴, que fornece uma API de alto nível para troca de informações bidirecional por meio de eventos. Além de abstrair a API de baixo nível do protocolo de WebSockets, o Socket.io já fornece eventos referentes ao status da conexão, facilitando o disparo de notificações caso o controlador de uma casa seja desconectado, e implementa um fallback para clientes que não suportam o protocolo de WebSocket. Por exemplo, se um usuário acessa um aplicativo por meio de um navegador antigo, a troca de dados continua sendo feita por meio de long polling.

¹¹<https://nodejs.org>

¹²<https://www.mongodb.com/>

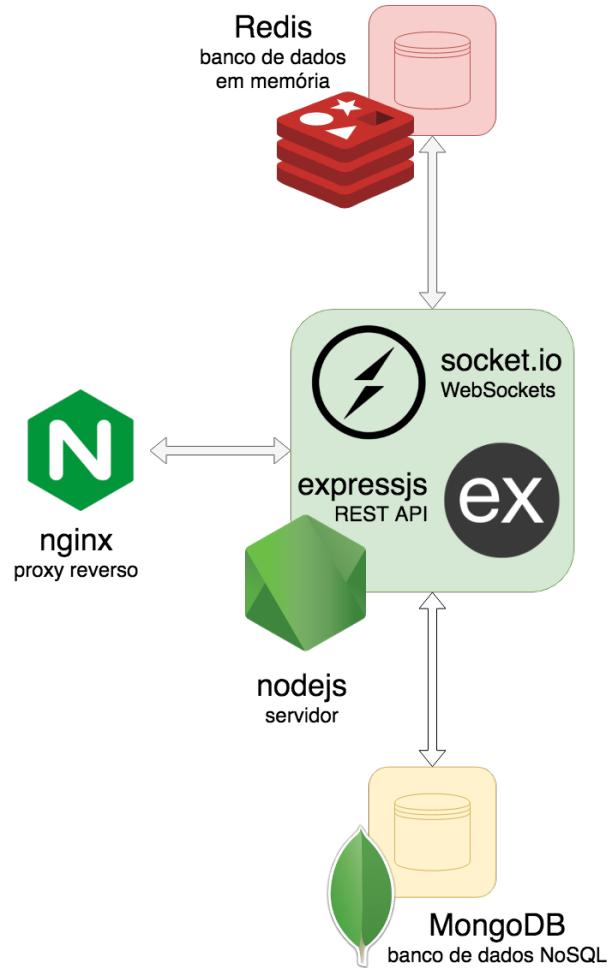
¹³<https://redis.io/>

¹⁴<https://socket.io/>

A arquitetura possui um proxy reverso que é responsável por enviar as requisições ao servidor em Node.js. Para isso, foi usado o nginx¹⁵, popularmente utilizado como servidor HTTP e proxy genérico para TCP e UDP. Ele permite a configuração de conexões seguras via HTTPS e dispensa a necessidade de delegar privilégios para acessar as portas reservadas 80 e 443 ao processo que roda o servidor Node.js.

Por fim, foi usada a ferramenta padrão do Ubuntu para firewall, ufw, que permite criar regras para bloquear tráfego IPv4 e IPv6.

Figura 17: Componentes e implementação na nuvem



6.2.4 Infraestrutura

Para hospedar o servidor do Hedwig, foi utilizado o serviço de computação na nuvem Digital Ocean¹⁶. Com ele, foi possível implantar o servidor em uma instância que roda Ubuntu 16.04.3 x64, com 1 CPU, 512MB de memória, 20GB de armazenamento de disco

¹⁵<https://nginx.org/>

¹⁶<https://www.digitalocean.com/>

SSD e 1000GB de cota disponível para transferência de dados. O data center que hospeda essa instância fica em Nova Iorque.

6.2.5 Segurança

O servidor na nuvem suporta conexões HTTPS, permitindo que navegadores verifiquem a autenticidade do servidor e garantindo a privacidade e integridade dos dados transmitidos. Para isso, foi usado o Let's Encrypt¹⁷, uma autoridade de certificação aberta, gratuita e automatizada. O Let's Encrypt usa o protocolo ACME (*Automatic Certificate Management Environment*) para automatizar a comunicação entre autoridade e candidato para assegurar a autenticidade deste e conceder-lhe certificados de forma rápida e prática. É realizado um teste para verificar que o candidato possui controle sobre o domínio e, então, é gerado um certificado válido por 90 dias que pode ser renovado a qualquer momento. Para usá-lo no servidor, basta acrescentar novas configurações ao nginx.

Além disso, outra medida de segurança foi usar um firewall para bloquear conexões nas portas TCP que não estão sendo usadas.

6.2.6 Operação

O serviço Keymetrics¹⁸ permite verificar se o servidor na nuvem está online, monitorar o uso de CPU e de memória, investigar a ocorrência de erros e realizar ações comuns, como reiniciar o processo do servidor, por meio de uma interface amigável. Investir em um sistema de monitoramento como esse auxilia tanto a manutenção preventiva como a corretiva, o que é essencial para um sistema de casa inteligente que tem a disponibilidade como requisito prioritário.

Figura 18: Monitoramento do servidor na nuvem



¹⁷<https://letsencrypt.org>

¹⁸<https://keymetrics.io/>

Outro ponto abordado é o uso de logs, arquivos que gravam eventos relevantes que acontecem no sistema. Eles podem ser usados para realizar a auditoria de falhas ocorridas e compreender melhor o funcionamento de um programa. Por questões de simplicidade, para classificar os eventos, o servidor na nuvem do Hedwig usa três dos sete níveis de severidade definidos pelo padrão syslog (NTWG, 2009): *error*, *warning* e *informational*. O servidor implementa um esquema de rotação de logs, criando arquivos individuais para cada dia, o que facilita o arquivamento de logs muito antigos e a pesquisa de eventos específicos. Com esse sistema, também é possível filtrar eventos de severidades diferentes em arquivos separados.

6.3 Aplicativo de dashboard

6.3.1 Descrição

O aplicativo desenvolvido é uma dashboard que permite ao morador da casa ver os dados dos sensores em tempo real, enviar requisições para os módulos realizarem alguma ação, receber confirmações de que essas ações foram realizadas e gerenciar seus dispositivos - Morpheus e módulos. Essa dashboard foi implementada com base nas características dos PWAs apresentadas no capítulo de Arquitetura a fim de permitir uma boa experiência de usuário tanto em smartphones como em computadores.

6.3.2 Requisitos

6.3.2.1 Requisitos funcionais

Realizar cadastro, login e gerenciamento de informações pessoais

- O usuário pode se cadastrar com seu email, nome e data de nascimento, criando também um nome de usuário e senha para acessar a dashboard.
- O usuário pode realizar login usando seu nome de usuário e senha.
- O usuário pode alterar as informações pessoais de seu cadastro.

Gerenciar Morpheus

- O usuário pode adicionar e remover controladores locais - Morpheus, sendo que para o cadastro basta adicionar o número de série do dispositivo.

- O usuário pode configurar o Morpheus, especificando se deseja que mensagens que não puderam ser enviadas por problemas de conectividade sejam mandadas assim que a conexão for reestabelecida.

Gerenciar módulos

- O usuário pode adicionar e remover módulos, sendo que para o cadastro deve-se adicionar o número de série do dispositivo, relacioná-lo ao Morpheus que ouvirá suas mensagens, dar um nome ao módulo e seus relês e especificar o seu tipo.
- O usuário pode configurar o módulo, especificando as configurações de conectividade, display e teste de auto-reset.

Receber dados e enviar ações em tempo real

- O usuário pode visualizar em tempo real os dados dos sensores de abertura, presença, temperatura, umidade e luminosidade do módulo básico.
- O usuário pode visualizar o estado dos relês e enviar ações para ligá-los e desligá-los.
- O usuário pode visualizar em tempo real os dados do estado do portão e do alarme do módulo de acesso.
- O usuário pode, no módulo de acesso, enviar uma ação para abrir o portão usando uma senha.

Alterar configurações avançadas dos módulos

- O usuário pode receber confirmações de que ações sensíveis foram recebidas pelos módulos corretamente.
- O usuário pode gerenciar o sensor de radiofrequência.
- O usuário pode enviar uma ação para sincronizar a hora do módulo.
- O usuário pode enviar uma ação para reiniciar o módulo (*soft reset*).

Visualizar estado das conexões

- O usuário pode ver o status de sua conexão com o servidor da nuvem e da conexão de seus Morpheus com a nuvem.

6.3.2.2 Requisitos não-funcionais

- O aplicativo deve ser responsivo e totalmente funcional nos navegadores mais recentes em suas versões desktop e mobile.

6.3.3 Tecnologias utilizadas

Para criar a aplicação web que demonstra a funcionamento do Hedwig, foi escolhido o React¹⁹, biblioteca open source de JavaScript mantida pelo Facebook. O React é conhecido por facilitar o desenvolvimento de aplicações *single-page*, renderizadas do lado do cliente, que permitem a atualização dinâmica da página de forma fluida e rápida, o que acaba enriquecendo a experiência do usuário. Possui uma linguagem declarativa e baseada em componentes, tornando-a altamente modularizável e reutilizável. Uma de suas características mais reconhecidas é o uso de um DOM virtual e um algoritmo de reconciliação que consegue identificar quais as partes da página que precisam ser renderizadas a cada interação com o usuário (FACEBOOK, 2017), melhorando a performance e permitindo maior fluidez em animações e mudanças visuais.

Outro ponto interessante para a utilização do React é que, com a biblioteca React Native²⁰ - uma extensão do React - é possível a geração de aplicativos nativos para iOS e Android. Assim, caso surja a necessidade de implementar uma nova funcionalidade que possua algum requisito que não pode ser contemplado por um *Progressive Web App*, mas pode ser atendido por um aplicativo nativo, pode-se usar o React Native. Isso diminui a necessidade de retrabalho e dispensa a necessidade de estudo aprofundado das linguagens e ambientes de desenvolvimento tradicionais de projeto de aplicativos nativos.

A fim de facilitar a implementação das interações com o usuário, usamos juntamente ao React a biblioteca Redux²¹, um gerenciador de estado global. Dessa forma, pretendemos facilitar o compartilhamento de informações entre diferentes componentes. A motivação por trás do Redux é facilitar a leitura e atualização do estado da aplicação, que, em sites *single-page* modernos, pode armazenar respostas do servidor, cache de dados e dados criados localmente que ainda não foram persistidos no servidor. O Redux é baseado em três princípios (REDUX, 2017):

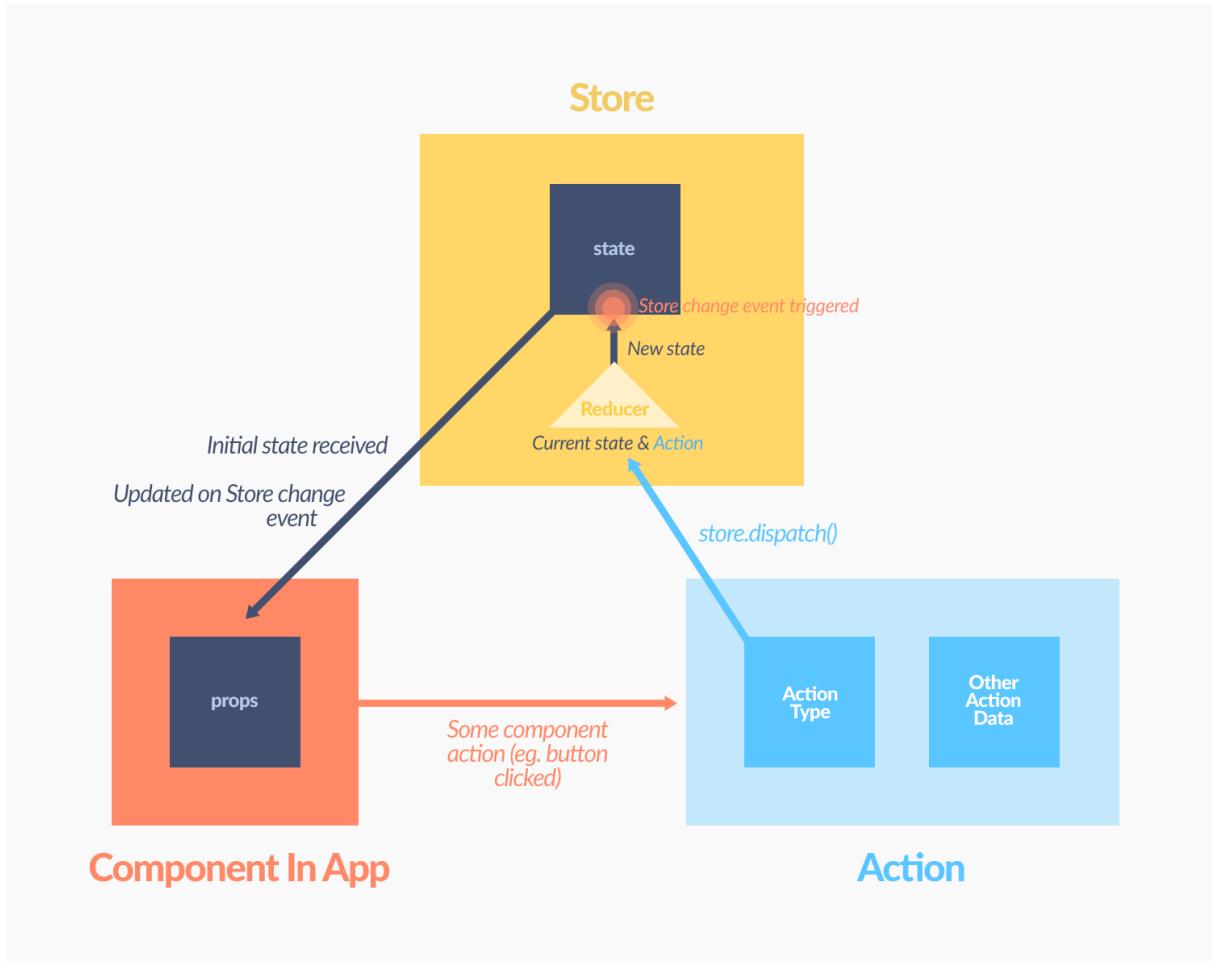
- O estado é o ponto único de verdade.

¹⁹<https://reactjs.org/>

²⁰<https://facebook.github.io/react-native/>

²¹<https://redux.js.org/>

Figura 19: Diagrama de funcionamento do Redux



Fonte: (FACEBOOK, 2016)

- O estado permite apenas a leitura - a única forma de alterá-lo é emitindo uma *action*.
- Alterações no estado devem ser feitas por funções puras - são os chamados *reducers*, que recebem o estado anterior e uma *action* e retornam o novo estado.

Para receber dados dos módulos em tempo real, foi usado um cliente do socket.io, framework que já foi discutido na seção do servidor na nuvem. O cliente de JavaScript já permite monitorar o status da conexão do aplicativo com o servidor, emitindo eventos em caso de desconexão ou reconexão. Assim, foi necessário criar *listeners* para os eventos customizados que criamos para nossos protocolos.

Uma das vantagens em usar JavaScript para criar a view da aplicação é sua versatilidade, visto que é uma linguagem multi-paradigmas que suporta programação imperativa, declarativa e orientada a objetos (MDN, 2016). Outro ponto é que ela é usada tanto para

desenvolvimento front-end como back-end, permitindo que o conhecimento acumulado na execução de um projeto ajude no outro.

Para aproveitar as funcionalidades e facilidades sintáticas das especificações mais recentes de JavaScript, usamos Babel²², um compilador capaz de converter as sintaxes novas e substituir as funções ainda não suportadas pelos navegadores com auxílio de *polyfills*. Muitas vezes, é usado o termo transpilador para referir-se ao Babel, visto que é um compilador de JavaScript para JavaScript, não deixando de emitir como saída uma linguagem de alto-nível. Assim, é possível usar ES6 - a versão mais recente de JavaScript - sem se preocupar com a compatibilidade do aplicativo com os navegadores que ainda não implementaram essa especificação completamente.

Para agilizar o desenvolvimento e prevenir falhas, usamos o *linter* dedicado a JavaScript ESLint²³. *Linter* é uma ferramenta para verificar o código e identificar erros de programação ou inconsistências de estilo (SUBLIMELINTER, 2016), o que possibilita produzir programas mais consistentes e menos suscetíveis a bugs.

Por fim, a pipeline de desenvolvimento do cliente usa o Webpack²⁴ como *module bundler*. O Webpack cria gráficos de dependência de todos os componentes da aplicação web - imagens, folhas de estilo, scripts - e então os processa transformando-nos em *bundles* ou pacotes, que nada mais são que arquivos estáticos.

6.3.4 Interface

6.3.4.1 Identidade visual

Para facilitar o desenvolvimento, foram usados os componentes do Material Design²⁵ da *Google*, que satisfazem várias necessidades básicas e casos de uso da criação de interfaces modernas. Para distinguir cada tipo de módulo, foram usados ícones e paletas de cores distintas.

²²<https://babeljs.io/>

²³<https://eslint.org/>

²⁴<https://webpack.js.org>

²⁵<https://material.io/>

6.3.5 Interações

6.3.5.1 Dados

Dados provenientes dos sensores dos módulos são atualizados em tempo real por meio de eventos do socket.io. Para situar o usuário, o instante de tempo em que a mensagem do módulo foi enviada fica visível no fim na página. Caso seja detectado que a conexão com o Morpheus se perdeu, isso é mostrado explicitamente pela dashboard a fim de evitar que o usuário olhe dados antigos demais.

6.3.5.2 Ações

Ao enviar uma ação, o estado local do aplicativo não muda até que seja recebida uma nova mensagem de dados ou de confirmação. Isto é, se um relê está ligado e o usuário pede para desligá-lo, o aplicativo só vai mostrar que o relê desligou quando o módulo o avisa disso. Isso evita que o aplicativo entre em estados inconsistentes com os módulos.

6.3.5.3 Conectividade

O aplicativo possui um indicador no canto superior direito indicando o status da conexão. Ele pode assumir três estados:

- Aplicativos e todos os Morpheus do usuário estão devidamente conectados ao servidor na nuvem
- Aplicativo está devidamente conectado ao servidor na nuvem, mas pelo menos um dos Morpheus não
- Aplicativo não está conectado ao servidor na nuvem

Além disso, caso o aplicativo perca conexão com a nuvem, são emitidas notificações na parte inferior da tela. São realizadas 10 tentativas de reconexão e o resultado - positivo ou não - também aparece como um alerta na tela. Caso não haja sucesso em nenhuma das 10 tentativas, o usuário é aconselhado a atualizar a página.

6.3.5.4 Aplicativo na tela inicial do dispositivo móvel

Usando um arquivo de manifesto, foi possível configurar como o aplicativo aparece na tela inicial de um dispositivo móvel como um smartphone. Diminuindo o número de

passos para o usuário acessar o aplicativo o encoraja a usá-lo mais vezes, além de criar uma sensação parecida com a de um aplicativo nativo.

6.3.6 Implantação

O aplicativo foi publicado com o serviço Surge²⁶, que permite a hospedagem de websites estáticos e oferece um domínio customizado. O Surge possui uma aplicação de linha de comando que permite a publicação de um diretório de arquivos HTML, CSS e JavaScript de maneira rápida e sem extensiva configuração. A dashboard está disponível em [⟨https://hedwig.surge.sh⟩](https://hedwig.surge.sh).

6.3.7 Segurança

O Surge usa a comunicação via HTTPS por padrão, oferecendo o suporte básico a SSL. Dessa forma, os dados transmitidos entre navegador e servidor podem ser criptografados, permitindo uma maior segurança em operações como cadastro, login e recebimento dos dados dos sensores.

6.3.8 Performance

Levando em consideração que o aplicativo pode ser acessado pelo celular em situações em que a qualidade da conexão não é a ideal, a otimização da dashboard para obter uma boa performance e baixos tempos de carregamento torna-se um ponto importante. Para isso, algumas medidas foram tomadas:

- Arquivos estáticos são servidos por uma CDN e podem ser armazenados no cache do navegador, medida que diminui o tempo de carregamento nas visitas subsequentes à dashboard.
- Arquivos são compactados em formato gzip antes de serem enviados ao cliente, o que diminui seu tempo de download.
- Arquivos HTML, CSS e JavaScript são minificados, diminuindo consideravelmente seu tamanho.
- As imagens são redimensionadas e otimizadas, diminuindo consideravelmente seu tamanho.

²⁶<https://surge.sh/>

- Não há redirecionamentos desnecessários.

6.4 App Backup

Para lidar com o caso de indisponibilidade do controlador local Morpheus, da rede local (roteador wireless) ou da conexão com a Internet, foi desenvolvido um aplicativo de *backup*. Esse aplicativo permite acesso direto ao módulo, através do endereço local (supondo que o dispositivo celular esteja na mesma rede) ou por conexão direta com o ponto de acesso do módulo (disponível todo o tempo, ou sempre que o módulo não consiga conexão com a internet, a depender da preferência do usuário).

6.4.1 Requisitos

Destacam-se os requisitos mais críticos do aplicativo backup:

1. Permitir acesso direto ao módulo em caso de indisponibilidade da rede wifi e internet;
2. Visualização de estado de sensores, e permitir a atuação em tempo próximo ao apresentado por botão físico;
3. Acesso local, por meio de endereço local, ao módulo, no caso de haver rede local disponível, mas sem acesso à internet;
4. Possibilitar ao usuário configurar rede WiFi, nome do módulo, cor do painel, offset de sensores, nome dos reles e regras de atuação (por rádio frequência, com senha ou não, por horário, por eventos dos sensores de presença e abertura,e tempo em que deve ficar ligado no caso de configurações automáticas);
5. Permitir configurar códigos RF (rádio frequência) múltiplos para sensor de abertura, atuação do relé 1 ou do relé 2;
6. Permitir ao usuário configurar controle de acesso e senha para atuação de um relé em específico;
7. Segregar permissões entre administrador e usuário (usuários não podem executar configurações, somente visualizar estados e atuar em relés sem senha)
8. Permitir visualização de vários módulos da residência;
9. Ter interface de fácil navegação, intuitiva.

6.4.2 Tecnologias usadas

Para o desenvolvimento do aplicativo backup, foram utilizados:

1. Módulo como servidor, utilizando bibliotecas de comunicação próprias do ESP8266, no caso de comunicação direta com o módulo, e uso do módulo como cliente, usando a mesma biblioteca;
2. Uso de ping para verificação de conexões e execução de rotinas para a correta configuração de estado do ponto de acesso (ligado se não conectado à rede), rotinas de desconexão e reconexão;
3. CSS para a implementação de interface amigável para o usuário, e segregação de configurações em níveis de navegação maiores para que configurações mais usadas sejam mais facilmente acessíveis a partir do menu principal do módulo;
4. Javascript para as rotinas de configuração e atualização do estado no menu principal;
5. HTML4 para o desenvolvimento da maioria das páginas;
6. EEPROM do módulo ESP8266, onde todas configurações de conexão, gerais e relés ficam armazenadas. Em caso de mudança desses parâmetros, ocorre sua persistência na EEPROM;
7. Bibliotecas próprias para interface com sensores e outros periféricos (DHT, I2C, por exemplo), comunicação em geral, além do WiFi e Access Point (PubSub para comunicação por MQTT).

6.4.3 Navegação

A partir da tela inicial, um cliente pode verificar todos os módulos presentes em sua casa, visualizar temperatura, umidade, luminosidade, sensores de presença ou abertura e apagar ou acender luzes (para atuadores protegidos com senha, o usuário deve acessar a página do respectivo módulo), numa interface configurável (o usuário pode configurar quais parâmetros observar nesse menu principal). A exibição da Figura 20 é própria para celulares, enquanto a exibição da Figura 21, com posicionamento configurável, simula a planta de uma casa (num cenário real, poderia ser a própria planta da casa do usuário), e é própria para desktops.

Figura 20: Telas principais do aplicativo backup



Figura 21: Visão geral de uma planta com o aplicativo backup



A partir do menu principal, podemos acessar o menu do módulo desejado (vide Figura 22). Nele, encontram-se informações de luminosidade, horário, data, temperatura,

umidade, sensor de presença (sensor 1) e sensor de abertura (sensor 2), além de controle de portão, lâmpada ou eletrodoméstico. Também exibe o nível de WiFi do módulo e a versão do firmware.

Figura 22: Menu Principal

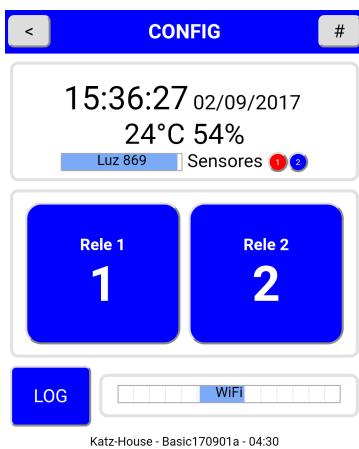


Figura 23: Teclado para digitação da senha



No caso de proteção de controle por senha, é exibido o painel numérico como na Figura 23. A cada requisição da página, o módulo manda um mapeamento das teclas diferente (por exemplo, de teclas A,B,C,... para (3,1), (9,2), ...). Após o usuário entrar com a senha (sequência de teclas do tipo A, B, C, ...), o módulo valida a sequência e autoriza o acionamento. Dessa forma, pessoas não conseguem copiar a senha ao visualizar a sequência de teclas do usuário, tampouco um invasor poderia copiar a sequência e utilizá-la para abertura logo em seguida, pois o mapeamento seria outro.

6.4.4 Configurações

A partir do menu principal do módulo, pode-se ter acesso ao seu log (para depuração, opção apenas para administradores) e um menu de configurações. Do menu de configurações, pode-se alterar o modo do display (entre 3 opções), e as cores do módulo, conforme opções anteriormente citadas.

Ainda do menu de configurações, pode-se configurar alertas e alarmes (sonoros a partir dos módulos, e pela internet através do provedor gratuito Blynk e e-mail), relés (possibilidade de auto ligar a partir de um sensor específico, ou de ser acionado a partir de um sinal de rádio - usualmente um controle remoto ou até um sensor de abertura adicional, ou mais de um), e o log (quais parâmetros serão persistidos e mandados para a nuvem). Também pode-se acessar o menu de Ferramentas, onde é possível realizar testes de auto reset (para verificação do funcionamento do circuito antitravamento, que age em

cerca de 30 segundos), reiniciar o módulo, desconectá-lo, voltar à versão de fábrica (versão implementada em software, enquanto a versão em hardware é realizada por meio de botão oculto), e atualizar o firmware (apenas disponível para administradores). Ao acessar a atualização de firmware, escolhe-se a opção, que mostra a versão atual e, após escolha do arquivo, a versão a ser inserida.

É possível, também, acessar as configurações avançadas 106. Nela, podemos configurar um offset para temperatura e umidade (de preferência realizados a partir de um medidor confiável para calibração).

Do menu de configurações avançadas, podemos configurar os sensores e controladores de radio frequência (sem fio), aspectos de conectividade do servidor gratuito Blynk e possivelmente trocar a rede WiFi em que o módulo está conectado. Ainda, para administradores, há a opção de configurar os usuários que têm acesso ao módulo.

Ainda pode-se trocar o nome do módulo, e ativar/desativar o ponto de acesso (para acesso direto ao módulo), além de configurar o nome (SSID) e senha de sua rede WiFi.

Demais ilustrações, referentes às configurações, estão disponíveis no Anexo D.

6.4.5 Abertura de porta do roteador

Para acessar o módulo/menu remotamente, podemos usar a abertura de porta (NAT/-virtual servers), configurável nas páginas de configuração dos roteadores. Maiores informações para essa configuração podem ser encontradas nos manuais dos roteadores. (Observe que há uma segurança menor envolvida com essa configuração). Há a abertura da porta para acesso remoto sem a necessidade de serviços em nuvem, para usuários que podem lidar com um nível de segurança mais baixo.

Figura 24: Página de configuração TP Link

Adicionar ou modificar uma entrada de servidor virtual	
Porta de serviço:	8030 <small>(XX-XX ou XX)</small>
Porta interna:	80 <small>(XX, insira um número de porta específico ou deixe em branco)</small>
Endereço IP:	192.168.0.30
Protocolo:	TCP
Ativar:	Ativado
Porta de serviço comum:	--Selecione--
<input type="button" value="Salvar"/> <input type="button" value="Anterior"/>	

Obs.: Caso sua operadora só forneça o CGNAT, a abertura de porta por parte do usuário não será possível.

6.4.6 Controle remoto

Para que o controle remoto seja corretamente configurado, são necessários os seguintes passos.

1. A partir da página inicial do módulo, entre em # → Configurações Avançadas → RF433
2. Configure o Sensor de Abertura “Sulton” no modo 2, para mandar sinais diferentes de abertura e fechamento. Consulte o manual do fabricante.
3. Aperte +, espere até a página indicar “Aguardando”, e abra o sensor de abertura. (Podemos incluir diversos sinais para o controle do mesmo relé, abertura ou fechamento de sensor). Aperte “OK” no canto superior direito da página, para salvar suas configurações.
4. Repita o passo 3 para o fechamento do sensor de abertura. **Cuidado:** O sensor de abertura repete algumas vezes o mesmo sinal. Aguarde alguns instantes entre gravar a abertura e o fechamento para não gravar o mesmo sinal (isso pode ser verificado na série numérica que aparece gravada. Se estiver igual, temos um problema).
5. (a) A partir da página inicial do módulo, entre em # → Configurações Avançadas → RF433.
- (b) Aperte + (ao lado do rele 1 ou rele 2), espere até a página indicar “Aguardando”, e abra o sensor de abertura. (Podemos incluir diversos sinais para o controle do mesmo relé, abertura ou fechamento de sensor).
- (c) Aperte ”OK” no canto superior direito da página, para salvar suas configurações.

6.4.7 Notificações

Para que as notificações sejam ativadas, utilize os próximos passos.

1. Para que o suporte consiga acessar remotamente o módulo e realize a coleta de dados, acesse # → Configurações Avançadas.→ Blynk
2. Insira o Auth Token (e.g. aa7a6dc1170640f08e951ed8cd2198a1).

3. Selecione Notificações ao iniciar: Sim, e WiFi: Sim.
4. Aperte em OK, no canto superior direito, para salvar suas alterações.

6.4.8 Offset Temperatura e Umidade

1. Para que o suporte consiga acessar remotamente o módulo e realize a coleta de dados, acesse # → Configurações Avançadas.→ Temperatura e Umidade.
2. Efetue a calibração do equipamento usando uma referência externa.
3. Aperte em OK, no canto superior direito, para salvar suas alterações.

6.4.9 Hard Reset

1. Ligue na tomada
2. Abra a tampa do módulo, e retire o isopor que cobre o sensor de temperatura azul;
3. Localize o botão (“pushbutton”), após retirar o isopor;
4. Pressione o botão até ouvir 6 beeps;
5. O módulo retornará para a configuração de fábrica. Veja as seções a seguir para sua configuração inicial.

6.4.10 Setup inicial

1. Primeiro, conecte à rede Wifi do Módulo, com nome CONFIG (senha: A senha da rede CONFIG é 12345678)
2. Abra um navegador e vá ao endereço 192.168.4.1 para entrar na página de configuração. Entre com as credenciais (login: admin e senha: 1234);
3. Espere até que as redes disponíveis apareçam, selecione-a e forneça sua senha, para conectar o módulo à internet.
4. Observe o endereço local 192.168.0.X que aparecerá no lcd do módulo. Caso não consiga ver, realize o passo 7 e use uma ferramenta como o “Zentri” (aplicativo Android) para descobrir em que endereço o módulo entrou. Para ver na tela novamente, pode tirar o módulo da tomada e ligá-lo novamente

5. Espere o módulo reiniciar (continue na rede CONFIG), e aguarde até a página recarregar. Insira o nome do módulo e depois aperte “Salvar e Reiniciar”.
6. Conecte-se novamente na rede WiFi da sua casa.
7. Entre no endereço 192.168.0.X que foi mostrado no módulo, clique em # e então em “Reiniciar Busca”, Aguarde até que todos os módulos sejam descobertos.
8. Retorne para a página anterior (usando o “j” no canto superior esquerdo). Você verá o menu principal da casa, e daí poderá controlar reles, visualizar dados coletados pelos módulos e entrar em cada módulo. O menu é personalizado na página anterior (ao pressionar o “#” no canto superior direito da tela).

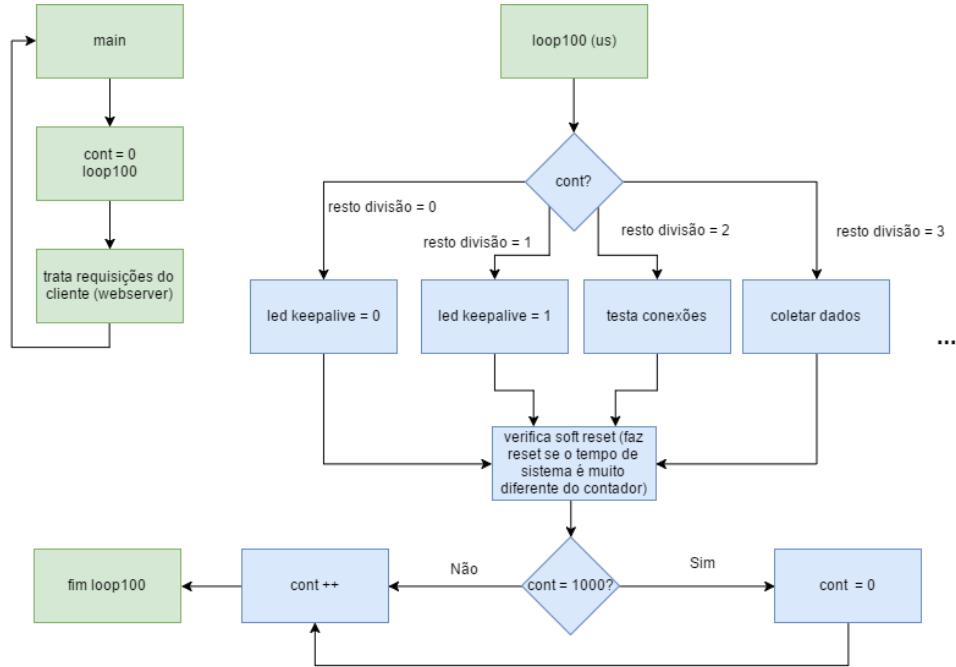
6.5 Módulos

6.5.1 Rotinas

6.5.1.1 Multiplexação no tempo

Para tratar indisponibilidade dos módulos devido a tentativas de reconexão e conexão e requisições não gerenciadas, e aumentar a disponibilidade, além do circuito antitravamento e *hard reset*, as diversas rotinas - desde configuração inicial, reconfigurações, coletas de dados, atuar por meio de relés, até conexão, desconexão, reconexão e envio de dados - foram multiplexadas no tempo da seguinte forma:

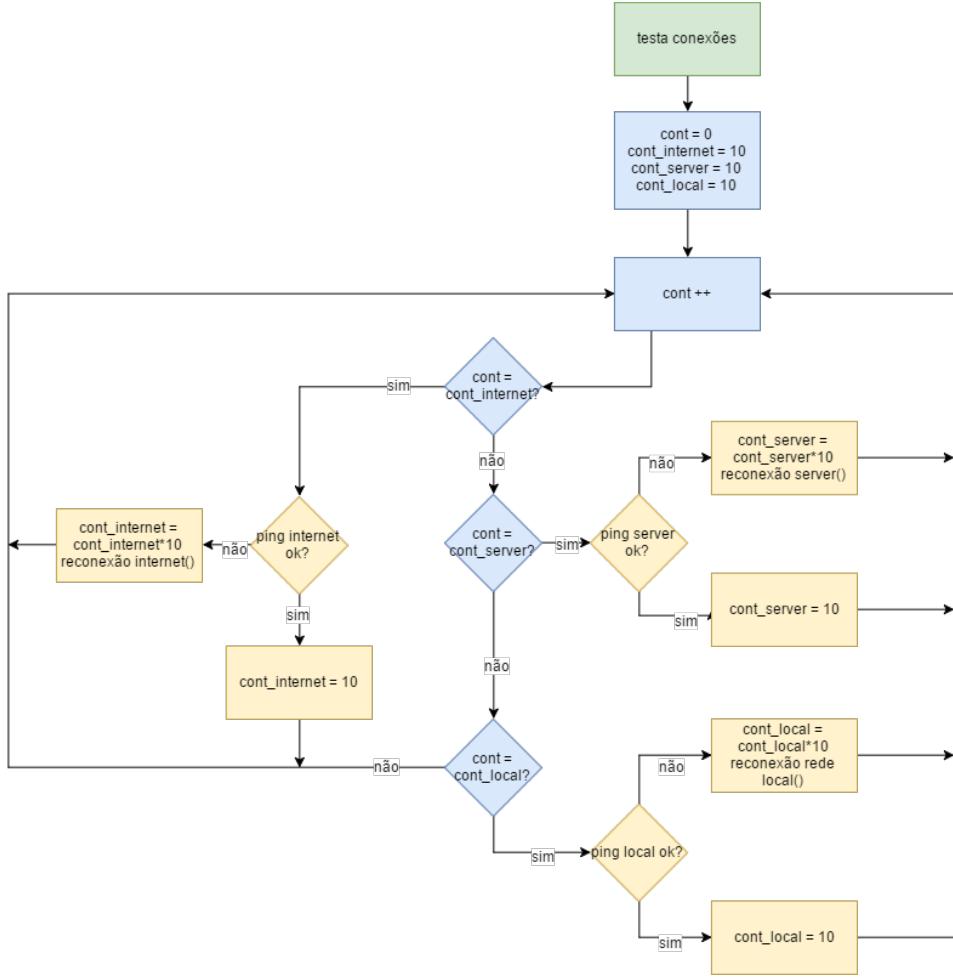
Figura 25: Rotina de multiplexação de procedimentos no tempo



6.5.1.2 Tratamento de indisponibilidade

Nos casos de indisponibilidade de Internet, servidor ou rede local, o seguinte procedimento foi adotado (observe que a indisponibilidade do próprio módulo é tratada pelo circuito antitravamento):

Figura 26: Tratamento de indisponibilidade de recursos



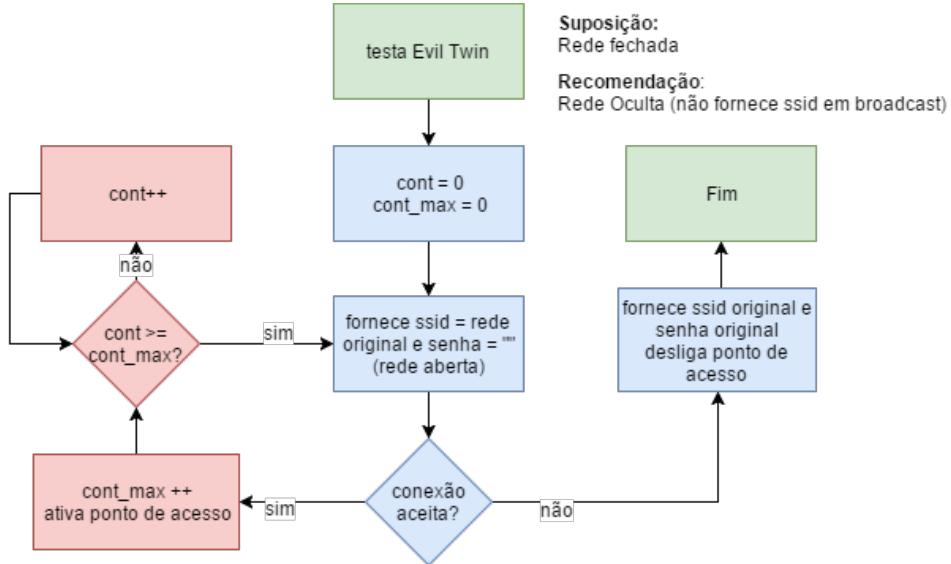
Com esse procedimento, as tentativas de reconexão à Internet, servidor e rede local estão segregadas e com tentativas realizadas em intervalos de tempo sucessivamente maiores. Desta forma, conseguimos gerenciar esses procedimentos, já que o nível de processamento é baixo.

6.5.1.3 DoS Local (*Evil Twin*)

No caso de ataque de *Evil Twin* - no qual uma rede mal intencionada, usualmente aberta, usa o mesmo SSID da rede original, com o objetivo de obter a senha - o sistema pode ficar indisponível até ao nível local. Módulos podem se conectar à rede mal-intencionada e ficarem somente com as funcionalidades offline, como acionamento de lâmpada por botão físico acoplado ao módulo. Outro problema é a queda da rede por interferência de radio frequênciа ou outro mecanismo utilizado pelo usuário mal intencionado para que os clientes se desconectem, tentem reconexão e forneçam a senha da rede.

Para mitigar esses riscos, os módulos executam o seguinte procedimento:

Figura 27: Tratamento de ataque de DoS Local



6.5.1.4 Comunicação por MQTT

Para o desenvolvimento da comunicação por MQTT com o Morpheus no módulo, houve o uso da biblioteca PubSub para Arduino.

Para cada módulo, temos de fábrica id e senhas próprias, além de configurações de porta e endereço do morpheus locais padrões.

Houve o controle de comunicação da seguinte forma:

1. Só há tentativa de conexão MQTT em caso de houver conexão do WiFi e o horário interno estar configurado (em caso contrário, podemos provocar travamento do módulo ou envio de mensagens sem timestamp);
2. Configuração de callback e servidor MQTT a cada reconexão (este ponto foi crítico para o bom funcionamento da comunicação);
3. No callback (recebimento de mensagens, tratamento (“parse”) da mensagem recebida, através de obtenção de seu tipo e redirecionamento para rotinas específicas para obtenção dos parâmetros de interesse de cada mensagem);
4. Envio de mensagens de estado a cada 1 minuto, ou quando houver mudança brusca em um dos sensores (presença, abertura, umidade ou temperatura) ou então requisição de atuação (nesses casos de envio rápido, a mensagem é inserida no início da fila de saída, e enviada logo em seguida, em até um segundo);

5. Após o tratamento inicial das mensagens (obtenção dos parâmetros de interesse), persistência nas variáveis internas e gravação da EEPROM para que configurações e estados executados no aplicativo backup ou pela dashboard sejam refletidos dos dois lados, tornando transparente ao usuário o uso de qualquer um dos aplicativos, e para que suas funcionalidades estejam integradas;
6. Ainda, em fase final, após a persistência de variáveis na EEPROM, a inclusão na fila de saída de mensagens de confirmação para o servidor local;
7. Uso da fila de saída para o envio periódico das mensagens de estado.

Figura 28: Exemplo de estado da fila de saída de mensagens MQTT do Módulo



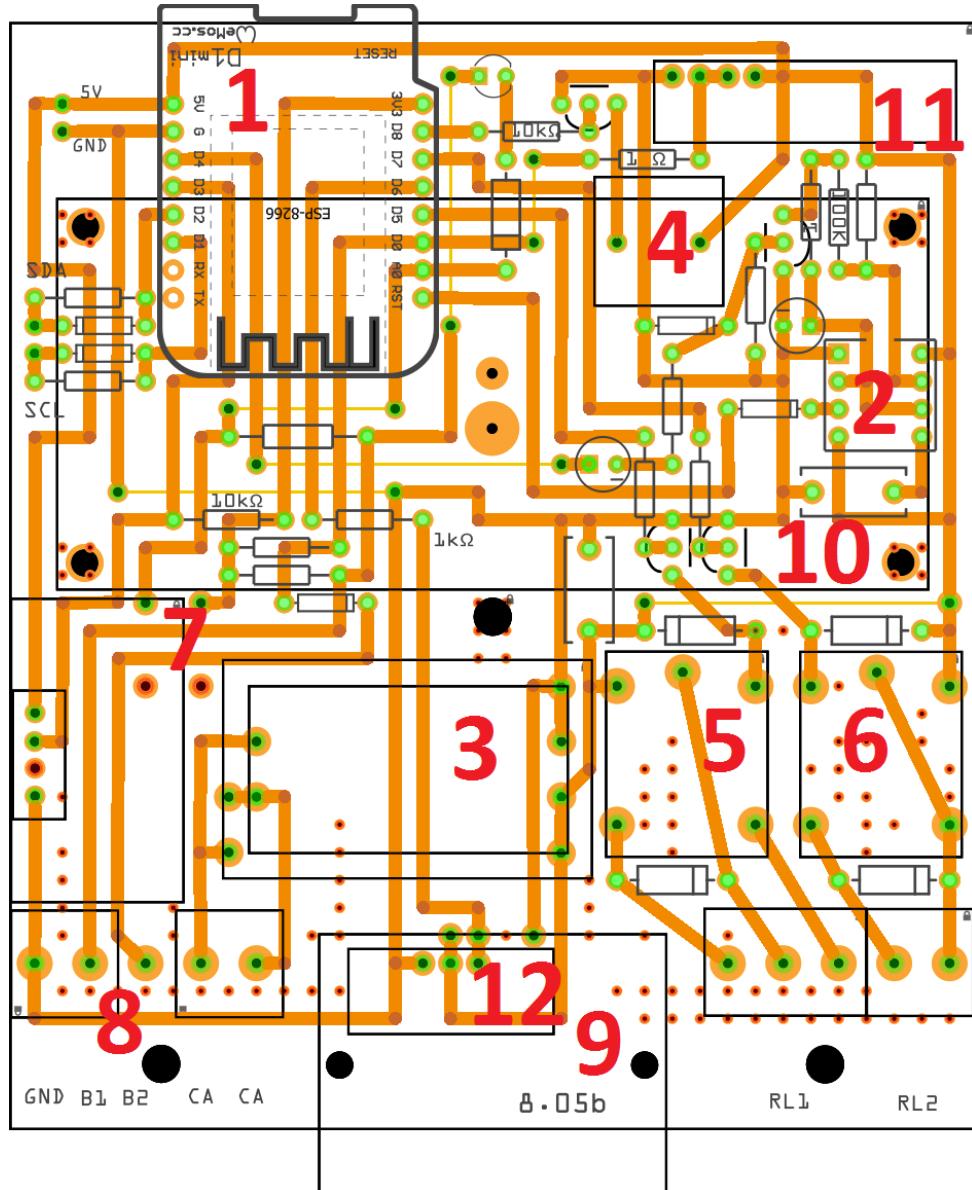
Mensagens de estado periódicas (azuis) são inseridas no final da fila. No caso de acionamento, uma mensagem de estado (vermelha) é inserida no início da fila, e em caso de configuração, uma mensagem de confirmação também é inserida no início da fila.

Por fim, vale destacar a necessidade de alteração da biblioteca PubSub para suportar mensagens maiores (em caso de impossibilidade de envio de mensagens maiores, todo o protocolo deveria ser alterado para comportar mensagens mais compactas ou então deveria haver o uso de algum mecanismo de codificação/compactação em conjunto com o protocolo desenvolvido).

6.5.2 Diagrama

Abaixo está o diagrama do circuito impresso (PCB).

Figura 29: Diagrama PCB do Módulo Base



1. Wemos D1 mini

2. Astável 555

3. Fonte 5V 3W

4. *Buzzer*

5. Relé 1

6. Relé 2

7. Hard Reset

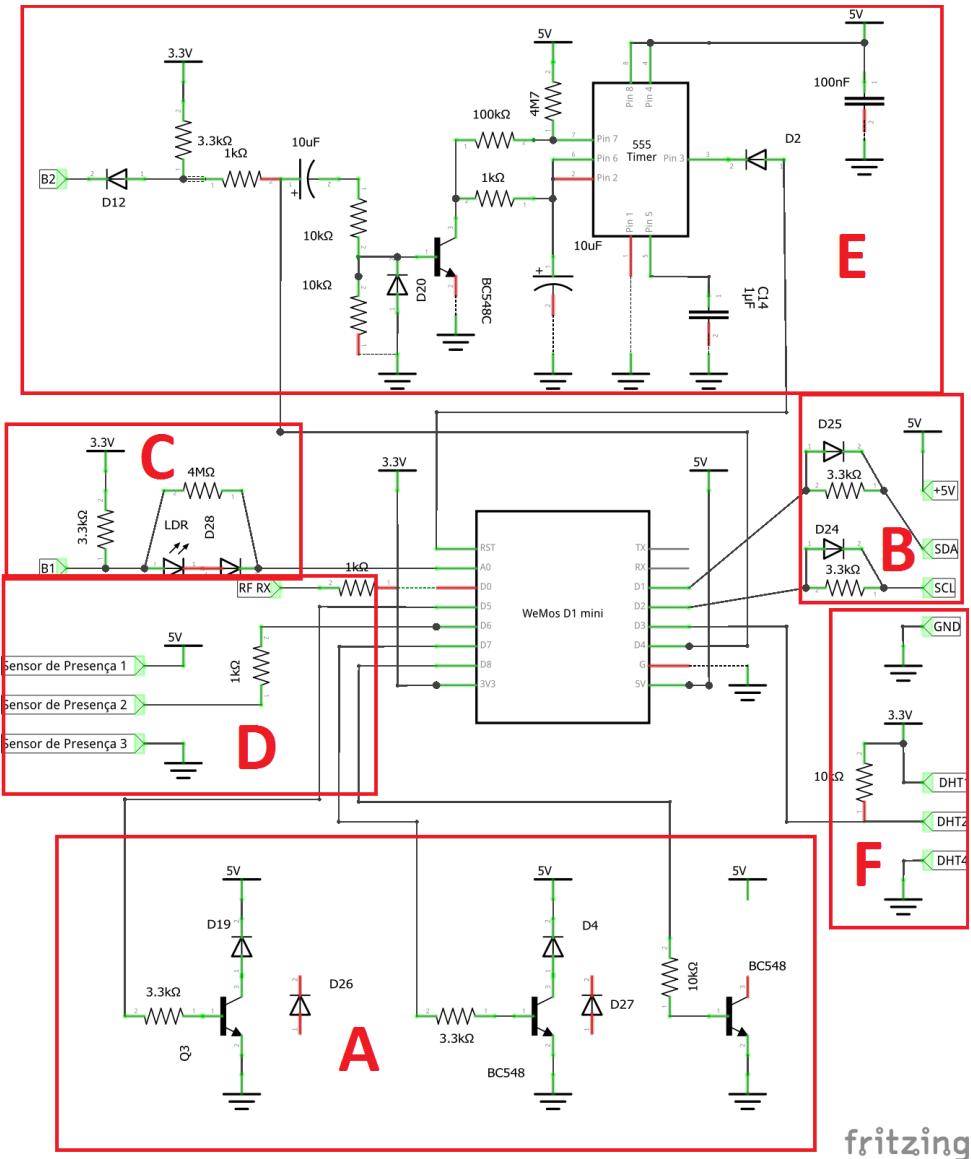
8. Botões

9. Presença

10. RF-RX

11. RF-TX

Figura 30: Diagrama Elétrico do Módulo Base

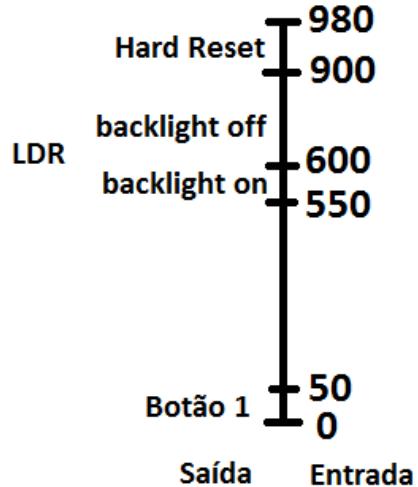


A - Saídas Circuitos simples de transistor para acionamento de relés (para lâmpadas) e buzzer.

Proteção 3V3 5V Como o display trabalha com tensão de 5V, há proteções com diodos para não danificar as entradas digitais do Wemos D1 mini, que trabalha com tensão de 3V3.

3 Entradas em A0 O circuito tem como entradas um botão (para acionamento do relé 1), o LDR (para chaveamento do backlight do display), e um outro botão para hard reset do dispositivo, todos numa entrada analógica, cujo mapeamento E/S é da seguinte forma:

Figura 31: Entradas Em A0



Presença ou RF-TX A entrada digital D6 é usada exclusivamente como entrada do sensor de presença PIR ou receptor RF.

Astável 555 para Hard Reset e Botão A porta D6 é usada como LED *keep alive* do módulo. Sua demora ao piscar indica que o módulo está travado ou demorando muito para processar algo, o que não deveria acontecer, uma vez que os procedimentos estão multiplexados no tempo, de acordo com seus tempos limite. Dessa forma, conectamos essa saída a um circuito antitravamento, que executa o reset nos casos mencionados, de travamento ou *timeout*.

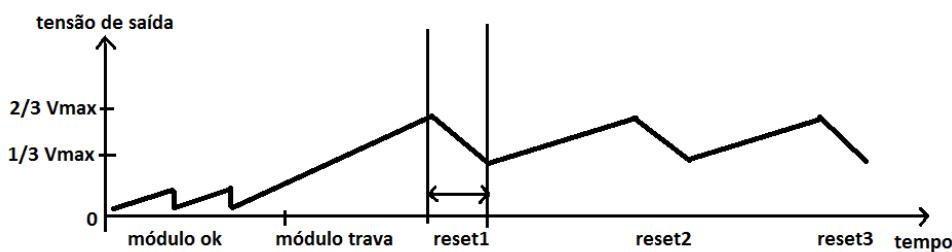
O primeiro capacitor tem como objetivo desacoplamento DC, de forma que a entrada do circuito envolvendo o astável 555 seja somente AC. Assim, travamentos em 0 ou 1 indicam travamento.

Enquanto o LED pisca em intervalos esperados (regularmente), o transistor conduz e mantém uma saída dente de serra muito próxima de 0. Quando o módulo trava, o transistor não conduz mais, e a saída passa a oscilar entre 1/3 e 2/3 da tensão total (observe que o carregamento é feito pelo resistor de 4M7, muito maior que o resistor de 100k, fazendo com que o tempo de carga seja muito maior que o tempo de descarga, uma vez que esses tempos são diretamente proporcionais à constante de tempo dos circuitos RC, que é dada pelo produto do $R \cdot C$). Durante a descarga,

o reset da placa é realizado. Observe que os tempos foram ajustados pelos valores dos componentes discretos, para que o tempo entre resets sucessivos seja menor que o tempo necessário para o módulo voltar a funcionar após um reset.

Segue abaixo uma ilustração sobre o funcionamento do circuito.

Figura 32: Funcionamento do Circuito de Antitravamento



DHT11 Entrada D3 é ligada a uma montagem básica para leitura de umidade e temperatura através do periférico DHT11.

6.5.3 Montagem

As fotos e comentários seguintes descrevem o processo de montagem física dos quatro módulos usados neste trabalho. Além destes, outras versões também foram construídas anteriormente no decorrer do projeto, instaladas na residência de um dos membros do grupo e usados para coleta de dados (que, por se tratarem de protótipos, não tem sua montagem completamente documentada tampouco, uniformidade como os módulos seguintes). Ao final, também consta o procedimento utilizado para validação dos módulos após sua construção, que contribuiu para a identificação de ligações não realizadas e outros problemas de montagem.

Figura 33: Evolução do Hardware (de fevereiro/17 a setembro/17)

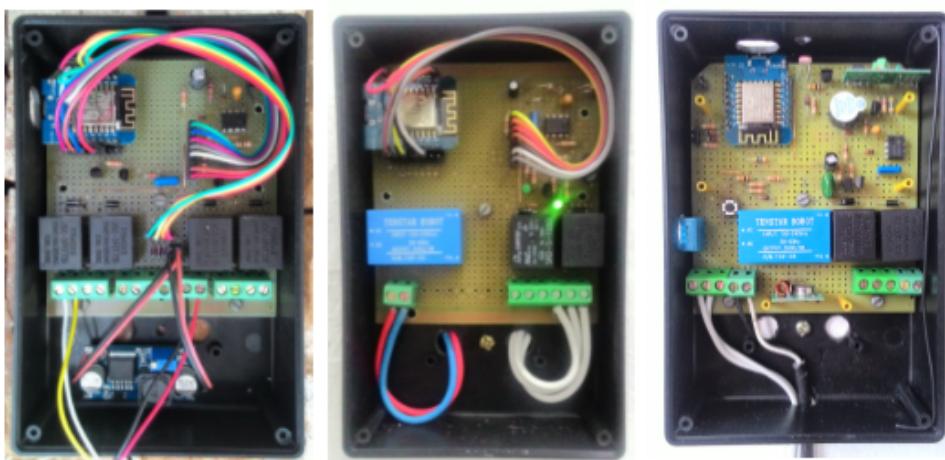
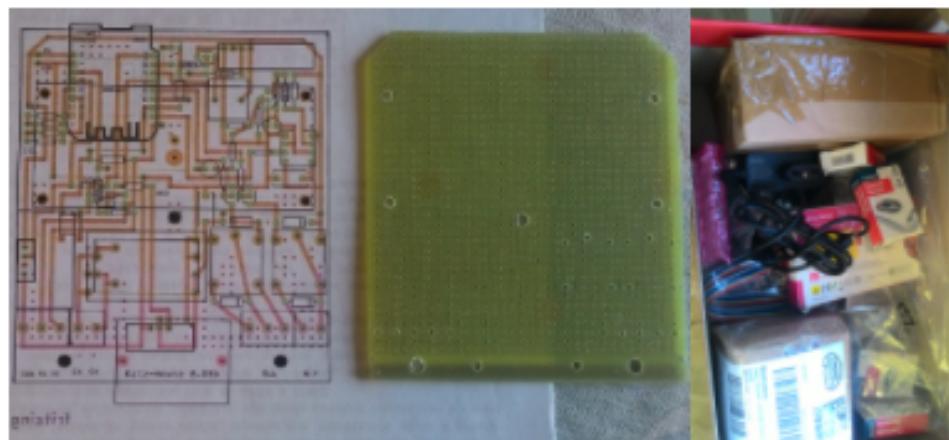


Figura 34: Evolução da Caixa de Proteção e Display

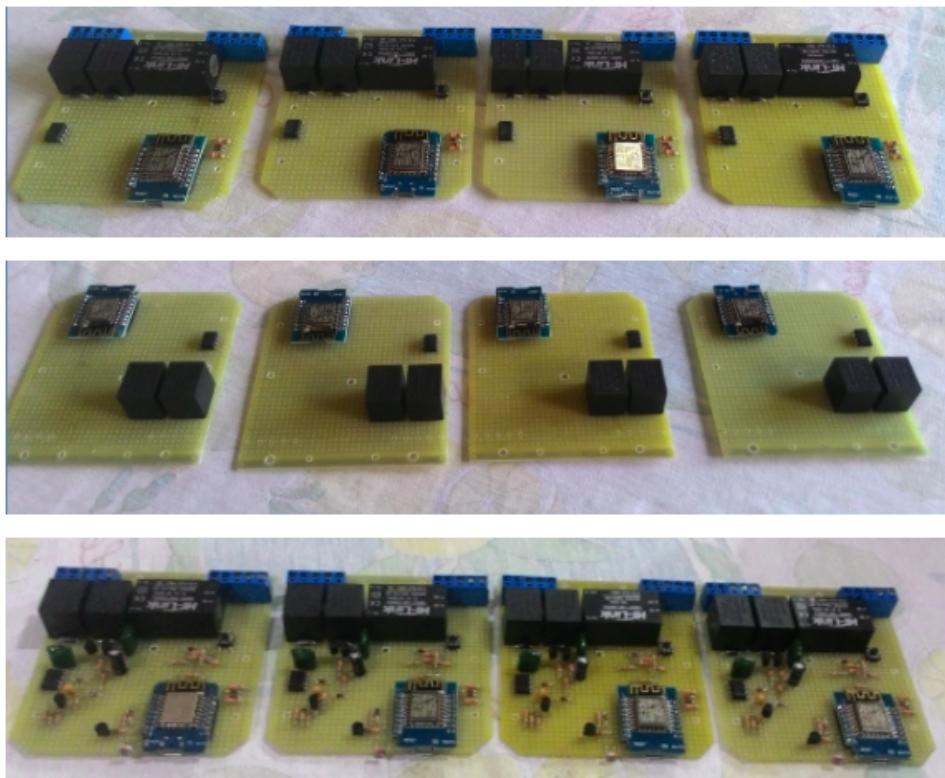


Figura 35: Materiais e preparação da placa padrão



1. Primeiramente, a partir do esquemático em escala real, foram cortados e realizados furos na placa padrão, conforme figura acima.

Figura 36: Posicionamento dos Componentes



2. Em seguida, foram posicionados os componentes, conforme mostra a figura acima.

Figura 37: Preparação dos displays com o I2C e soldagem



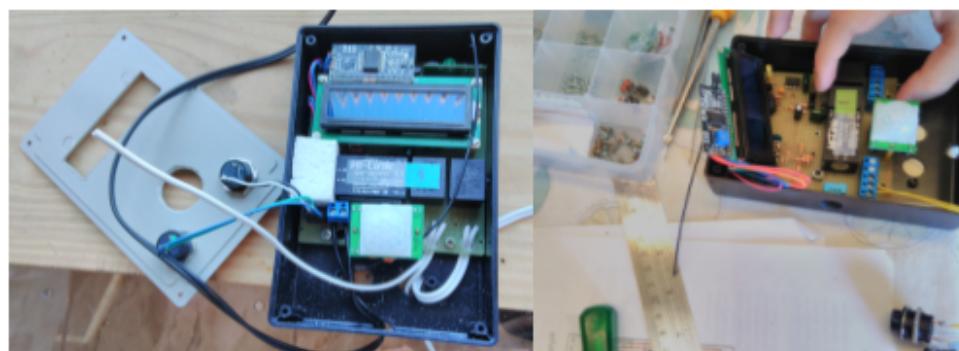
3. A próxima etapa é soldar as trilhas por baixo conforme o diagrama. Também é necessário soldar o I2C com o display. Essa é a etapa mais demorada, que poderia ser facilmente operacionalizada ao adotarmos placas de circuito impresso, o que aumentaria muito a capacidade de montagem.

Figura 38: Preparação das caixas para os módulos



4. Prossegue-se para a marcação das caixas para uso das furadeiras e lixas (com ferramentas mais adequadas, essa etapa também poderia ser mais rápida e eficiente).

Figura 39: Caixa protetora; ligação dos botões e cabos de força



5. Com a caixa preparada, foi possível inserir as placas padrão com os componentes e trilhas. Resta fixá-los com parafusos, montar a sustentação do display e sensor de presença (também integrado nesse passo), isopor para isolar termicamente o medidor de temperatura e umidade DHT, além das ligações dos botões, cabos de força e fios dos relés.

Figura 40: Quatro Módulos Prontos



6.5.3.1 Montagem

1. Carregar Programa para testar módulo;
2. Realizar Setup da Conexão com sua rede WiFi;
3. Verificar com o multímetro se há curto em algumas ligações principais (terra, VCC);
4. Checar a alimentação da fonte e sua saída correta;
5. Realizar o Hard Reset ao apertar o botão atrás do isopor do DHT até ouvir 10 beeps
6. Fazer o passo 2 novamente (pois o módulo deve ter voltado à versão de fábrica). Logo em seguida fazer o teste de Auto Reset, que é o teste para verificar se o circuito antitravamento está funcionando. Para esse teste, é simulado uma pausa do sinal keep alive que o circuito baseado no astável 555 monitora;
7. Cobrir o sensor de presença. Verificar a inatividade no aplicativo. Descobrindo e verificar atividade;
8. Executar o passo 7 para o sensor de luminosidade (LDR) também;
9. Verificar com um medidor externo ou consulta a um site de previsões do tempo se as medidas de temperatura e umidade estão de acordo. Realizar ajuste (offset) no aplicativo se necessário;
10. Verifique o funcionamento dos botões, acionando-os um por um;
11. Checar o acionamento dos reles pelo aplicativo web também;

12. Após gravar o RF de um controle para os dois relés, testar seu funcionamento;
13. Verifique com um multímetro a saída dos relés (se troca de nível com acionamento pela página web, botões e controle RF).

7 Aprendizado de Máquina e Análise de Dados

Para o desenvolvimento de funcionalidades de aprendizado de máquina, será utilizada a linguagem Python, que possui diversos pacotes que facilitam sua utilização para implementar algoritmos de aprendizado, e funcionalidades para tratamento de dados. Além disso, é usada em vários outros âmbitos como cursos acadêmicos voltados ao ensino de programação e aplicações web, o que facilita a familiarização com o desenvolvimento nela.

7.1 Coleta e Análise de Dados

Um dos processos mais críticos para o sucesso de um projeto é a obtenção de resultados e dados, de onde se obterá conhecimento sobre o comportamento do sistema em atividade.

7.1.1 Coleta

Para a coleta de dados, foram usados um total de onze módulos, distribuídos em locais e residências diferentes, de modo a simular maior diversidade de utilização.

1. Aquário
2. Corredor
3. Lavanderia
4. Sala/Cozinha
5. Entrada
6. Caixa d'água
7. Victor
8. Jarinu

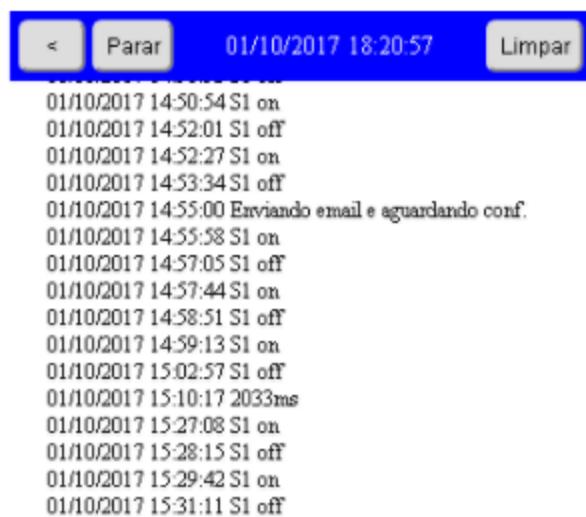
9. Daniela

10. Hugo

11. Gabriela

Os módulos de 1 a 7 estão localizados na mesma residência, em Santo André. Já os módulos 9, 10 e 11 estão em casas diferentes na Grande São Paulo, e o módulo 8 está localizado na cidade de Jarinu-SP.

Figura 41: Log na página do Aplicativo Backup



```

        < Parar 01/10/2017 18:20:57 Limpar
01/10/2017 14:50:54 S1 on
01/10/2017 14:52:01 S1 off
01/10/2017 14:52:27 S1 on
01/10/2017 14:53:34 S1 off
01/10/2017 14:55:00 Enviando email e aguardando conf.
01/10/2017 14:55:58 S1 on
01/10/2017 14:57:05 S1 off
01/10/2017 14:57:44 S1 on
01/10/2017 14:58:51 S1 off
01/10/2017 14:59:13 S1 on
01/10/2017 15:02:57 S1 off
01/10/2017 15:10:17 2033ms
01/10/2017 15:27:08 S1 on
01/10/2017 15:28:15 S1 off
01/10/2017 15:29:42 S1 on
01/10/2017 15:31:11 S1 off

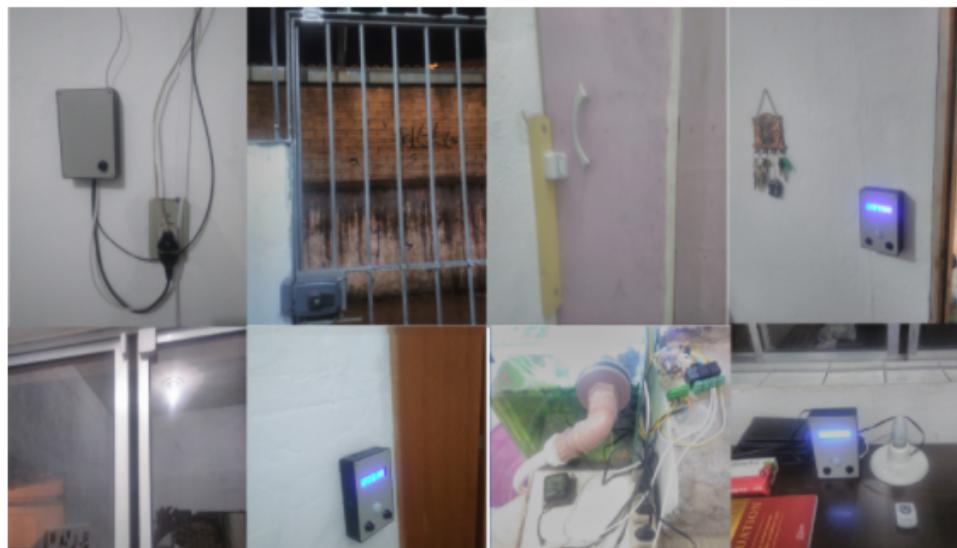
```

Figura 42: Módulo com cartão SD para coleta local de dados



Para os módulos em Santo André, foi utilizado um dispositivo com cartão SD para persistência de dados. Sua análise tem como objetivo principal acompanhar parâmetros relacionados à disponibilidade, e sua coleta é local.

Figura 43: Módulos instalados em Santo André



O módulo de Jardinu possui uma interface com o sistema de alarmes (já instalado no local) e tem como objetivo obter dados de presença e abertura de portas (teste de conceito para validar possível integração futura com parceiros estratégicos).

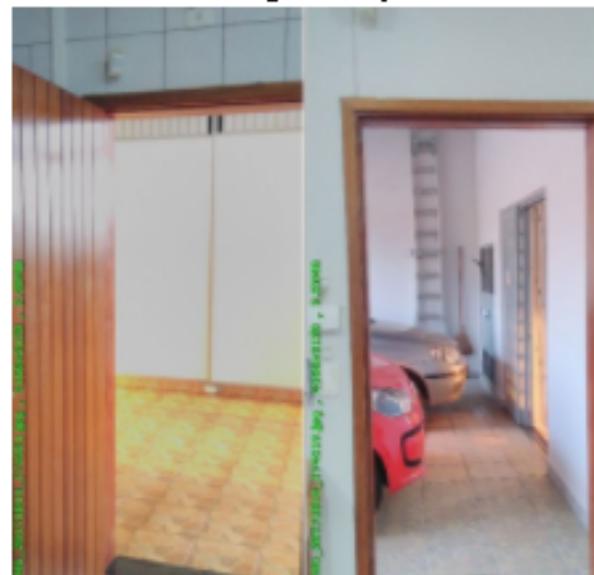
Figura 44: Controle do Sistema de Alarme e módulo de interface de Jardinu



Figura 45: Módulo Básico e sensor de presença no corredor de Jarinu

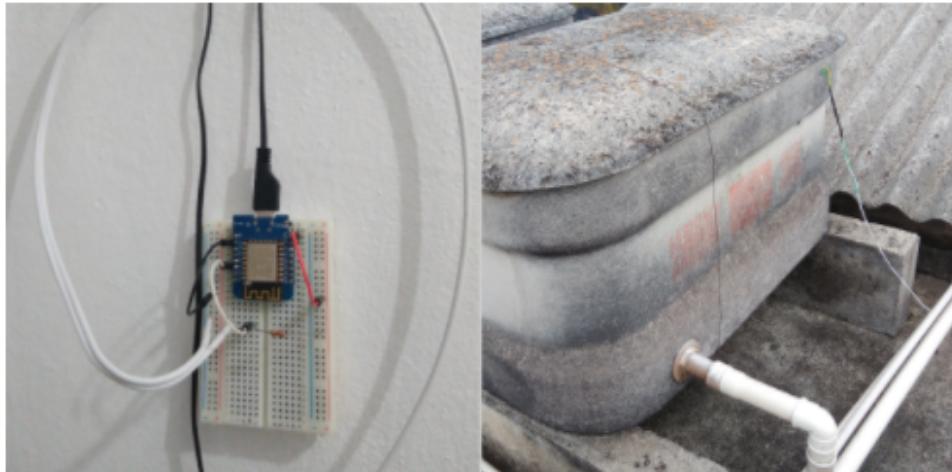


Figura 46: Sensor de abertura das portas da cozinha (à esquerda) e da sala (à direita)



Os demais módulos possuem coleta de dados a partir de persistência na nuvem e, passando pelo controlador local Morpheus, e possuem como principal objetivo prover dados para o Machine Learning.

Figura 47: Módulo instalado para coleta de dados do nível da caixa d'água



Conforme destacado, os dados a seguir foram coletados localmente e possuem informações sobre disponibilidade e uso das funções pelo aplicativo backup (também com escopo local).

De modo geral, temos a coleta dos seguintes dados:

Figura 48: Relação dos Módulos e Dados Coletados



Foram coletados dados desde o dia 10 de setembro de 2017 até o dia 13 de novembro de 2017, com processo semanal de backup. Os dados foram segregados, suas duplicatas foram removidas e dados dos diversos arquivos de log foram consolidados.

- **Dados do aquário:** temperatura da água, estado do aquecedor e estado da lâmpada usada como fonte de iluminação artificial;
- **Dados de disponibilidade:** memória livre disponível (“free heap”, em casos de pouca memória disponível, o espaço de dados invade o espaço de programa e ocorre travamento do ESP); “loops” (monitoramento do contador de loops de 1 segundo,

mas que devido a espera ou demasiado processamento demoraram mais que 2 segundos para ocorrerem) e monitoramento de desconexões, reconexões, updates do firmware dos módulos e reinícios;

- **Luminosidade:** monitoramento a partir de dado analógico de um LDR;
- **Temperatura e Umidade:** sensor DHT;
- **Presença:** Sensor de Presença PIR;
- **Uso dos Reles:** pelo botão físico presente no módulo, pela página web do aplicativo backup, automaticamente por regras configuradas pelo usuário, e por controle de radiofrequência (rf) (nos módulos usados para coleta, não havia a dashboard disponível);
- **Abertura do Portão e Estado do Portão:** monitoramento do estado do portão por sensor eletromagnético com fio, conectado diretamente ao módulo, e acompanhamento das aberturas por pessoa (fabio, victor ou admin - alguma das outras duas pessoas da casa), uso do sistema (aberturas pelo sensor x aberturas pelo sistema) e monitoramento de uma porta intermediária da escada, por meio de sensor de abertura sem fio (comunicação por radiofrequência);
- **Nível máximo da caixa d'água:** sensor de boia próximo ao nível máximo;
- **Porta da Sala (Estado) e Presença (Corredor e Cozinha) de Jarinu:** equipamentos próprios do sistema de alarmes.

Tabela 2: Análise Consolidada de Disponibilidade

Item	Aquário	Corredor	Lavanderia	Sala	Cozinha	Entrada
memória livre [bytes]	30278	27218	26990	26728	26237	
# (loops)	1251	771	76	925	1140	
# (reinícios)	34	18	35	20	91	
# (desconexões do WiFi)	15	26	24	27	77	

Ao analisar o aspecto de disponibilidade a partir da média da memória disponível (quanto maior, melhor), o número de loops de 1s que ocorreram em 2s ou mais (indicando a existência de procedimentos que estão atrasando a correta execução de rotinas periódicas, como aquelas de “keep alive” de comunicação), o número de reinícios e desconexões da rede WiFi (isentos de updates de firmware, isto é, eventos de reinício e desconexões desconsiderando aqueles causados por updates de firmware dos módulos).

Como o firmware do Corredor, Lavanderia e SalaCozinha é o mesmo (basic), conclui-se que:

- A versão aquário possui mais memória disponível, possui um menor número de desconexões da rede WiFi e número perto da média de reinícios. Contudo, possui maior ocorrência de loops de 1s maiores que 2s, indicando algum procedimento que está consumindo tempo do loop;
- A versão basic possui ocorrências de desconexões, reinícios e memória livre em valores médios;
- A versão entrada possui o maior número de reinícios e desconexões da rede WiFi (o que já era esperado, pois é o módulo mais longe do roteador da residência, que está em outro andar);
- A memória livre média das versões basic e entrada possuem valores próximos.

Tabela 3: Análise Consolidada de Uso dos Reles

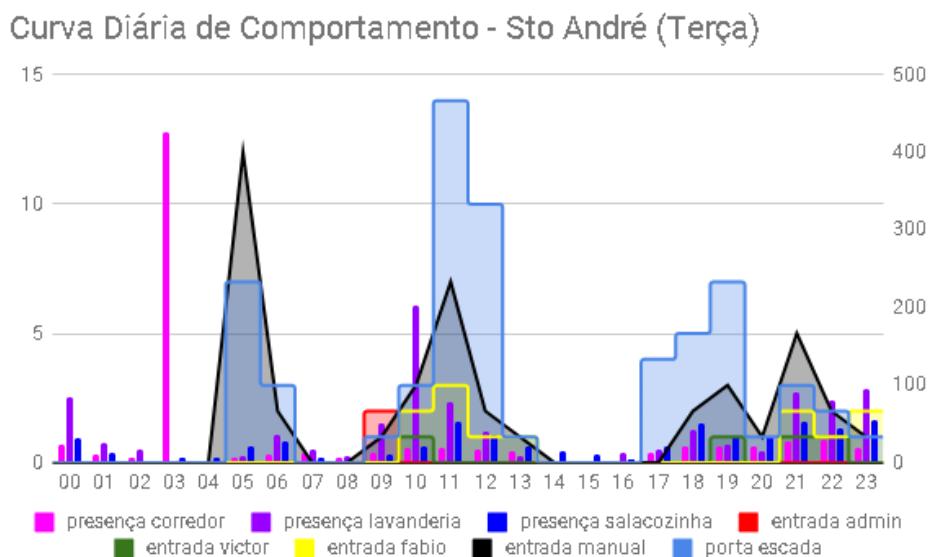
Nome	Corredor	Sabrina	Varanda	Escada	Sala	Cozinha
% Ligado	35.08%	22.91%	41.62%	6.89%	80.06%	60.99%
# Acionamentos	238	202	516	382	190	233
% Botão Físico	11.61%	87.96%	54.14%	13.57%	26.21%	20.33%
% Web	21.43%	2.55%	14.79%	1.36%	44.66%	40.66%
% RF	37.50%	9.49%	0.00%	0.00%	29.13%	39.00%
% Auto	29.46%	0.00%	31.08%	85.07%	0.00%	0.00%

Quanto ao uso dos acionamentos possíveis (botão físico presente no módulo, pelo aplicativo backup - web, por controle de radiofrequência - rf e por regra automática configurada pelo usuário - auto), número de acionamentos e porcentagem do tempo ligado, observa-se que:

- No caso de áreas comuns, temos as maiores porcentagens de tempo de lâmpadas acesas, com maiores números de acionamentos pela web, nenhum acionamento por regra automática, dobro de acionamentos por rf do que botão físico para a lâmpada da cozinha e números de acionamentos por rf e botão físico em valores próximos para a Sala;

- Para a escada, temos o melhor uso do acionamento automático (indicando que a regra de acender a lâmpada da escada quando a porta de entrada abrir em horário noturno é bem empregada);
- Para a lâmpada da varanda, ainda que com um bom uso automático, temos cerca de metade dos acionamentos ocorrendo por meio do botão físico;
- Quarto da Sabrina possui nenhum uso de acionamento automático, e poucos acionamentos pela web (celular) e controle rf, indicando que o sistema não agrupa muito valor;
- Lâmpada do corredor possui uma distribuição aproximadamente igualitária entre os acionamentos, contrariando o comportamento esperado de que as regras automáticas de acendimento de luz quando da presença detectada seriam suficientes para o controle dessa lâmpada;
- O uso de acionamentos pelo aplicativo (web) somente são consideráveis nas áreas comuns (provavelmente quando os usuários desligam as luzes da casa toda antes de dormir). Pode-se considerar que, apesar de ser um meio adequado de apresentar o funcionamento do sistema, no uso cotidiano os usuários preferem usar controles rf, acionamento manual por botão físico ou deixar uma regra para atuação automática.

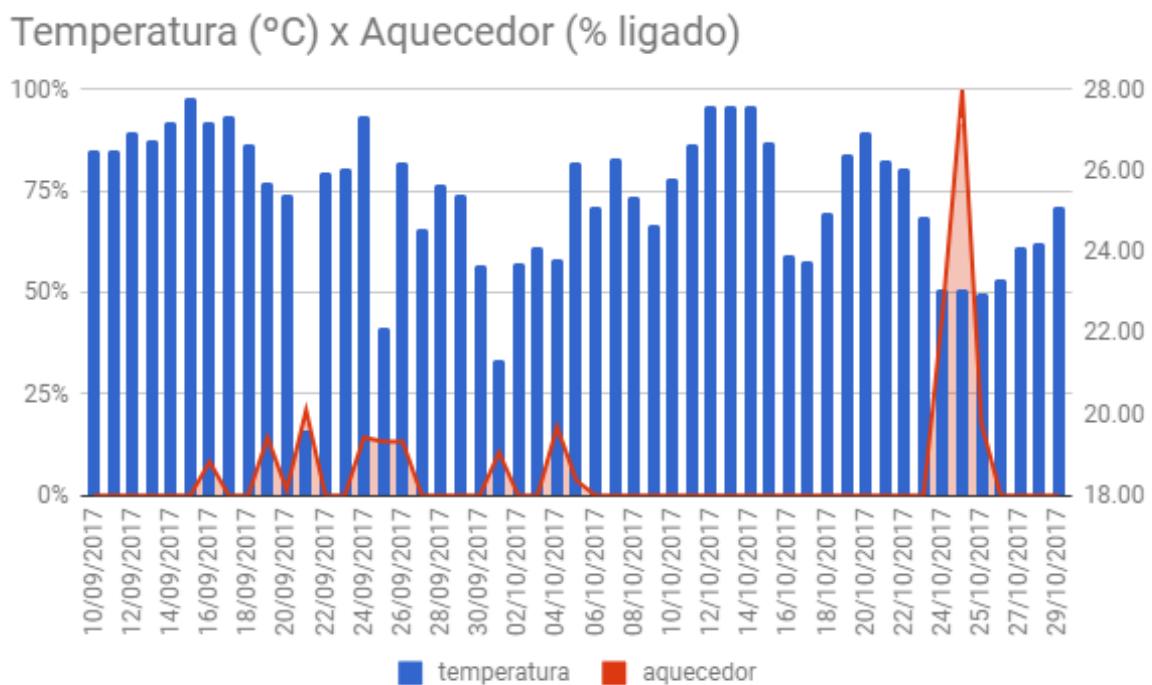
Figura 49: Curva Diária Santo André - Terça



Tomando o dia de terça como exemplo dos gráficos consolidados de Santo André como exemplo, observa-se que:

- Há um pico de presença no corredor às 3:00 a.m. Seria o horário em que os gatos da residência estão mais ativos;
- A curva de presença indica maiores movimentos entre 9hs e 13hs, e entre 17hs e 01hs (horários em que as pessoas entram ou saem da residência);
- Considerando que a segmentação de usuários da entrada está em Fabio, Victor e Admin (Nair e Sabrina):
 - Das 4hs às 7hs, Nair/Sabrina sai da casa;
 - Das 9hs às 13hs, Fabio sai de casa, algumas vezes usando seu usuário, outras vezes manualmente;
 - Às 10hs, Victor sai de casa;
 - Das 17hs às 19hs, Sabrina/Nair voltam para casa;
 - Entre 19hs e 22hs, Victor volta para casa;
 - Entre 21hs e 23hs, Fabio volta para casa;
- Os picos de porta escada indicam os horários de maior entrada e saída da residência.

Figura 50: Temperatura e Aquecedor - Aquário



Considerando a curva diária e histórico do período dos sensores e atuador do aquário, observa-se que:

- Às 3hs da manhã, temos lâmpada acesa e aquecedor aceso. Isso pode ter ocorrido em um dia em particular (25/11/2017, dia atípico, como observado no gráfico do período);
- Das 9hs às 23hs, há uma regra para que a lâmpada do aquário fique acesa;
- A temperatura é menor às 10hs, e das 14hs às 18hs. O comportamento do aquecedor é no sentido de ligar nesses períodos.

Figura 51: Consolidado Diário - Aquário

Consolidado Diário

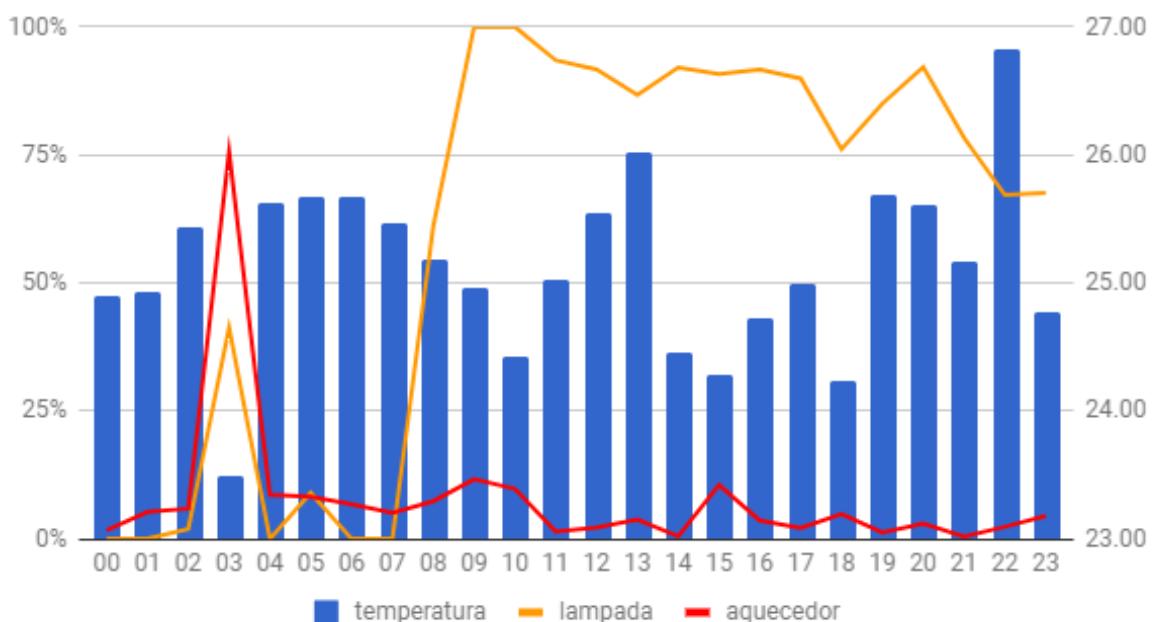
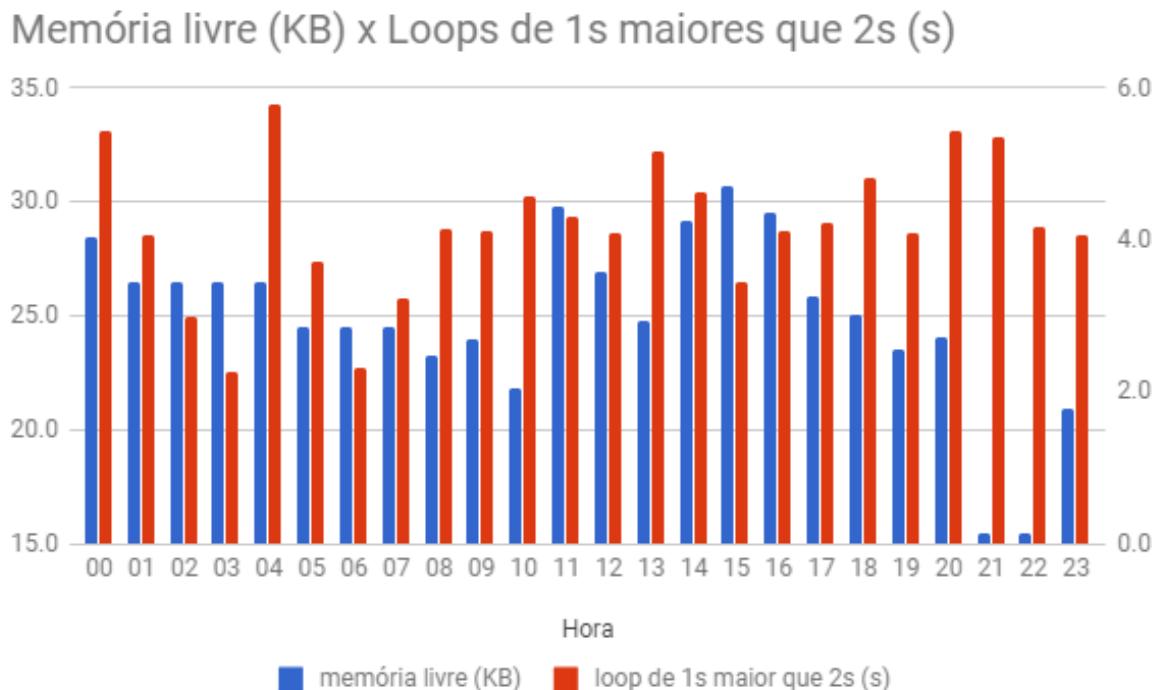


Figura 52: Memória e loops do Corredor

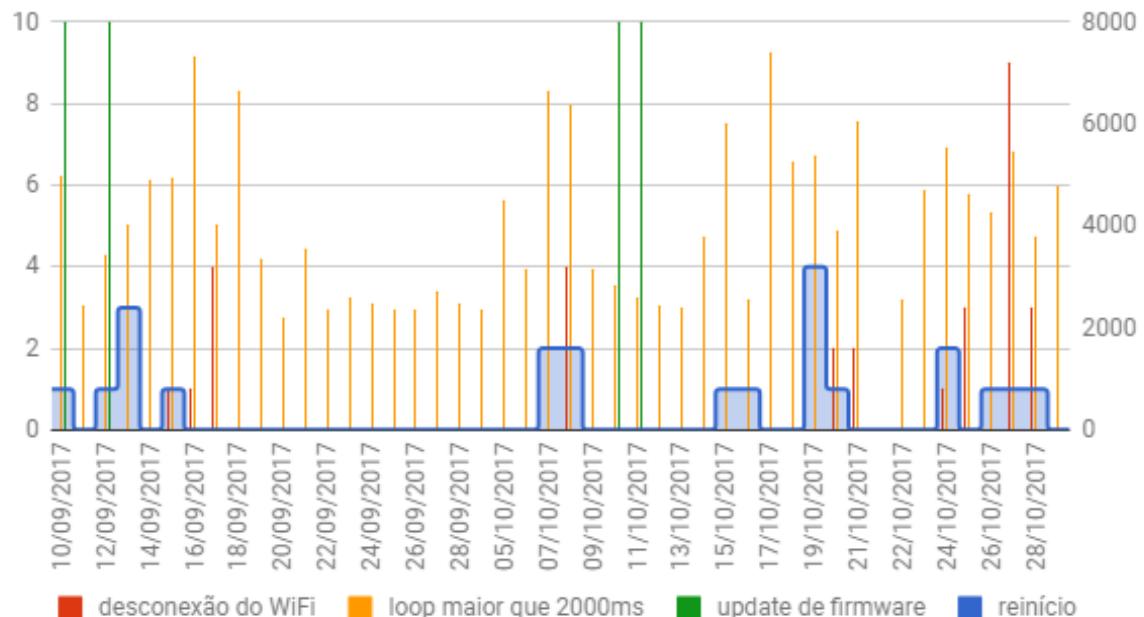


Observando sua curva diária de memória livre e loops de 1s que foram executados em mais de 2s, temos que:

- Os períodos de maior disponibilidade são das 11hs às 16hs, pois temos maior memória livre (parâmetro mais crítico para possíveis reinícios e travamentos);
- Os períodos de menor disponibilidade são das 21hs às 22hs, onde tivemos picos de loops demorados e pouquíssima memória livre.

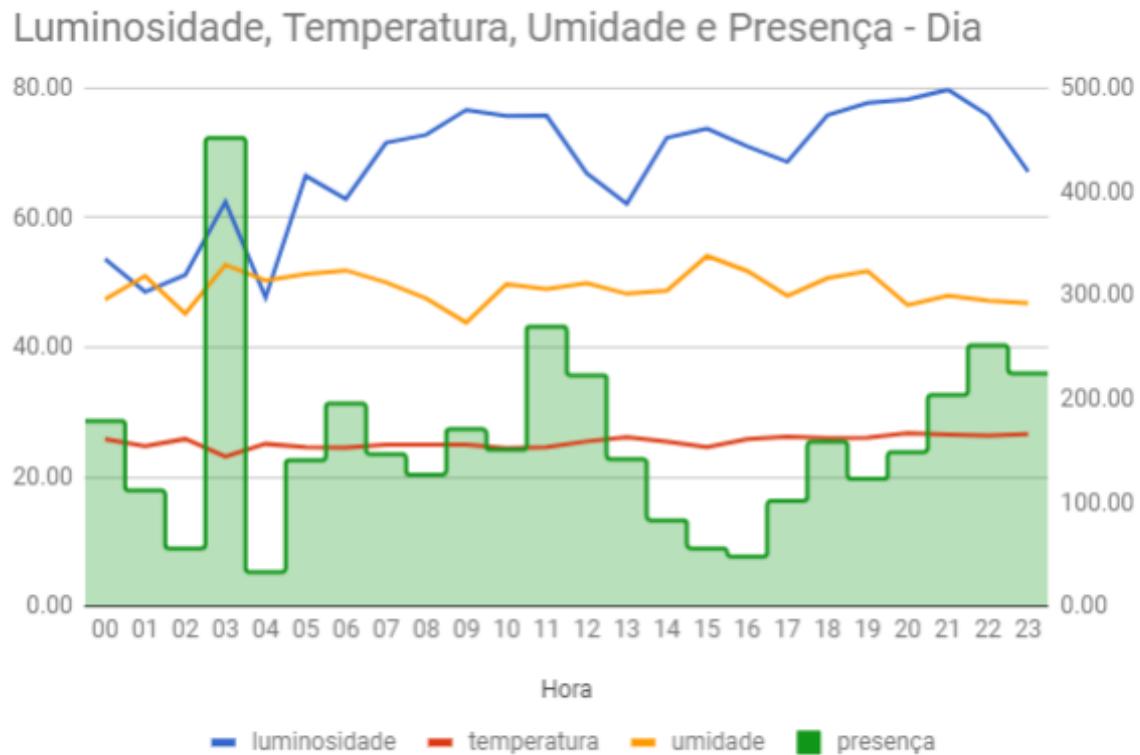
Figura 53: Consolidado no Período - Corredor

Consolidado no Período



Observamos, em verde, no gráfico acima, os updates de firmware (possibilidade de acompanhamento de melhoria de disponibilidade após a implantação de novas versões), a quantidade de reinícios e ocorrências de desconexão do WiFi. Entre 24/10 e 28/10, tivemos grande número de desconexões do WiFi, indicando indisponibilidade da rede nesses dias, que causaram certa quantidade de reinícios no mesmo período.

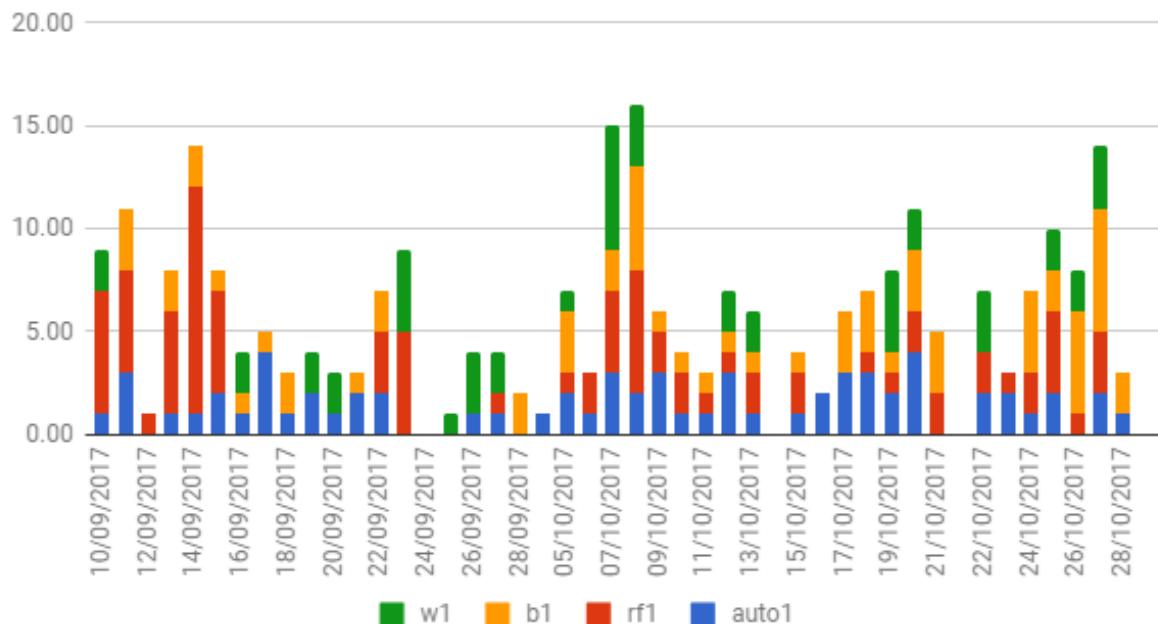
Figura 54: Consolidado Diário dos Sensores - Corredor



No gráfico acima, observamos inatividade às 4hs da manhã e entre as 14hs e 16hs. A temperatura possui pouca variação, assim como a umidade. Já a curva de luminosidade acompanha a de presença (apenas das 14hs às 16hs, considere que temos luz natural, por isso não é um valor próximo àquele apresentado de madrugada, entre 0hs e 4hs).

Figura 55: Uso dos acionamentos no período para a lâmpada do Corredor

Uso dos tipos de acionamento para o Rele 1 - Período



No primeiro gráfico, observamos a evolução do uso dos acionamentos para o rele 1 (lâmpada do corredor) no período. Observam-se dias com maiores e menores quantidades de acionamentos, e a participação das regras automáticas em relação aos outros tipos de acionamento. No segundo, verifica-se que, no dia, os acionamentos automáticos ocorrem entre 17hs e 22hs. Possivelmente, podemos aplicar regras no período das 23hs às 2hs.

Figura 56: Uso dos acionamentos por dia para a lâmpada do Corredor

Uso dos tipos de acionamento para o Rele 1 - Dia

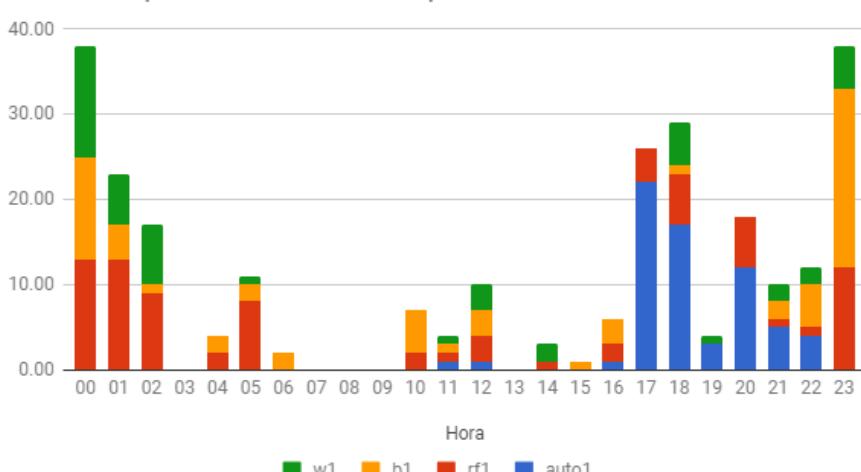
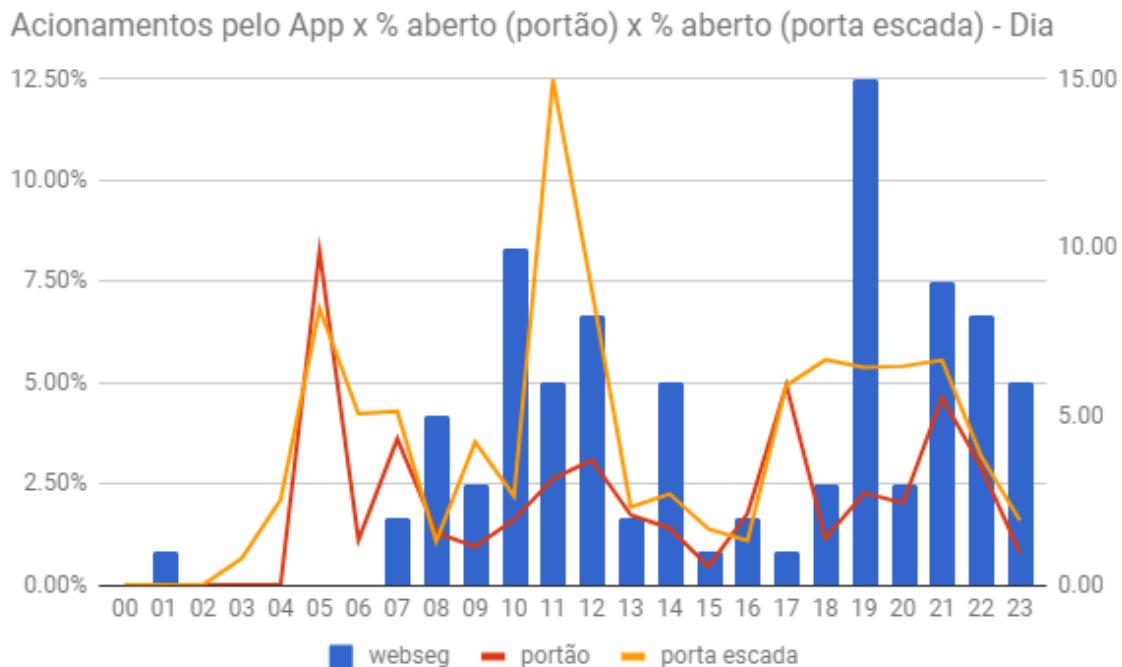


Figura 57: curva Diária do Módulo de Acesso

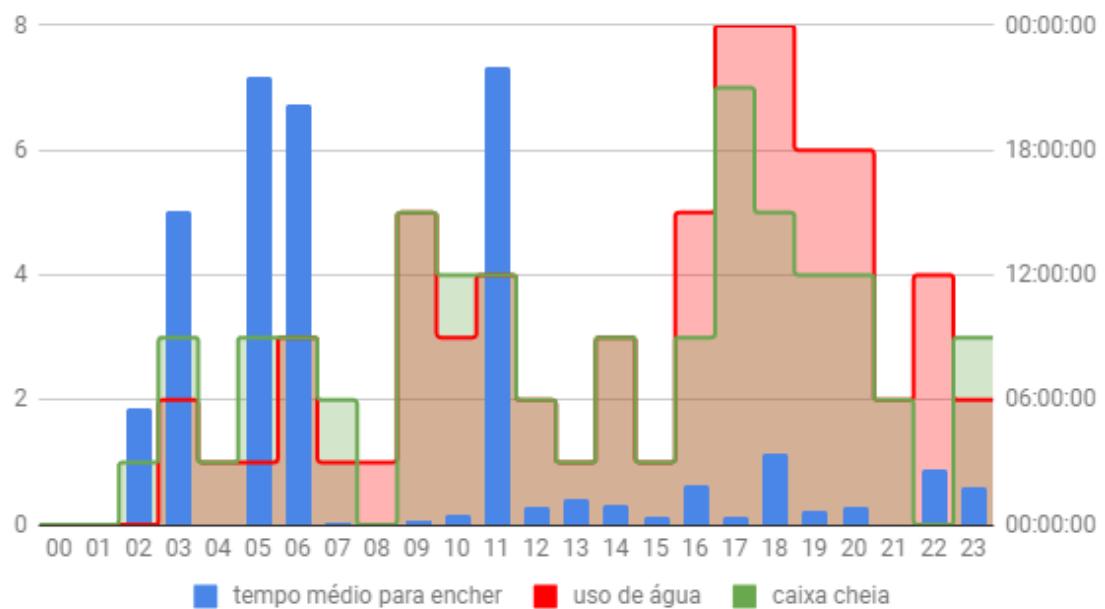


Com a curva diária de comportamento para o módulo de entrada, observa-se que:

- Os horários com maior número de acionamentos pelo celular são 10hs, 12hs, 19hs, 21hs, 22hs e 23hs;
- Usuários que entram ou saem de casa às 17hs não usam o celular para abrir o portão, tampouco aqueles que entram ou saem às 5hs.

Figura 58: Nível da Caixa d'água - Consolidado Diário

Caixa d'água - Consolidado por Dia



Na curva diária do módulo da caixa d'água, nota-se que:

- Há maior uso de água (provavelmente para banho) nos períodos das 9hs às 12hs, e entre 16hs e 23hs;
- Os horários em que temos, provavelmente, menor disponibilidade de água da rua (para preencher a caixa e fazer o nível voltar a ser alto) são das 5hs às 6hs da manhã e às 11hs da manhã.

Observando o gráfico a seguir, observam-se os dias em que houve falta de água (picos de demora para preencher a caixa de água após algum uso perceptível pelo sensor do tipo boia).

Figura 59: Tempo para Encher a Caixa d'água - Período

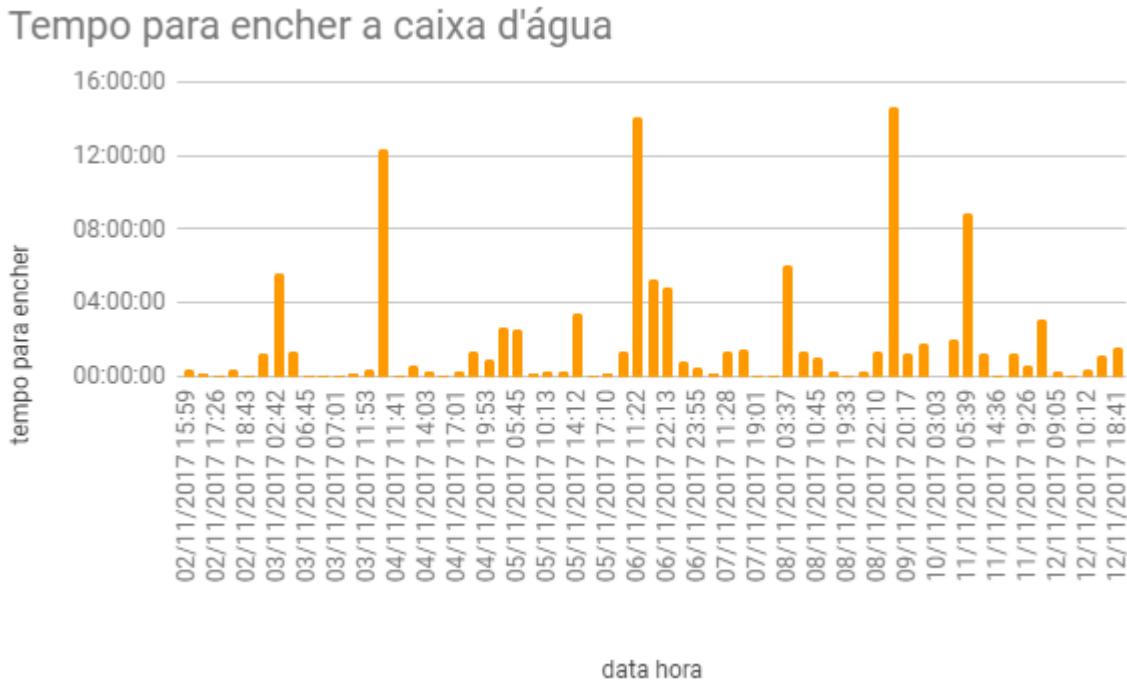
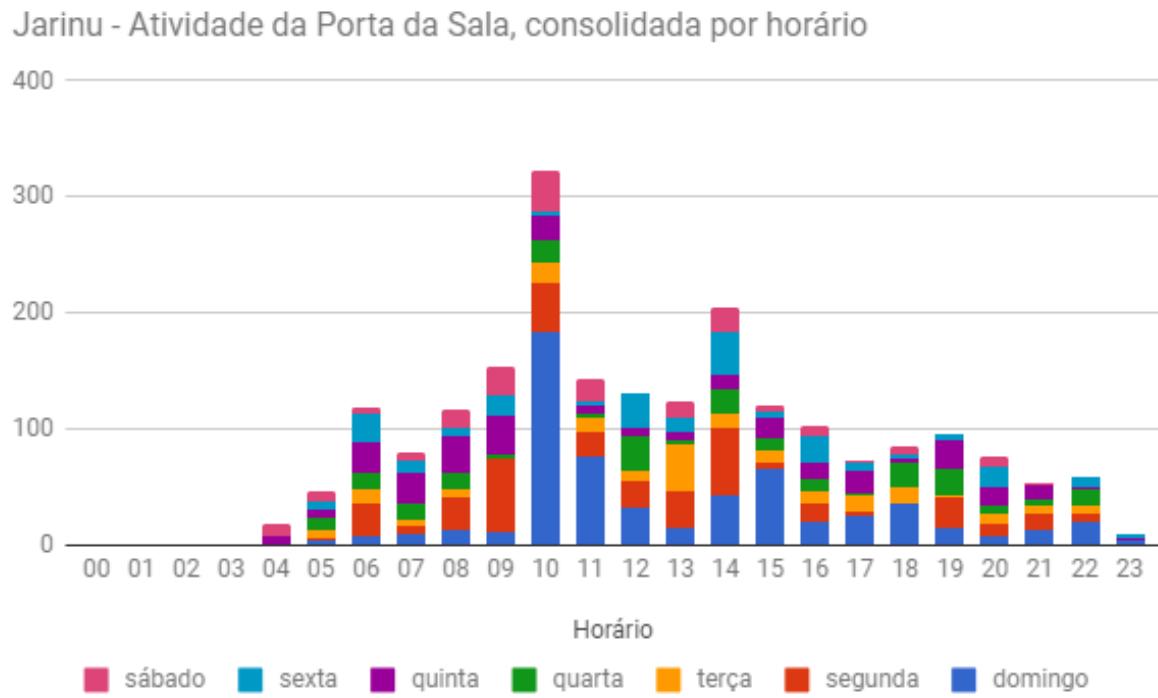


Figura 60: Atividade da Porta da Sala de Jardinu



Em Jardinu, temos um idoso, com boa saúde física, que mora sozinho em sua residência.

Ao analisar as curvas diárias de atividade da porta da sala e presença, temos:

- Atividade de presença entre 4hs e 5hs da manhã; entre 10hs e 12hs, e entre 16hs e 20hs.
- Atividade de presença maior em domingos, segundas e terças;
- Grande atividade da porta da sala às 10hs e 14hs. O pico das 10hs ocorre principalmente aos domingos.

Figura 61: Atividade da Presença da Sala de Jarinu

Jarinu - Atividade de Presença, consolidada por horário

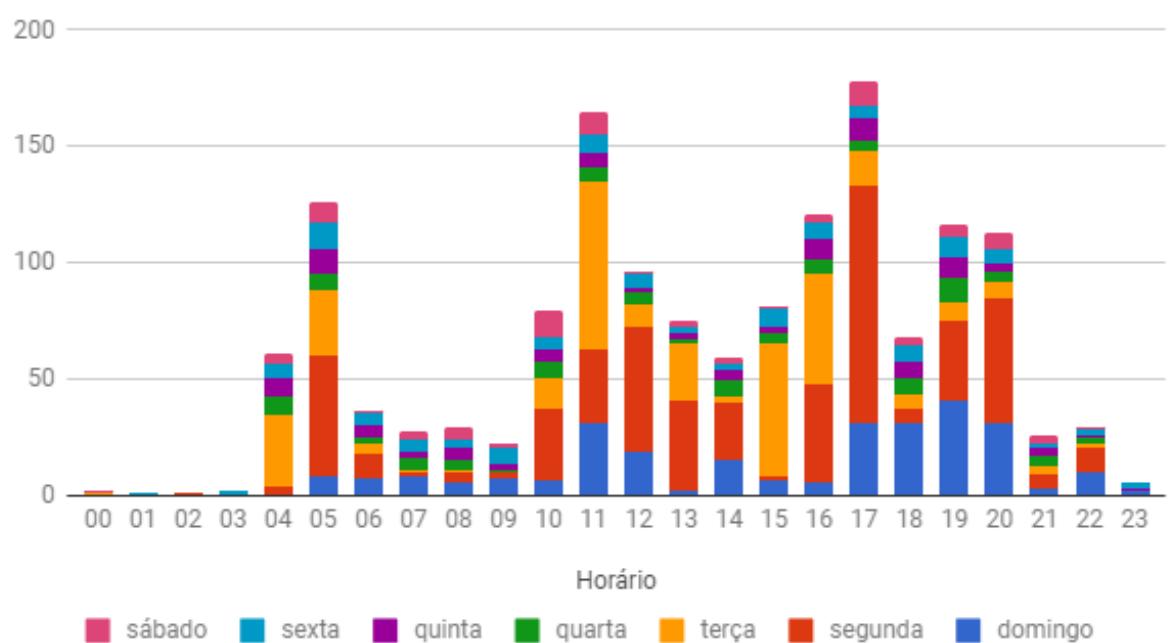


Figura 62: Consolidado no Período - Jarinu



Ao analisar por período, podemos verificar o aumento ou diminuição de atividade, o que pode ser um indicativo de bem estar. Por exemplo, no gráfico acima verifica-se que o dia 04/11 foi um dia atípico, com menores presença e atividade da porta da sala. Em geral, não houve tendência de crescimento ou diminuição do nível de atividade geral no período observado.

8 Conclusões

REFERÊNCIAS

- 9 TO 5 MAC. *Jobs' original vision for the iPhone: No third-party native apps.* 2011. <<https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/>>. Accessed: 2017-11-07.
- ATLASSIAN. *Comparing Workflows.* 2017. <<https://www.atlassian.com/git/tutorials/comparing-workflows>>. Accessed: 2017-04-17.
- BENSON, M. *An end or an error: App downloads are on a dangerous decline in the USA.* 2016. <<https://www.androidauthority.com/end-era-app-downloads-decline-usa-698555/>>. Accessed: 2017-11-10.
- BIBLIOTECA VIRTUAL. *São Paulo: população do estado.* 2017. <<http://www.bibliotecavirtual.sp.gov.br/temas/sao-paulo/sao-paulo-populacao-do-estado.php>>. Accessed: 2017-02-20.
- BROWSER COOKIE LIMITS. *Browser Cookie Limits.* 2016. <<http://browsercookielimits.squawky.net/>>. Accessed: 2017-11-08.
- BUSINESS WIRE. *Internet of Things Spending Forecast to Grow 17.9% 2016 Led by Manufacturing, Transportation, and Utilities Investments, According to New IDC Spending Guide.* 2017. <<http://www.businesswire.com/news/home/20170104005270/en/Internet-Spending-Forecast-Grow-17.9-2016-Led>>. Accessed: 2017-11-17.
- CHAMPEON, S. *Progressive Enhancement and the Future of Web Design.* 2003. <http://hesketh.com/publications/progressive_enhancement_and_the_future_of_web_design.html>. Accessed: 2017-11-11.
- CHEN, A. *New data shows losing 80% the best apps do better.* 2015. <<http://andrewchen.co/new-data-shows-why-losing-80-of-your-mobile-users-is-normal-and-that-the-best-apps-do-much-better>>. Accessed: 2017-11-10.
- CISCO SYSTEMS. *The Zettabyte Era: Trends and Analysis.* 2017. <<https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf>>. Accessed: 2017-11-09.
- ELECTRONIC FRONTIER FOUNDATION MEDIA RELEASE. *Movie Legend Hedy Lamarr to be Given Special Award at EFF's Sixth Annual Pioneer Awards.* 1997. <<https://w2.eff.org/awards/pioneer/1997.php>>. Accessed: 2017-05-15.
- ESPRESSIF SYSTEMS. *ESP8266EX Datasheet.* 2015. <<http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf>>. Accessed: 2017-05-22.
- FACEBOOK OPEN SOURCE. *Building the F8 2016 App: Integrating Data with React Native.* 2016. <<http://makeitopen.com/docs/en/1-3-data.html>>. Accessed: 2017-11-11.

- FACEBOOK OPEN SOURCE. *Reconciliation*. 2017. <<https://reactjs.org/docs/reconciliation.html>>. Accessed: 2017-11-09.
- G1 SÃO PAULO. *Grande SP atinge 83Sabesp*. 2015. <<http://glo.bo/1K5JsQh>>. Accessed: 2017-02-20.
- GITHUB. *The state of the Octoverse 2016*. 2016. <<https://octoverse.github.com/>>. Accessed: 2017-05-17.
- GOOGLE CHROME. *Managing HTML5 Offline Storage*. 2017. <https://developer.chrome.com/apps/offline_storage>. Accessed: 2017-11-08.
- GOOGLE DEVELOPERS. *AliExpress*. 2016. <<https://developers.google.com/web/showcase/2016/aliexpress>>. Accessed: 2017-11-08.
- GOOGLE DEVELOPERS. *eXtra Electronics*. 2016. <<https://developers.google.com/web/showcase/2016/extra>>. Accessed: 2017-11-08.
- GOOGLE DEVELOPERS. *Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage*. 2016. <<https://developers.google.com/web/showcase/2017/twitter>>. Accessed: 2017-11-08.
- GOOGLE DEVELOPERS. *Progressive Web App Checklist*. 2017. <<https://developers.google.com/web/progressive-web-apps/checklist>>. Accessed: 2017-11-07.
- GOOGLE DEVELOPERS. *Progressive Web Apps*. 2017. <<https://developers.google.com/web/progressive-web-apps/>>. Accessed: 2017-11-07.
- I-SCOOP. *IoT in smart home automation: the fast growing list of use cases*. 2017. <<https://www.i-scoop.eu/smart-home-home-automation/iot-home-automation-applications/>>. Accessed: 2017-11-17.
- INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. *Censo Demográfico de 2010. Fundação Instituto Brasileiro de Geografia e Estatística, dados referentes ao município de São Paulo*. 2010. <<http://cod.ibge.gov.br/6QV>>. Accessed: 2017-02-20.
- INTERNET ENGINEERING TASK FORCE. *The WebSocket Protocol*. 2011. <<https://tools.ietf.org/html/rfc6455>>. Accessed: 2017-11-19.
- INTERNET ENGINEERING TASK FORCE. *JSON Web Token (JWT)*. 2015. <<https://tools.ietf.org/html/rfc7519>>. Accessed: 2017-11-08.
- ISO/IEC. Iso/iec is 25010: Software engineering: Software product quality requirements and evaluation (square) — quality model. In: . [S.l.], 2011.
- ISO/IEC-IEEE. *ISO/IEC 12207: Systems and software engineering — Software life cycle processes, 2^a ed.* 2. ed. [S.l.]: ISO/IEC-IEEE, 2008.
- JAMES, G. e. a. *An Introduction to Statistical Learning with Applications in R*. [S.l.]: [S.L.]: Springer, 2013.
- JWT. *Introduction to JSON Web Tokens*. 2016. <<https://jwt.io/introduction/>>. Accessed: 2017-11-08.

- KENNEMER, Q. *Google Home gets a \$5 million ad spot in the Superbowl.* 2017. <<http://phandroid.com/2017/02/01/google-home-gets-a-5-million-ad-spot-in-the-super-bowl>>. Accessed: 2017-05-15.
- LEWIS, J.; FOWLER, M. *Microservices: a definition of this new architectural term.* 2014. <<https://martinfowler.com/articles/microservices.html>>. Accessed: 2017-05-22.
- LUBBERS, P.; GRECO, F. *HTML5 WebSocket: A Quantum Leap in Scalability for the Web.* 2016. <<https://websocket.org/quantum.html>>. Accessed: 2017-11-19.
- MCGRAW, G. *Software Security.* 2004. <<http://ieeexplore.ieee.org/abstract/document/1281254/>>. Accessed: 2017-10-21.
- MCKINSEY & COMPANY. *There's No Place Like A Connected Home.* 2016. <http://www.mckinsey.com/spContent/connected_homes/index.html>. Accessed: 2017-04-24.
- MOZILLA DEVELOPER NETWORK. *JavaScript.* 2016. <<https://developer.mozilla.org/bm/docs/Web/JavaScript>>. Accessed: 2017-11-11.
- MULLINS, C. S. *What is an In-Memory Database System?* 2017. <<http://www.dbta.com/Columns/DBA-Corner/What-is-an-In-Memory-Database-System-119241.aspx>>. Accessed: 2017-11-19.
- MURAMATSU, F. T.; RODRIGUES, H.; GALLEGOS, R. B. Projeto homesky. trabalho de conclusão de curso (graduação em engenharia de computação) - escola politecnica, universidade de são paulo, são paulo. 2016.
- MYSLAWSKI, R. *iPhone App Store breezes past 500 million downloads.* 2009. <https://www.theregister.co.uk/2009/01/16/half_billion_iphone_apps/>. Accessed: 2017-11-10.
- NETWORK WORKING GROUP. *The Syslog Protocol.* 2009. <<https://tools.ietf.org/html/rfc5424>>. Accessed: 2017-11-20.
- NOSQL DATABASES. *NOSQL Databases.* 2009. <<http://nosql-database.org/>>. Accessed: 2017-11-17.
- PEREIRA, T. *Quando utilizar RDBMS ou NoSQL?* 2016. <<http://datascienceacademy.com.br/blog/quando-utilizar-rdbms-ou-nosql/>>. Accessed: 2017-11-19.
- PROJECT MANAGEMENT INSTITUTE. *PMI. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos. Guia PMBOK®.* 2004, 3^a ed. 3. ed. [S.l.]: PMI, 2004.
- RAIMA. *In-Memory Advantages From RDM.* 2013. <<https://raima.com/in-memory-database/>>. Accessed: 2017-11-19.
- RASPBERRY PI FOUNDATION. *Model 3.* 2017. <<https://www.raspberrypi.org/products/raspberry-pi-3-model-b>>. Accessed: 2017-06-12.
- REDUX. *Introduction.* 2017. <<https://redux.js.org/docs/introduction/>>. Accessed: 2017-11-09.

- RICKER, T. *Jobs: App Store launching with 500 iPhone applications, 25free.* 2008. <<https://www.engadget.com/2008/07/10/jobs-app-store-launching-with-500-iphone-applications-25-free/>>. Accessed: 2017-11-10.
- ROTEM-GAL-OZ, A. *Services, Microservices, Nanoservices – oh my!* 2014. <<http://aronon.me/2014/03/services-microservices-nanoservices/>>. Accessed: 2017-05-23.
- RUDOLPH, P. *Hybrid Mobile Apps: Providing A Native Experience With Web Technologies.* 2014. <<https://www.smashingmagazine.com/2014/10/providing-a-native-experience-with-web-technologies/>>. Accessed: 2017-11-08.
- RUSELL, A. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul.* 2015. <<https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>>. Accessed: 2017-11-07.
- SAVIT, J. *Availability Best Practices - Avoiding Single Points of Failure.* 2013. <<https://blogs.oracle.com/jsavit/availability-best-practices-avoiding-single-points-of-failure>>. Accessed: 2017-10-19.
- SHEARER, S. M. *Beautiful: The Life of Hedy Lamarr.* [S.l.]: Thomas Dunne Books, 2010. ISBN 978-0-312-55098-1.
- SUBLIMELINTER. *About SublimeLinter.* 2016. <<http://www.sublimelinter.com/en/latest/about.html>>. Accessed: 2017-11-09.
- THOMSEN, A. *Como programar o NodeMCU com IDE Arduino.* 2016. <<https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/>>. Accessed: 2017-05-22.
- VISWANATHAN, P. *Cloud Computing and Is it Really All That Beneficial?* 2017. <<https://www.lifewire.com/cloud-computing-explained-2373125>>. Accessed: 2017-11-18.

ANEXO A – CÓDIGOS DAS APLICAÇÕES DESENVOLVIDAS

Todos os códigos das aplicações desenvolvidas neste projeto estão disponíveis em:
[⟨https://github.com/hedwig-project/⟩.](https://github.com/hedwig-project/)

**ANEXO B – LISTA DE MATERIAIS
PARA MONTAGEM DOS
MÓDULOS**

Tabela 4: Lista de Materiais

Item	Quantidade	Preço	Unid.	Total
Wemos D1 mini	1	19.99	19.99	
RF 433	1	7.59	7.59	
DHT11	1	5.99	5.99	
Módulo I2C	1	7.99	7.99	
Display LCD 16x2	1	11.99	11.99	
Fonte 5V 3W	1	18.9	18.90	
Sensor de Presença embutido	1	6.99	6.99	
Chave Push Button R13-507 Sem Trava Preta	2	1.51	3.02	
Chave Tactil 6x6x5mm 4 Terminais	1	0.12	0.12	
Rolo de Solda Best Azul 189 MSX10 60x40 1/2 Kilo Fio 1mm	0.04	57.99	2.42	
Placa de Circuito Impresso Padrão 10x20 cm Tipo Ilha	0.5	13.2	6.60	
Cabo de Força	1	2.49	2.49	
Caixa Patola PB-114/2 36x97x147	1	18.02	18.02	
Relé T73 5V 1 Pólo 2 Posições 5 Terminais 125V 10A	2	2.32	4.64	
Circuito Integrado LM555 (NE555/NE555P)	1	0.67	0.67	
Buzzer 12mm Com Oscilador Interno 5V	1	1.28	1.28	
Borne KF-3000 2 Terminais	2	0.52	1.04	
Borne KF-3000 3 Terminais	2	0.95	1.90	
Capacitor de Tântalo 10uF	1	0.67	0.67	
LDR	1	0.41	0.41	
Capacitor poliéster 100nF	2	0.28	0.56	
Chave Gangorra KCD1-102 Preta 3 Terminais	1	0.8	0.80	
Resistor 1k	1	0.05	0.05	
Resistor 100k	1	0.05	0.05	
Resistor 3M3	2	0.05	0.10	
Capacitor eletrolítico 10uF	1	0.09	0.09	
Resistor 470m	1	0.05	0.05	
Pino 180º	4	0.41	1.64	
Diodo 1N4007	2	0.06	0.12	
Diodo 1N4148	7	0.04	0.28	
Transistor BC548C	4	0.19	0.76	
Resistor 3k3	6	0.05	0.30	
Resistor 10k	4	0.05	0.20	
Controle Remoto	1	14.5	14.50	
Sensor de Abertura sem fio	1	17.9	17.90	
Total				160.12

ANEXO C – DADOS COLETADOS DE 10/09/2017 A 13/11/2017

Figura 63: Resumo do Aquário

Módulo	Aquário		
Data Início	10/09/2017		
data Fim	29/10/2017		
Disponibilidade		Sensores	
Total de reinícios*	34	Temperatura	
# Updates de Firmware	18	# Medidas	111
# Erros ping local	87	Média (°C)	25.14
# Tentativas de conexões WiFi	42	Atuadores	
# Desconexões WiFi	15	Lâmpada	
# Medidas Free Heap (memória livre)	363	# Acionamentos	200
Memória livre média (bytes)	30,278	% Ligado	69.10%
# Tentativas Conexões NTP	60	Aquecedor	
% de falha NTP	0.00%	# Acionamentos	165
# Tentativas Conexões myIP	227	% Ligado	6.86%
% de falha myIP	60.35%	Bomba	
Tempo Medio para Conexão myIP (segundos)	2.76	# Acionamentos	3
# Tentativas Conexões Blynk	74	% Ligado	96.01%
% de falha Blynk	50.00%		
Tempo Medio para Conexão Blynk (segundos)	0.56	*Exclui updates de firmware	

Figura 64: Disponibilidade Aquário - Período

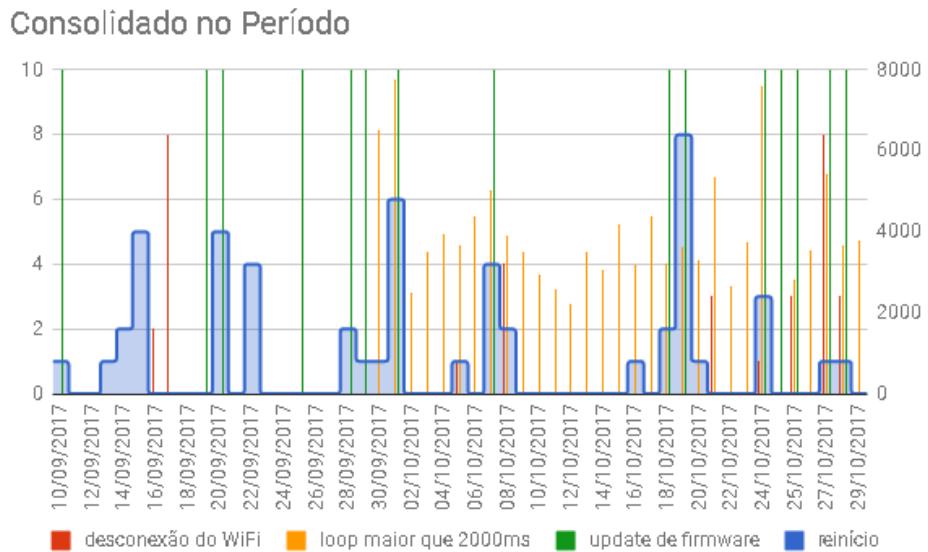


Figura 65: Disponibilidade Aquário - Dia

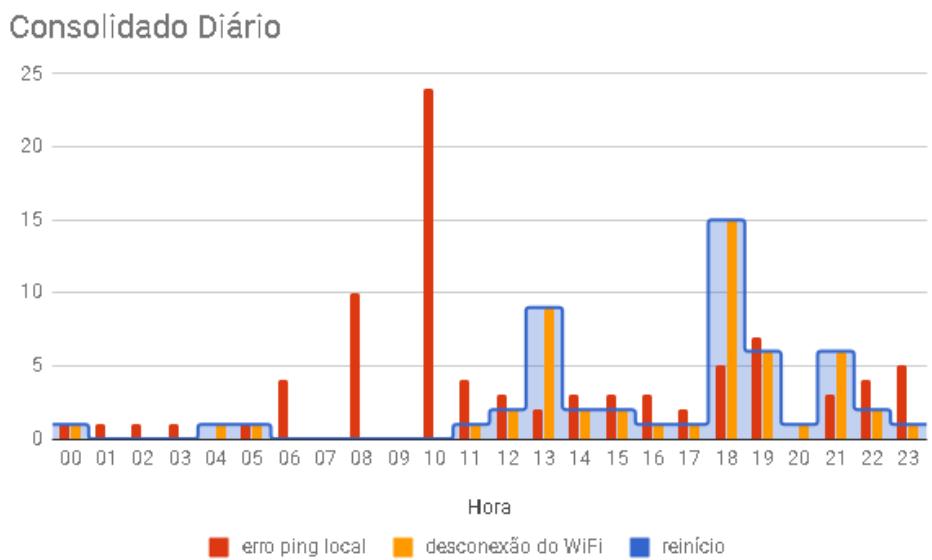


Figura 66: Memória Livre do Aquário

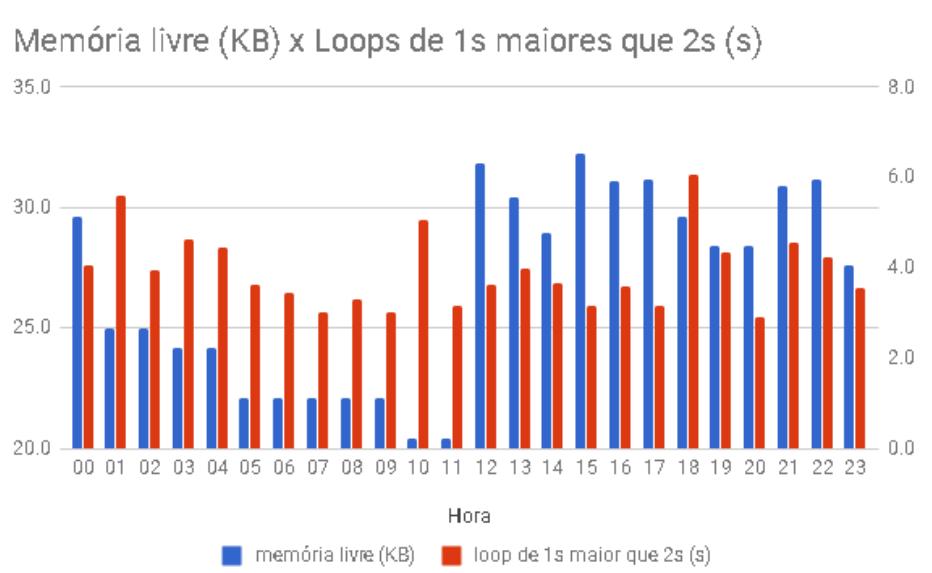


Figura 67: Resumo do Corredor

Módulo	Corredor	Sensores	
Data Início	10/09/2017	Temperatura	
Data Fim	29/10/2017	# Medidas	234
Disponibilidade			
Total de reinícios*	18	Umidade	
# Updates de Firmware	4	# Medidas	237
# Erros ping local	32	Média (%)	43.53
# Tentativas de conexões WiFi	25	Luminosidade	
# Descconexões WiFi	26	# Medidas	729
# Medidas Free Heap (memória livre)	75	Média	637.56
Memória livre média (bytes)	27,218	Presença	
# Tentativas Conexões NTP	19	# Medidas	3,801
% de falha NTP	0.00%	Atuadores	
# Tentativas Conexões myIP	151	Rele 1	
% de falha myIP	78.81%	Nome	Lâmpada Corredor
Tempo Médio para Conexão myIP (segundos)	3.40	% Ligado	Nome
# Tentativas Conexões Blynk	43	# Acionamentos	Quarto Sabrina
% de falha Blynk	60.47%	% Botão Físico	35.08%
Tempo Médio para Conexão Blynk (segundos)	0.82	% Web	% Ligado
*Exclui updates de firmware		# Acionamentos	22.91%
		% Botão Físico	202
		% Web	87.96%
		% Ligado	11.61%
		% Botão Físico	21.43%
		% Web	9.49%
		% RF	29.46%
		% Auto	0.00%

Figura 68: Sensores do Corredor - Período

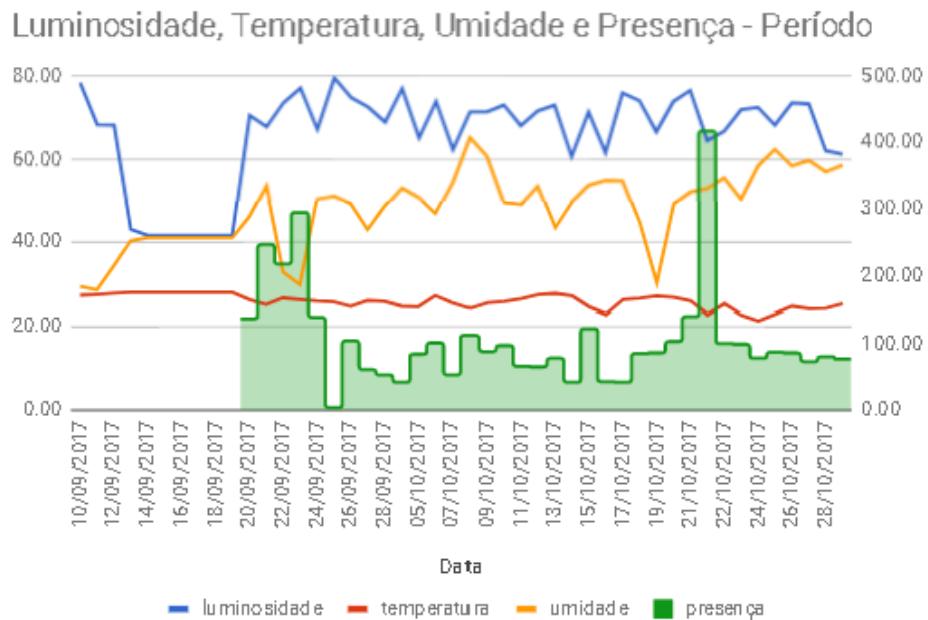


Figura 69: Uso Rele 2 Corredor - Dia

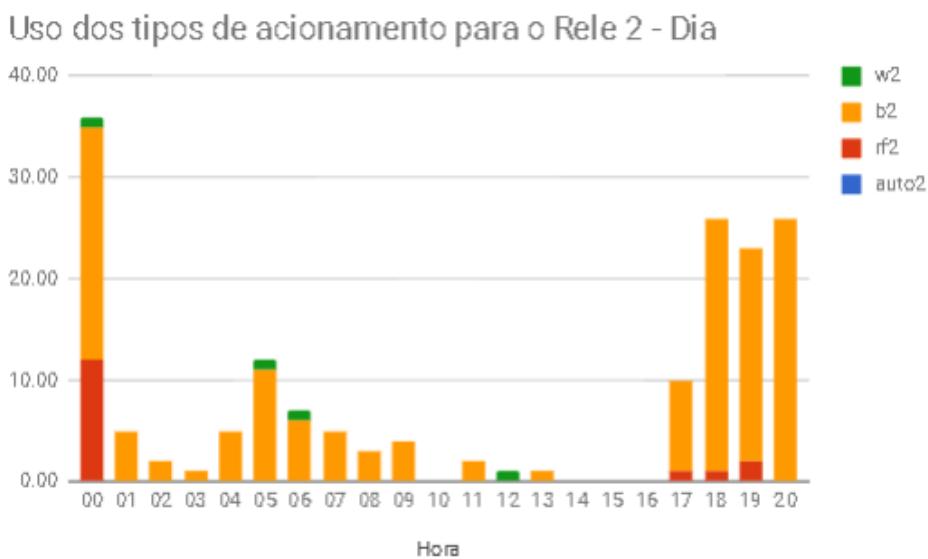


Figura 70: Uso Rele 2 Corredor - Período

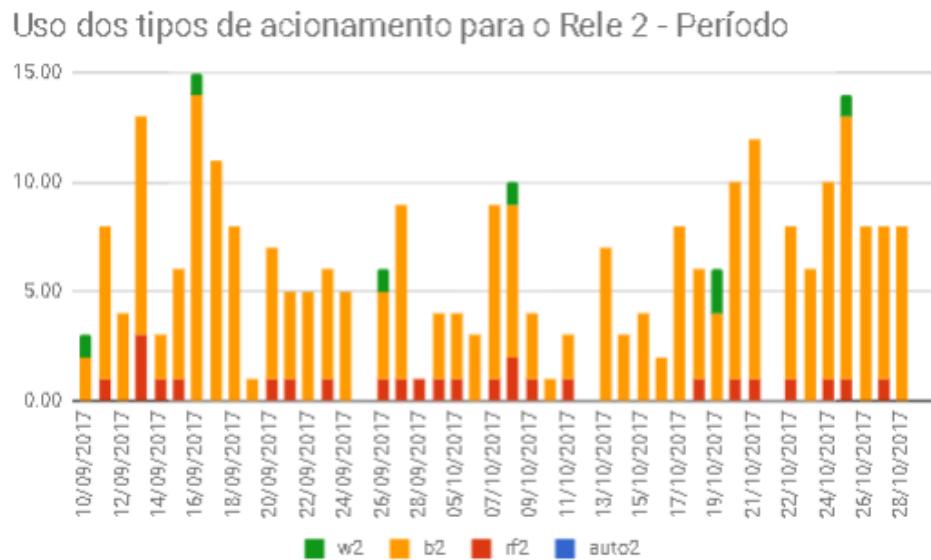


Figura 71: Consolidado Diário Corredor

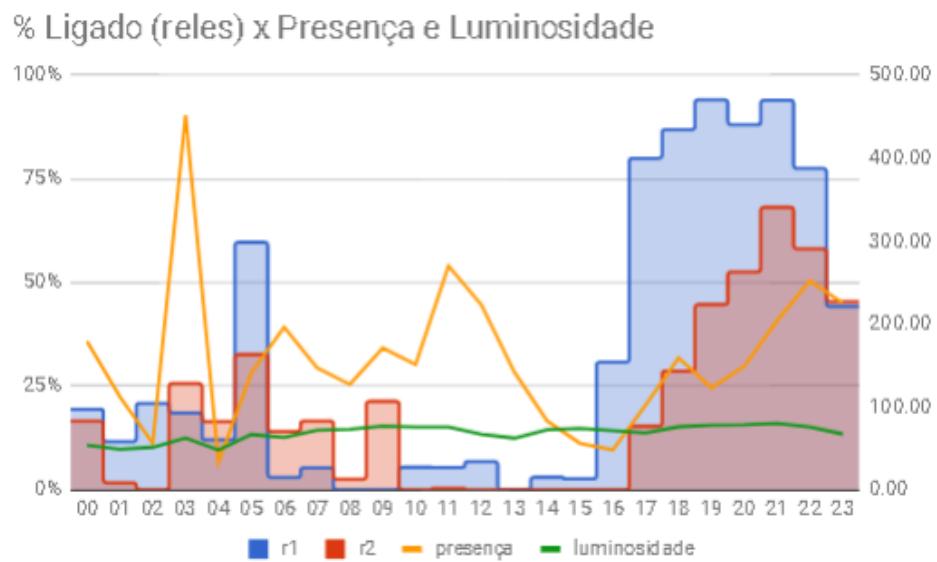


Figura 72: Resumo da Lavanderia

Módulo	Lavanderia	Sensores	
Data Início	12/09/2017	Temperatura	
Data Fim	29/10/2017	# Medidas	589
Disponibilidade		Média (°C)	25.73
Total de reinícios*	35	Umidade	
# Updates de Firmware	3	# Medidas	410
# Erros ping local	23	Média (%)	48.27
# Tentativas de conexões WiFi	36	Luminosidade	
# Descconexões WiFi	24	# Medidas	2,154
# Medidas Free Heap (memória livre)	76	Média	569.05
Memória livre média (bytes)	26,990	Presença	
# Tentativas Conexões NTP	32	# Medidas	10,177
% de falha NTP	3.03%	Atuadores	
# Tentativas Conexões myIP	182	Rele 1	
% de falha myIP	84.07%	Nome	Lâmpada Varanda
Tempo Médio para Conexão myIP (segundos)	2.91	% Ligado	41.62%
# Tentativas Conexões Blynk	59	# Acionamentos	516
% de falha Blynk	55.93%	% Botão Físico	54.14%
Tempo Médio para Conexão Blynk (segundos)	0.80	% Web	14.79%
*Exclui updates de firmware		% RF	0.00%
		% Auto	31.08%
			85.07%

Figura 73: Disponibilidade da Lavanderia - Dia

Consolidado Diário

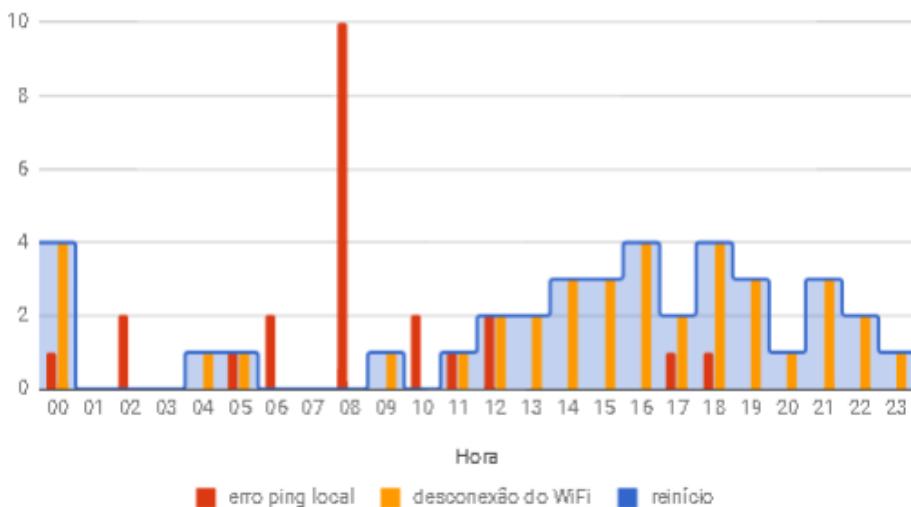


Figura 74: Disponibilidade da Lavanderia - Período

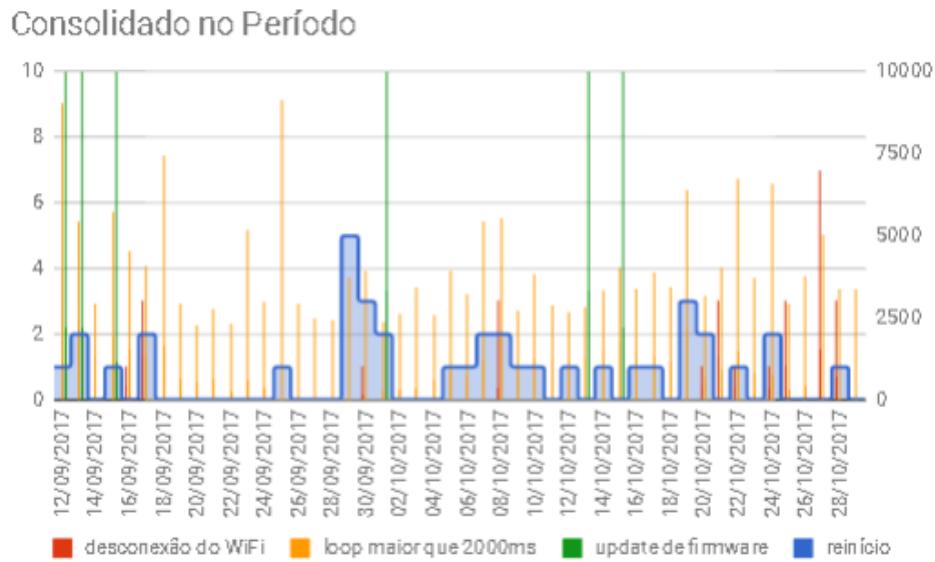


Figura 75: Memória Livre Lavanderia

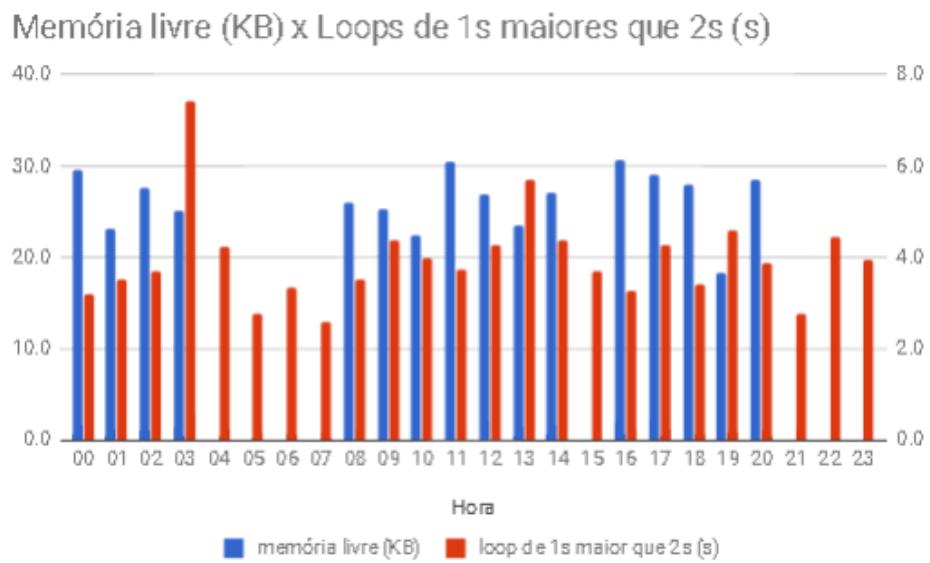


Figura 76: Sensores da Lavanderia - Período

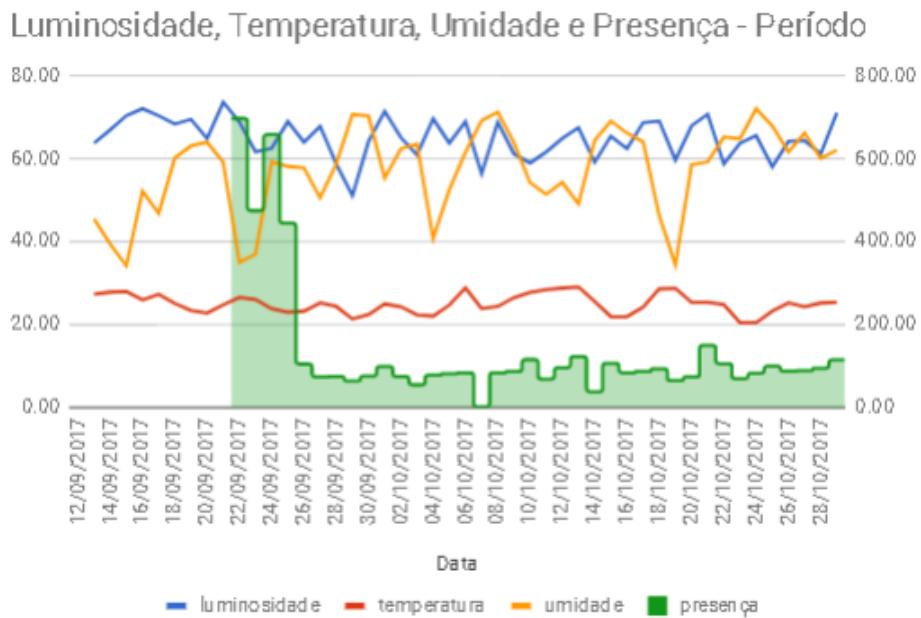


Figura 77: Sensores da Lavanderia - Dia

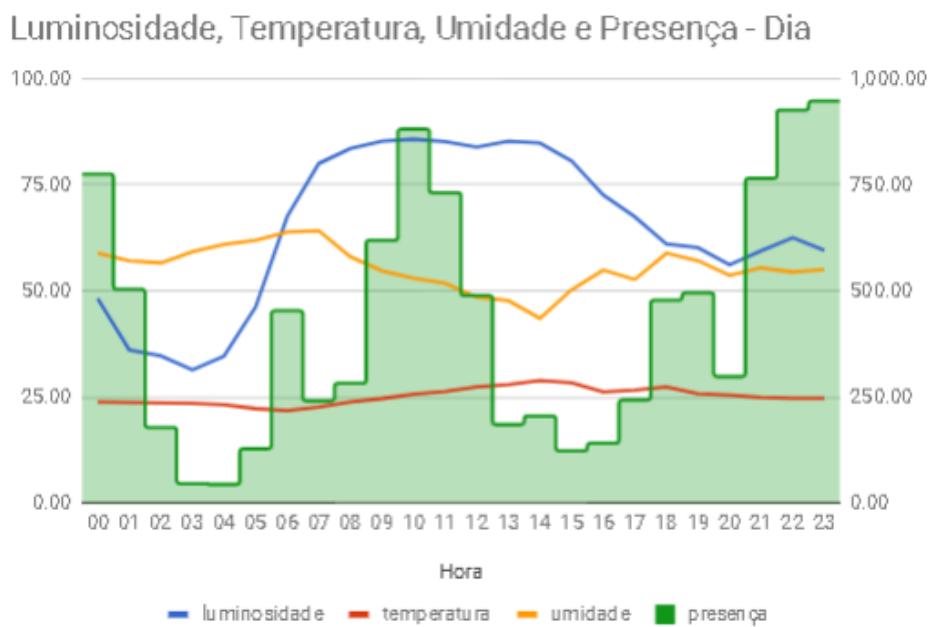


Figura 78: Uso Rele 1 Lavanderia - Dia



Figura 79: Uso Rele 1 Lavanderia - Período

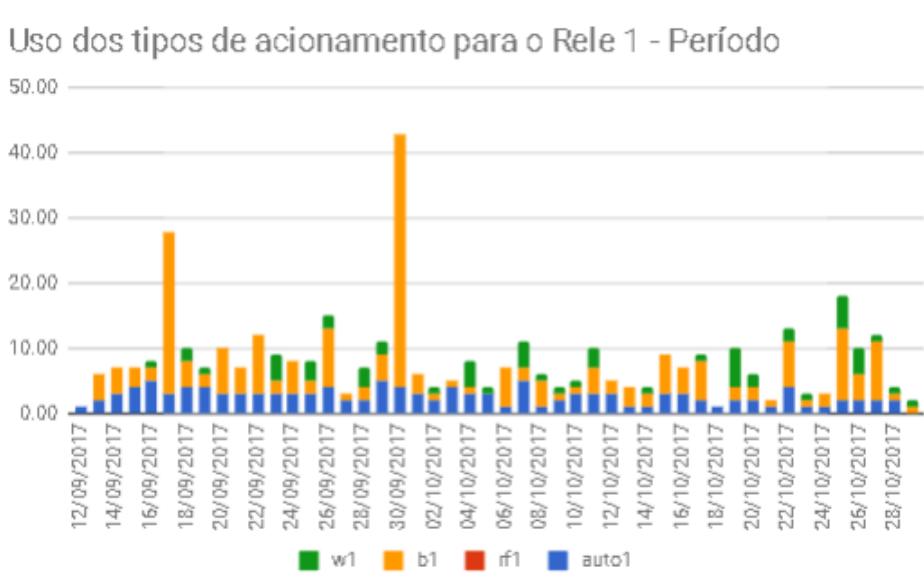


Figura 80: Uso Rele 2 Lavanderia - Dia



Figura 81: Uso Rele 2 Lavanderia - Período

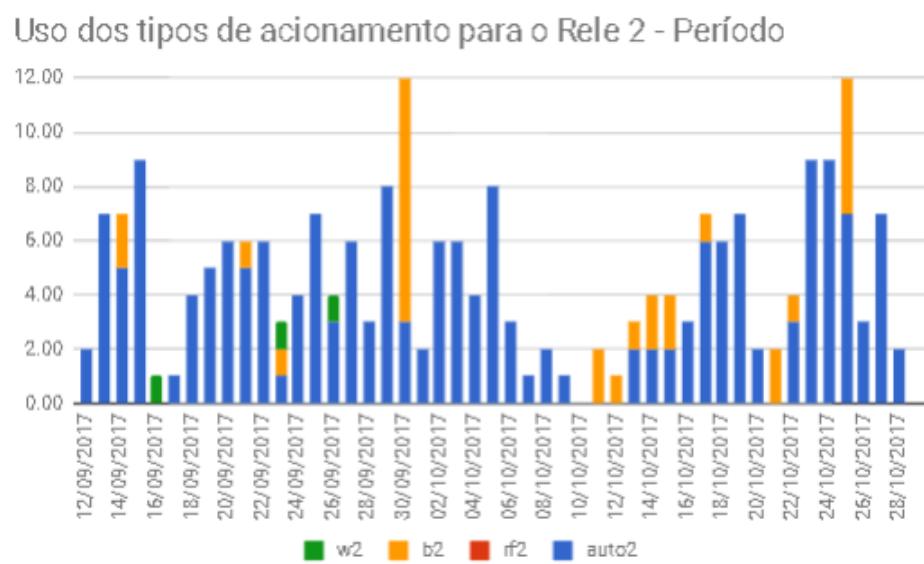


Figura 82: Consolidado Diário da Lavanderia

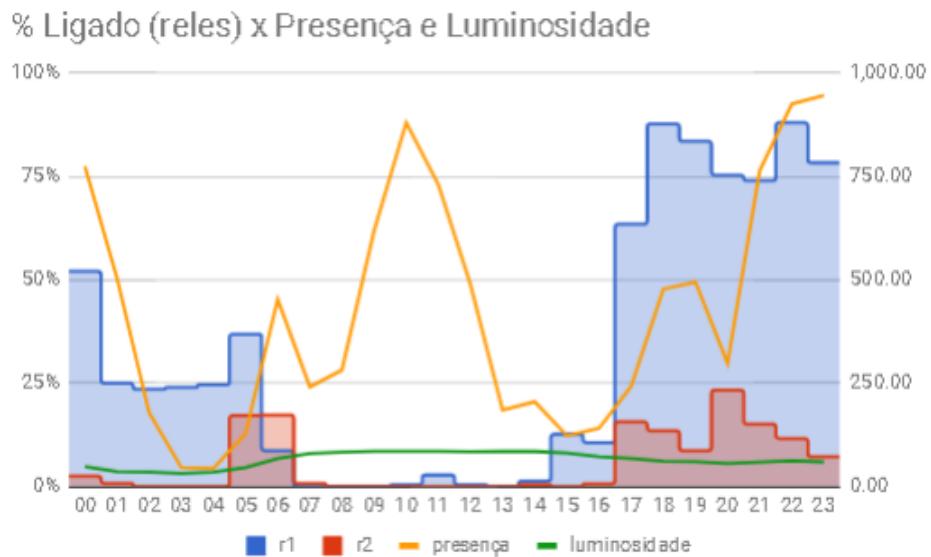


Figura 83: Resumo da Sala/Cozinha

Módulo	SalaCozinha	Sensores	
Data Início	20/09/2017	Temperatura	
data Fim	4/11/2017	# Medidas	304
Disponibilidade		Média (°C)	24.14
Total de reinícios*	20	Umidade	
# Updates de Firmware	0	# Medidas	492
# Tentativas de conexões WiFi	12	Média (%)	56.49
# Desc欠onexões WiFi	27	Luminosidade	
# Medidas Free Heap (memória livre)	60	# Medidas	517
Memória livre média (bytes)	26,728	Média	537.67
Tempo Médio para Conexão myIP (segundos)	2.78	Presença	
# Tentativas Conexões Blynk	38	# Medidas	7.030
% de falha Blynk	13.16%	Sensor de Abertura (Porta de Entrada da Casa)	
Tempo Médio para Conexão Blynk (segundos)	4.09	# Medidas	848
Atuadores		% Aberto	43.61%
Rele 1		Rele 2	
Nome	Sala	Nome	Cozinha
% Ligado	80.06%	% Ligado	60.99%
# Açãoamentos	190	# Açãoamentos	233
% Botão Físico	26.21%	% Botão Físico	20.33%
% Web	44.66%	% Web	40.66%
% RF	29.13%	% RF	39.00%
% Auto	0.00%	% Auto	0.00%

*Exclui updates de firmware

Figura 84: Disponibilidade da Sala/Cozinha - Período

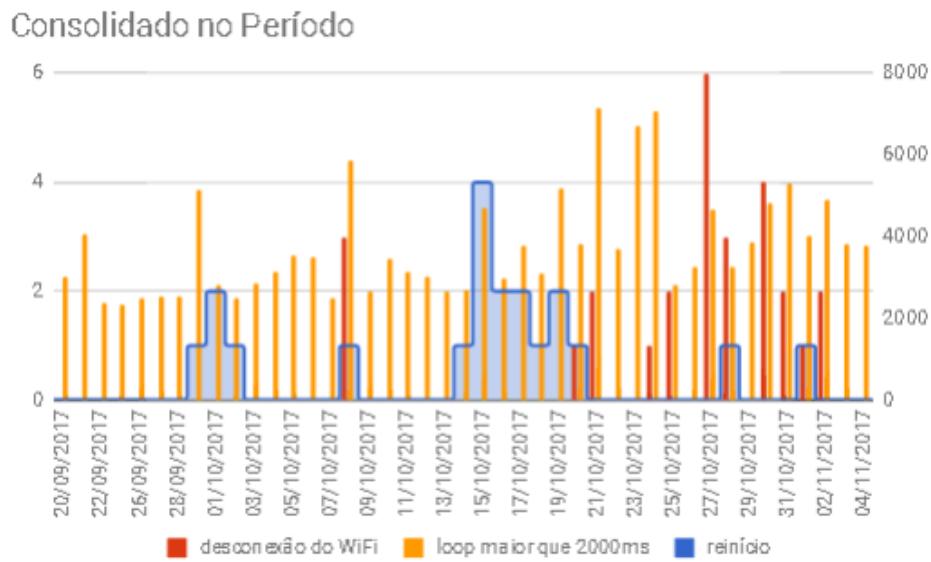


Figura 85: Memória Livre da Sala/Cozinha

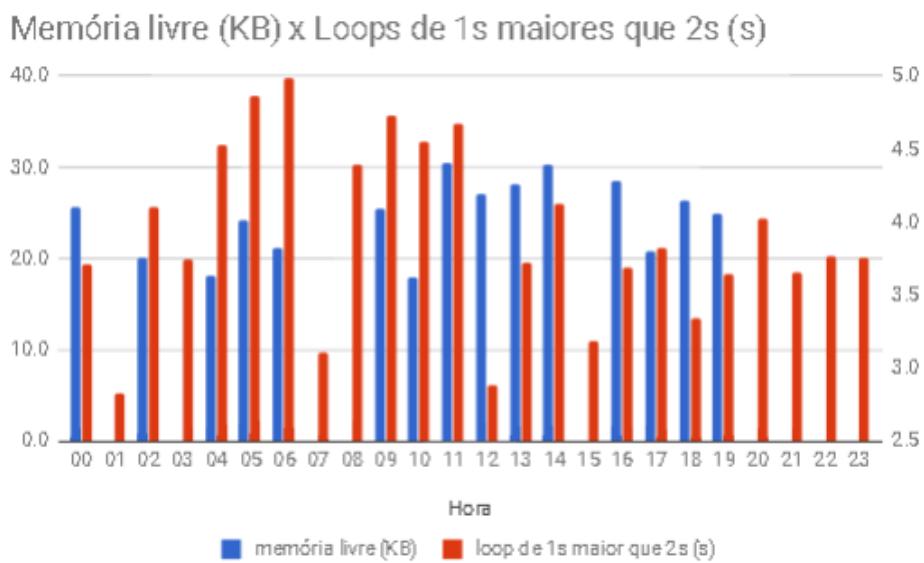


Figura 86: Sensores da Sala/Cozinha - Dia

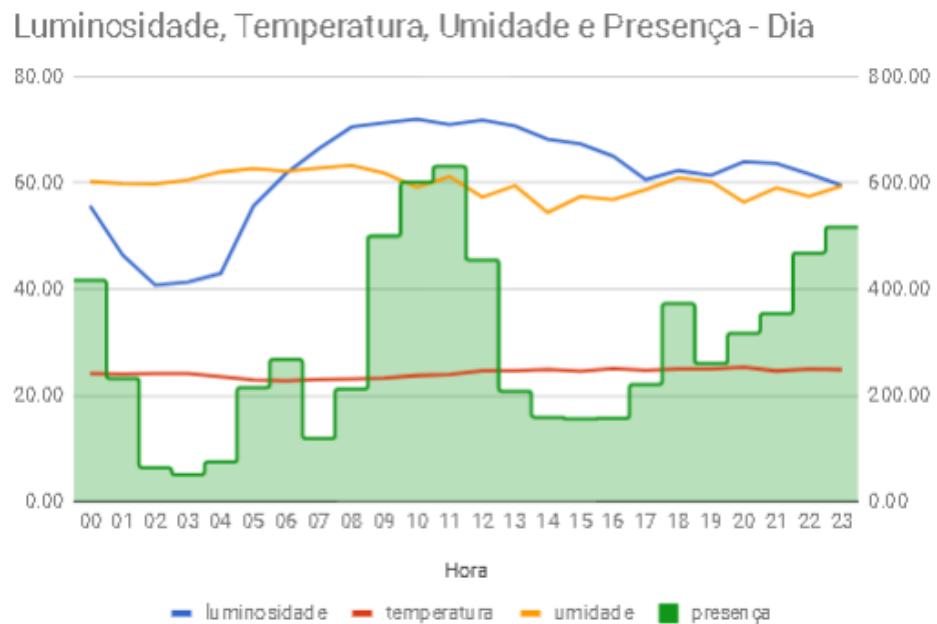


Figura 87: Sensores da Sala/Cozinha - Período

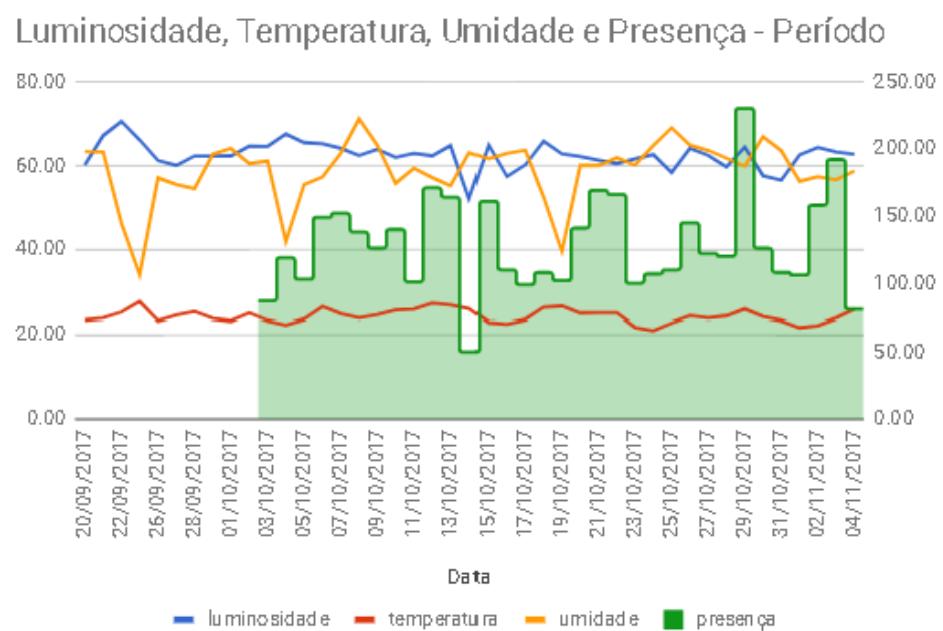


Figura 88: Uso do Rele 1 Sala/Cozinha - Dia

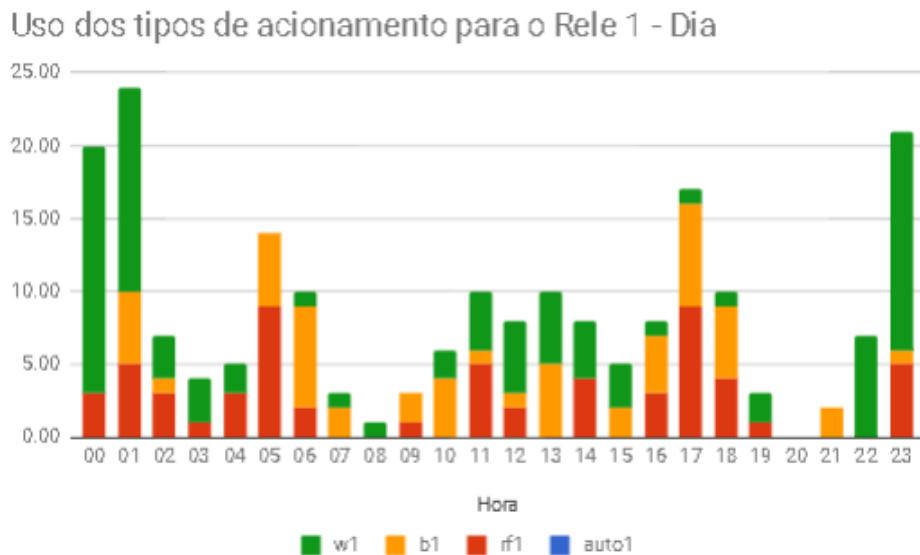


Figura 89: Uso do Rele 1 Sala/Cozinha - Período

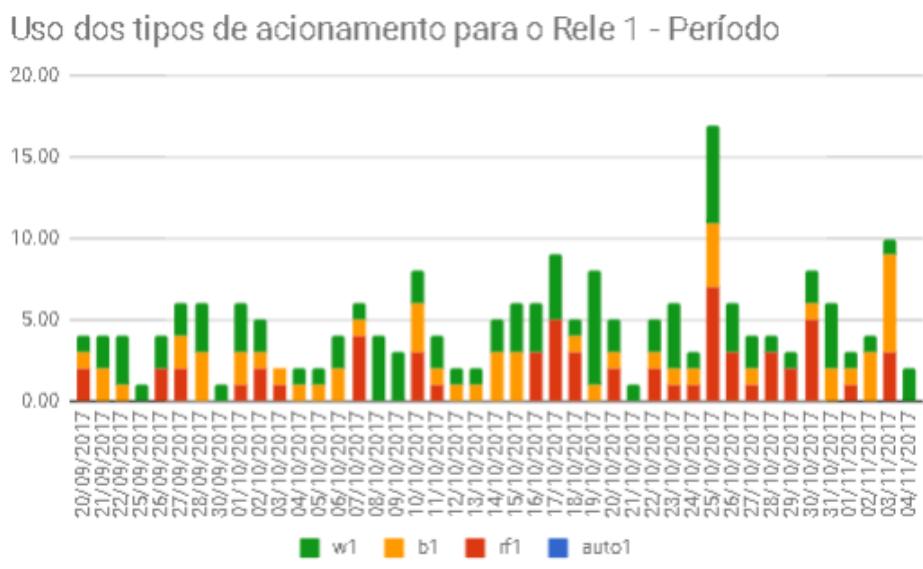


Figura 90: Uso do Rele 2 Sala/Cozinha - Dia



Figura 91: Uso do Rele 1 Sala/Cozinha - Período

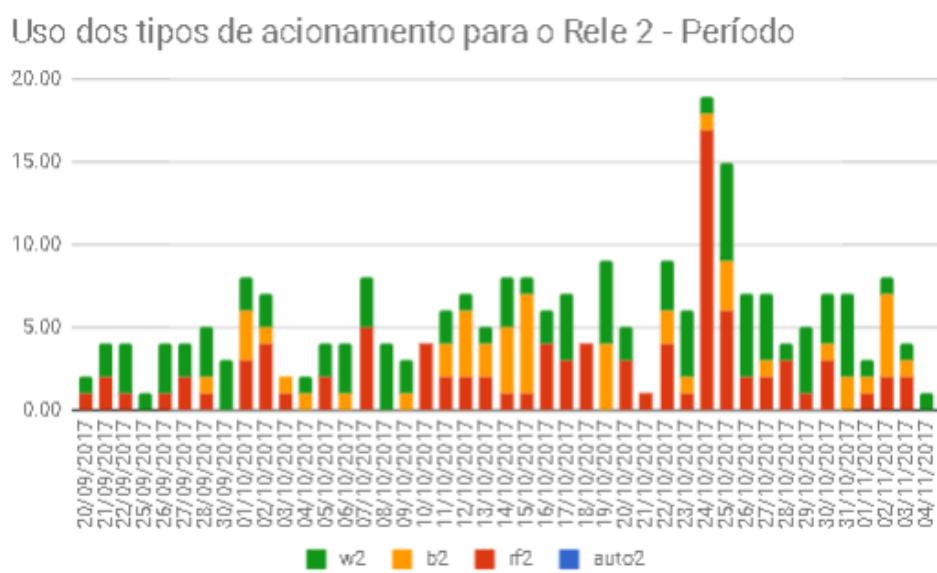


Figura 92: Porta de Acesso - Dia

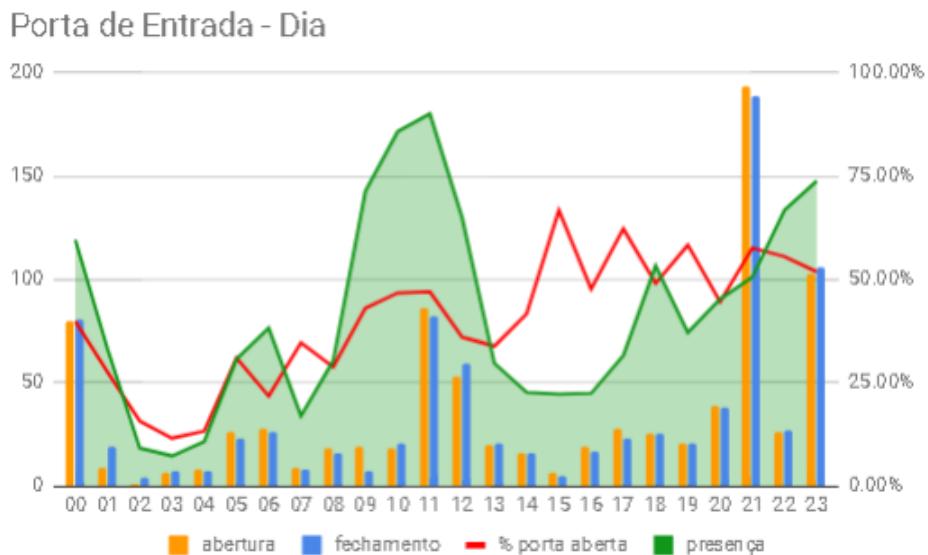


Figura 93: Porta de Acesso - Período

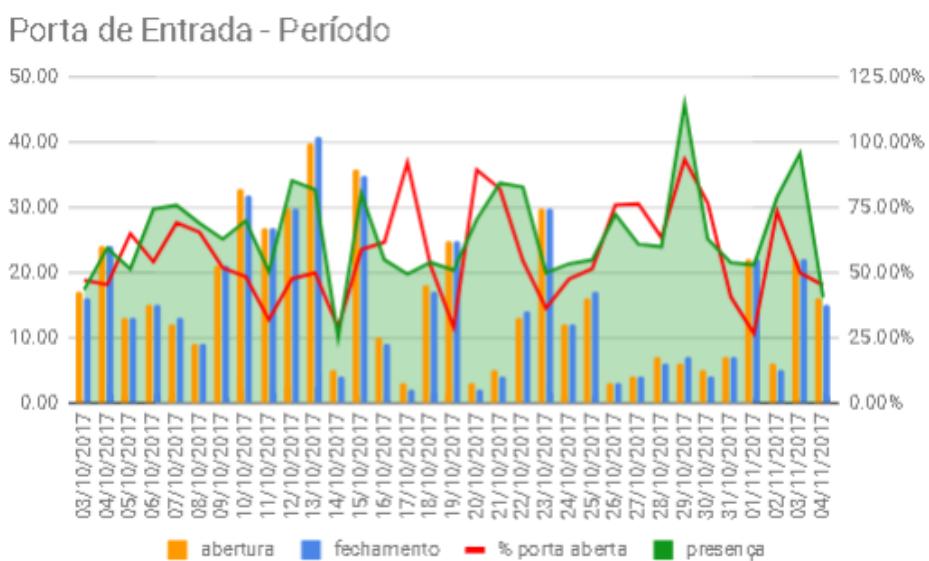


Figura 94: Resumo da Entrada

Módulo	Entrada
Data Início	08/10/2017
Data Fim	02/11/2017
Disponibilidade	Atuadores
Total de reinícios*	91
# Tentativas de conexões WiFi	3
# Desconexões WiFi	77
# Medidas Free Heap (memória livre)	945
Memória livre média (bytes)	26,237
# Tentativas Conexões Blynk	110
Tempo Médio para Conexão Blynk (segundos)	0.32
# Erros m yIP	716
Sensores	Rele 2
Temperatura	Nome
# Medidas	3,011
Média ($^{\circ}$ C)	22.74
Umidade	% Ligado
# Medidas	213
Média (%)	34.31
Presença	% Acionamentos
# Medidas	146
Porta Intermediária da Casa	% Botão Físico
# Medidas	356
% Aberto	3.83%
Portão (sensor de abertura)	# Aberturas
# Aberturas	146
% Aberto	2.12%

Figura 95: Consolidado da Entrada - Período

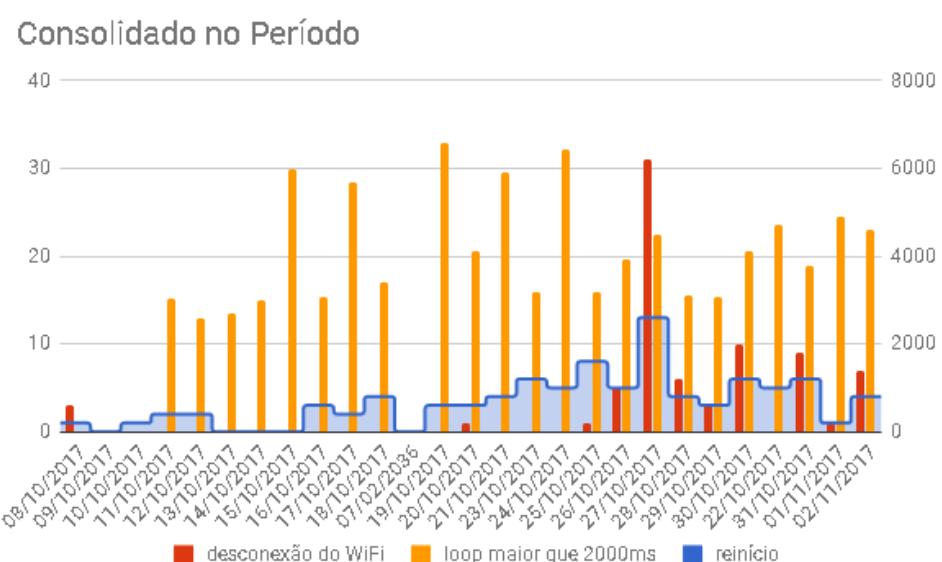


Figura 96: Memória Livre Entrada

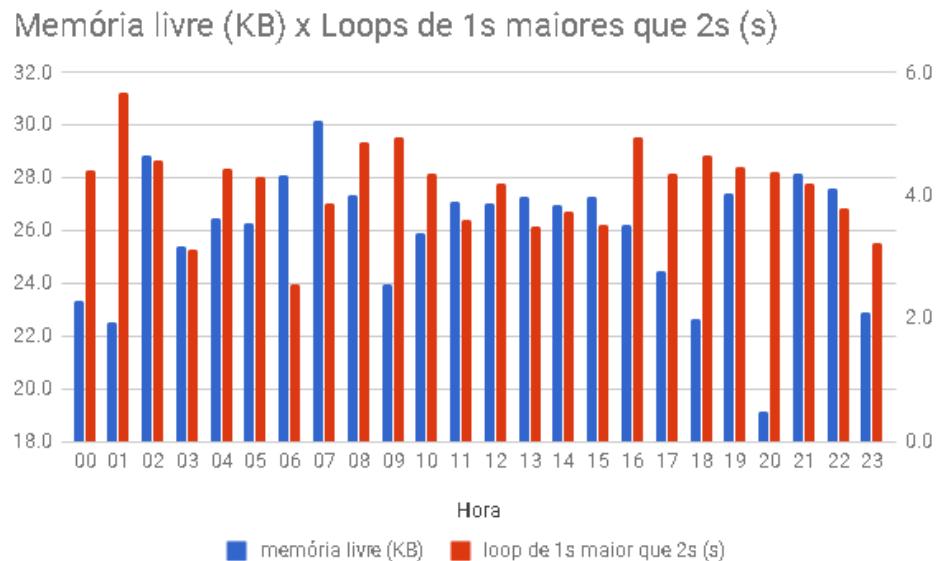


Figura 97: Sensores da Entrada - Dia

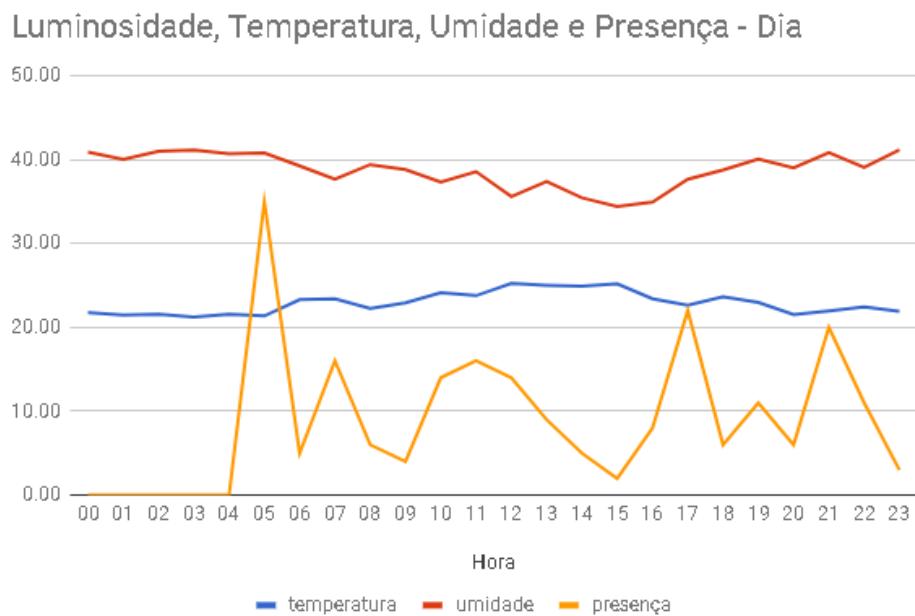


Figura 98: Sensores da Entrada - Período

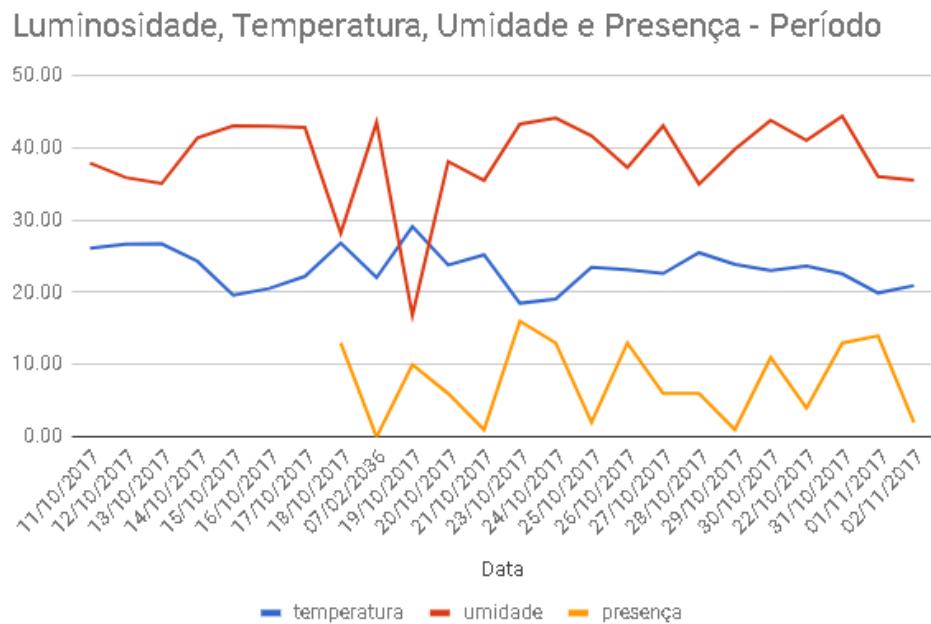


Figura 99: Uso Lâmpada Entrada - Dia

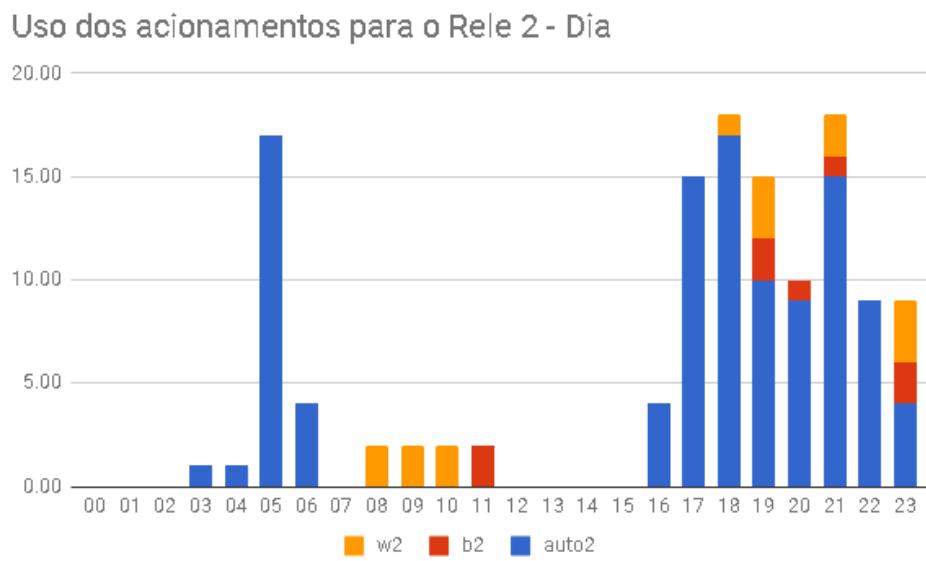


Figura 100: Uso Lâmpada Entrada - Período

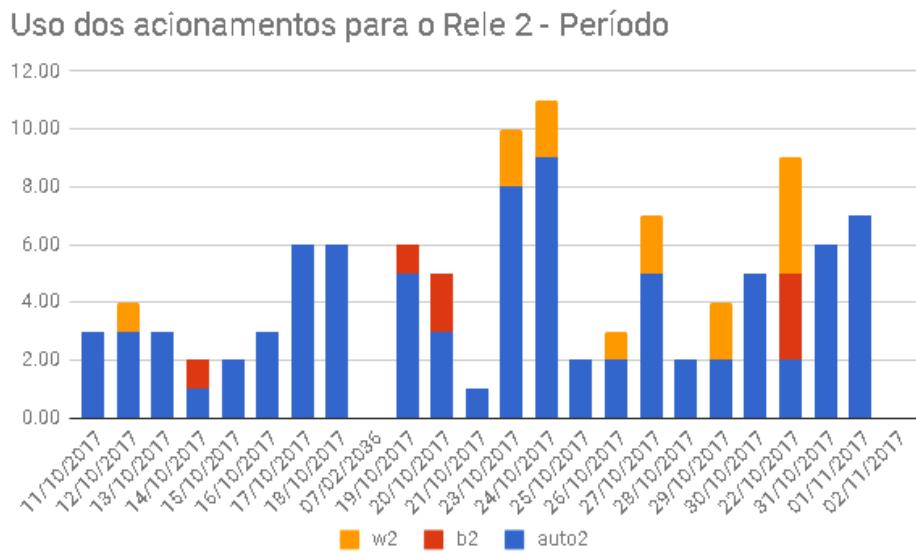


Figura 101: Uso da Entrada - Dia

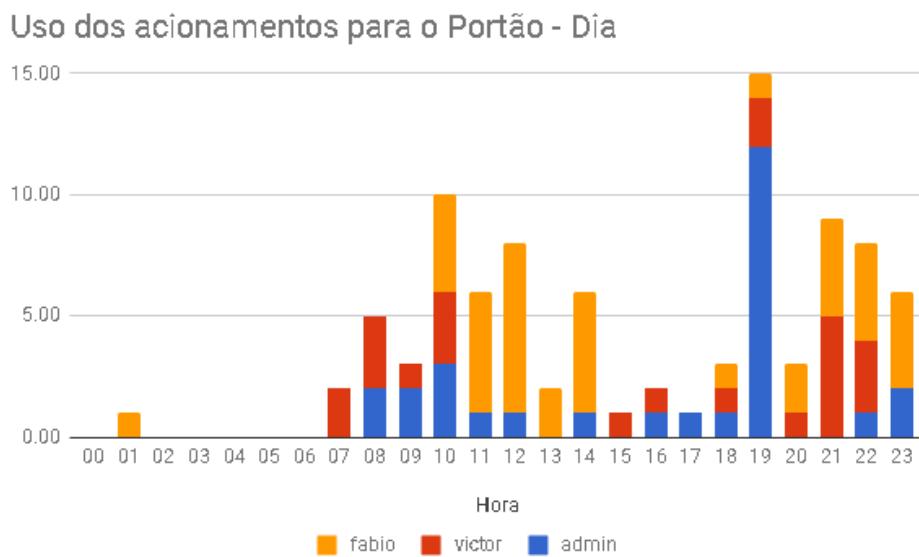


Figura 102: Uso da Entrada - Período

Uso dos acionamentos para o Portão - Período

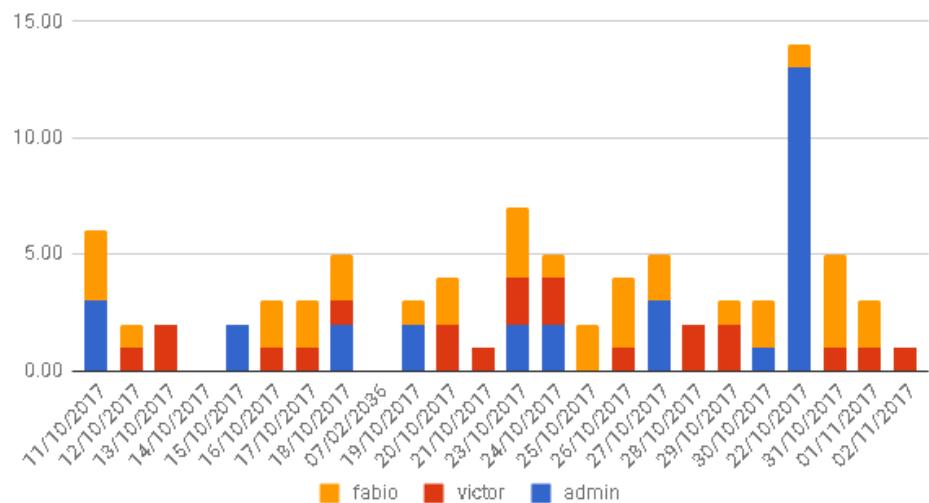
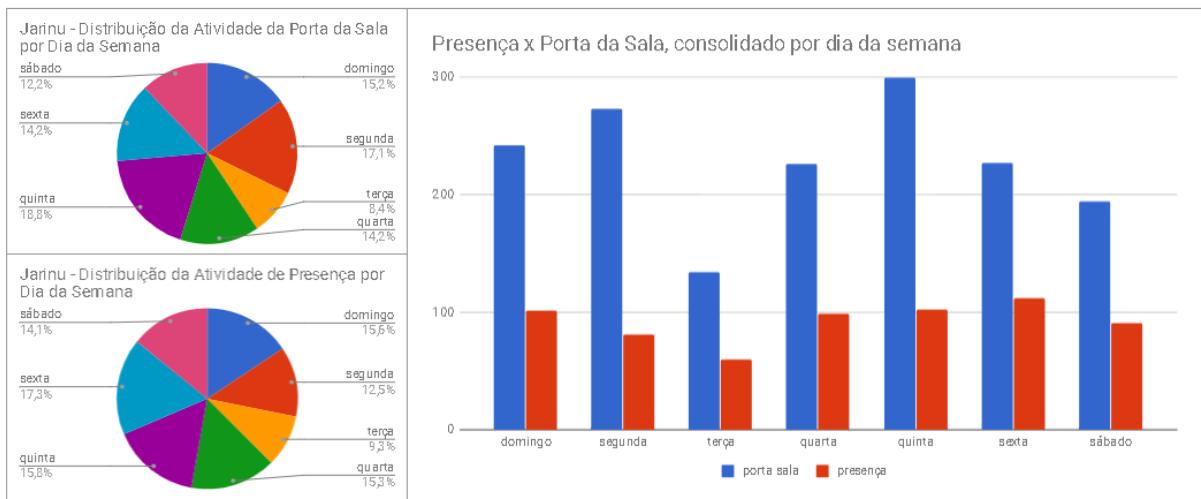


Figura 103: Jarinu - Consolidado por Dia da Semana



ANEXO D – IMAGENS DE CONFIGURAÇÃO PARA O APLICATIVO BACKUP

Figura 104: Configuração de Cores e Layout do Display do Módulo

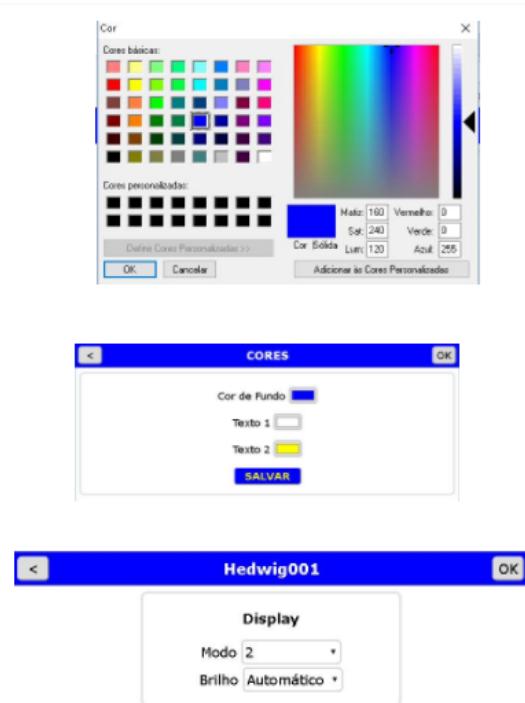


Figura 105: Configurações de alertas, alarmes, reles, log e menu de ferramentas do Aplicativo Backup



Figura 106: Atualização de Firmware, menu de configurações avançadas e DHT



Figura 107: Configurações de Radio Frequênci, Blynk e WiFi

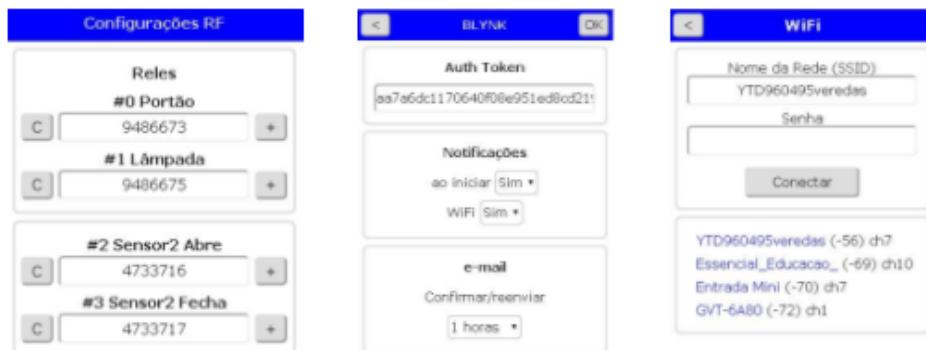


Figura 108: Configurações de Nome e Ponto de Acesso WiFi

