

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

Aos meus pais, Celina e Valdemar, por todo apoio e compreensão dados durante os anos de faculdade e ao longo de toda minha vida. São eles a quem devo todos os frutos que colhi durante a vida acadêmica. A Renata, Mariane, Priscila e Eduardo pela imprescindível companhia e pelos momentos incríveis que compartilhamos durante essa jornada. A Carol, Erika, Flávia, Kelly, Letícia, Luana, Luiza, Patrícia, Paula e Suemi por todas as experiências e aprendizados pelos quais passamos juntas durante os últimos anos.

Daniela

Aos meus pais e irmão, que me apoiaram e incentivaram durante toda minha trajetória. Aos meus amigos, que foram parte fundamental de todos esses anos de faculdade.

Gabriela

Aos meus pais, Rosinete da Silva Possani e Ademir Possani, por sempre me apoiar em todas as decisões e caminhos. A minha noiva, Ellen Ashley Wright, por todo o suporte, compreensão e companhia. Aos meus avós.

Hugo

A meus pais, Fabio e Nair, por todo o apoio em tempo integral durante esses anos. A minha irmã Sabrina, pela inspiração. A equipe de Karatê da Poli, por todo o cumplicismo. A Carol Kimura e família, por todo o auxílio e compreensão.

Victor

Aos nossos orientadores, Reginaldo Arakaki e Marcelo Pita, por todo o conhecimento que nos foi passado e pela atenção dada durante o projeto. A Fabio Hirotugu Hayashi, principal responsável pela montagem e circuito eletrônico dos módulos. Sem sua ajuda, esse trabalho não seria possível.

Grupo

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-chave 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 por objetivo o estudo da arquitetura de um sistema completo para automação residencial, com a utilização do conceito de Internet das Coisas, e de contingências no caso de falhas na comunicação. Nele, são exploradas as tecnologias de comunicação e conectividade entre dispositivos, criando assim uma plataforma acessível e expansível para a automatização e monitoramento 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 híbridos, 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 study on the architecture of a complete house automation system, using the Internet of Things concept and contingency plans for communication failures. 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 client applications, which use 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	28
2	Logotipo do projeto Hedwig	30
3	Camadas da arquitetura usada no Projeto HomeSky. As camadas em verde correspondem às bibliotecas desenvolvidas no trabalho.	35
4	Primeira versão da arquitetura do projeto Hedwig	42
5	Segunda versão da arquitetura do projeto Hedwig	43
6	Diagrama ilustrativo do Módulo de Acesso ao Portão	46
7	Diagrama ilustrativo do Módulo de Quarto/Sala/Cozinha	48
8	Diagrama ilustrativo do Módulo de Aquário	51
9	Diagrama ilustrativo do Módulo de Interface com Sistema de Alarmes . .	52
10	Raspberry Pi 3 Modelo B	54
11	Comparação entre WebSockets e <i>polling</i>	57
12	Comparação entre uma aplicação monolítica (esquerda) e com micro-serviços (direita)	58
13	Diagrama de interação na autenticação por JWT	65
14	Visão alto nível da comunicação no Hedwig	66
15	Diagrama de sequência - transmissão de dados	69
16	Diagrama de sequência - configuração e requisição de ação	70
17	Diagrama de sequência - transmissão de dados do Aplicativo Backup . .	71
18	Diagrama de sequência - configuração e requisição de ação do Aplicativo Backup	72
19	Diagrama de sequência - requisição de ação com senha no Aplicativo Backup	73
20	Rotina de multiplexação de procedimentos no tempo	75
21	Tratamento de indisponibilidade de recursos	76

22	Tratamento de ataque de DoS Local	77
23	Exemplo de estado da fila de saída de mensagens MQTT do módulo	78
24	Diagrama PCB do Módulo Base	79
25	Diagrama elétrico do Módulo Base	80
26	Entradas em A0	81
27	Funcionamento do circuito antitravamento	82
28	Evolução do hardware (de fevereiro/17 a setembro/17)	82
29	Evolução da caixa de proteção e display	83
30	Materiais e preparação da placa padrão	83
31	Posicionamento dos componentes	84
32	Preparação dos displays com o I2C e soldagem	84
33	Preparação das caixas para os módulos	85
34	Caixa protetora, ligação dos botões e cabos de força	85
35	Quatro módulos prontos	86
36	Arquitetura do servidor local	90
37	Componentes e implementação na nuvem	120
38	Monitoramento do servidor na nuvem	124
39	Diagrama de funcionamento do Redux	127
40	Tela correspondente ao Módulo de Acesso	129
41	Tela correspondente ao Módulo Básico	130
42	Telas principais do Aplicativo Backup	135
43	Visão geral da planta de uma casa com o Aplicativo Backup	135
44	Menu principal do Aplicativo Backup	136
45	Teclado para digitação da senha	136
46	Página de configuração TP Link	138
47	Log na página do Aplicativo Backup	142

48	Módulo com cartão SD para coleta local de dados	142
49	Módulos instalados em Santo André	143
50	Controle do Sistema de Alarme e Módulo de Interface de Jarinu	143
51	Módulo Básico e sensor de presença no corredor de Jarinu	144
52	Sensor de abertura das portas da cozinha (à esquerda) e da sala (à direita) .	144
53	Módulo instalado para coleta de dados do nível da caixa d'água	145
54	Relação dos módulos e dados coletados	145
55	Curva diária Santo André – terça-feira	148
56	Temperatura e aquecedor – Módulo do Aquário	150
57	Consolidado Diário – Módulo do Aquário	151
58	Memória e loops – Módulo do Corredor	151
59	Consolidado no período – Módulo do Corredor	152
60	Consolidado diário dos sensores – Módulo do Corredor	153
61	Uso dos acionamentos no período para a lâmpada – Módulo do Corredor .	154
62	Uso dos acionamentos por dia para a lâmpada – Módulo do Corredor . .	154
63	Curva diária – Módulo de Acesso	155
64	Nível da caixa d'água – consolidado diário	156
65	Tempo para encher a caixa d'água – período	157
66	Atividade da porta – Sala de Jarinu	157
67	Atividade da presença – Sala de Jarinu	158
68	Consolidado no período – Jarinu	159
69	EAP do Hedwig	166
70	Tela de login	189
71	Tela de cadastro	190
72	Tela de adição de Morpheus	190
73	Tela de adição de módulo	191

74	Tela de configuração de Morpheus	191
75	Tela de configuração de módulo	192
76	Tela de configurações de usuário	192
77	Tela mostrando se os Morpheus do usuário estão devidamente conectados .	193
78	Ícone do aplicativo na tela inicial do celular	194
79	Tela de login e tela principal com menu aberto no celular	195
80	Dados de sensores apresentados no celular	195
81	Configuração de cores e <i>layout</i> do display do módulo	197
82	Configurações de alertas, alarmes, relés, log e menu de ferramentas do Aplicativo Backup	198
83	Atualização de <i>firmware</i> , menu de configurações avançadas e DHT	198
84	Configurações de radiofrequência, Blynk e Wi-Fi	198
85	Configurações de nome e ponto de acesso Wi-Fi	199
86	Resumo do Módulo do Aquário	201
87	Disponibilidade do Módulo do Aquário – período	202
88	Disponibilidade do Módulo do Aquário – dia	202
89	Memória livre do Módulo do Aquário	203
90	Resumo do Módulo do Corredor	203
91	Sensores do Módulo do Corredor – período	204
92	Uso do relé 2 – Módulo do Corredor – dia	204
93	Uso do relé 2 – Módulo do Corredor – período	205
94	Consolidado diário – Módulo do Corredor	205
95	Resumo do Módulo da Lavanderia	206
96	Disponibilidade do Módulo da Lavanderia – dia	206
97	Disponibilidade do Módulo da Lavanderia – período	207
98	Memória livre do Módulo da Lavanderia	207
99	Sensores da Lavanderia – período	208

100	Sensores do Módulo da Lavanderia – dia	208
101	Uso do relé 1 do Módulo da Lavanderia – dia	209
102	Uso do relé 1 do Módulo da Lavanderia – período	209
103	Uso do relé 2 do Módulo da Lavanderia – dia	210
104	Uso do relé 2 do Módulo da Lavanderia – período	210
105	Consolidado diário do Módulo da Lavanderia	211
106	Resumo do Módulo da Sala/Cozinha	211
107	Disponibilidade do Módulo da Sala/Cozinha – período	212
108	Memória livre do Módulo da Sala/Cozinha	212
109	Sensores do Módulo da Sala/Cozinha – dia	213
110	Sensores do Módulo da Sala/Cozinha – período	213
111	Uso do relé 1 do Módulo da Sala/Cozinha – dia	214
112	Uso do relé 1 do Módulo da Sala/Cozinha – período	214
113	Uso do relé 2 do Módulo da Sala/Cozinha – dia	215
114	Uso do relé 2 do Módulo da Sala/Cozinha – período	215
115	Porta do Módulo de Acesso – dia	216
116	Porta do Módulo de Acesso – período	216
117	Resumo do Módulo da Entrada	217
118	Consolidado do Módulo da Entrada – período	217
119	Memória livre do Módulo da Entrada	218
120	Sensores do Módulo da Entrada – dia	218
121	Sensores do Módulo da Entrada – período	219
122	Uso da lâmpada do Módulo da Entrada – dia	219
123	Uso da lâmpada do Módulo da Entrada – período	220
124	Uso do Módulo da Entrada – dia	220
125	Uso do Módulo da Entrada – período	221

LISTA DE TABELAS

1	Riscos para o aquário	50
2	Análise consolidada de disponibilidade	146
3	Análise consolidada de uso dos relés	147
4	Lista de materiais	182

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>
CGNAT	<i>Carrier-grade NAT</i>
CORS	<i>Cross-Origin Resource Sharing</i>
DDoS	<i>Distributed Denial of Service</i>
DoS	<i>Denial of Service</i>
E/S	Entrada / Saída
EEPROM	<i>Electrically-Erasable Programmable Read-Only Memory</i>
HTTP	<i>Hypertext Transfer Protocol</i>
I2C	<i>Inter-Integrated Circuit</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>
LCD	<i>Liquid Crystal Display</i>
LDR	<i>Light Dependent Resistor</i>
M2M	<i>Machine to Machine</i>
MOOC	<i>Massive Open Online Course</i>
MQTT	<i>Message Queuing Telemetry Transport</i>
NAT	<i>Network Address Translation</i>
NoSQL	<i>Not Only SQL</i>
PIR	<i>Passive Infrared Sensor</i>
PWA	<i>Progressive Web App</i>
QoS	<i>Quality of Service</i>
REST	<i>Representational State Transfer</i>
RF	Radiofrequênciā
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	27
1.1	Motivação	27
1.2	Projeto Hedwig	28
1.2.1	Objetivo	28
1.2.2	Nome do Projeto	30
1.2.3	Logo	30
1.3	Aplicações	30
2	Projetos Relacionados	33
2.1	Sistemas Existentes no Mercado	33
2.1.1	Sistemas Comerciais	33
2.1.2	Sistemas <i>Open-source</i>	34
2.2	Projeto HomeSky	34
3	Arquitetura: Especificação de Negócios e de Requisitos	37
3.1	Partes Interessadas	37
3.2	Requisitos	38
3.2.1	Requisitos Funcionais	38
3.2.2	Requisitos Não-funcionais	39
3.2.3	Funcionalidades por Nível de Conectividade	40
4	Arquitetura: Solução Técnica - Mecanismos	41
4.1	Visão Geral	41
4.2	Evolução Arquitetural	41
4.3	Módulos	44

4.3.1	Módulos Base	45
4.3.1.1	ESP8266	45
4.3.2	Módulo de Acesso	46
4.3.3	Módulo de Quarto/Sala/Cozinha	48
4.3.4	Módulo de Aquário	50
4.3.5	Módulo de Interface com Sistema de Alarmes	52
4.4	Controlador Local	53
4.4.1	Raspberry Pi	53
4.5	Servidor na Nuvem	54
4.5.1	Computação em Nuvem	54
4.5.2	Banco de Dados Não-relacional	55
4.5.3	Banco de Dados em Memória	55
4.5.4	WebSockets	56
4.5.5	Arquitetura de Microsserviços	57
4.6	Cliente Web	60
4.6.1	<i>Progressive Web Apps</i>	60
4.6.1.1	Contexto	60
4.6.1.2	Conceito	61
4.6.1.3	Tecnologias e Técnicas	62
4.6.1.4	Aplicações	63
4.6.2	<i>JSON Web Tokens</i>	63
4.6.2.1	Definição	63
4.6.2.2	Autenticação	64
4.7	Comunicação	65
4.7.1	Comunicação entre Controlador Local e Módulos	66
4.7.2	Comunicação entre Controlador Local e Nuvem	67

4.7.3	Comunicação entre Nuvem e Aplicativos	68
4.7.4	Comunicação entre Módulos e Aplicativos	68
4.7.5	Diagramas de sequência	68
5	Arquitetura: Tecnologia e Implementação	75
5.1	Módulos	75
5.1.1	Rotinas	75
5.1.1.1	Multiplexação no Tempo	75
5.1.1.2	Tratamento de Indisponibilidade	76
5.1.1.3	DoS Local (<i>Evil Twin</i>)	76
5.1.1.4	Comunicação por MQTT	77
5.1.2	Diagrama PCB	78
5.1.3	Montagem	82
5.1.3.1	Procedimento de Validação do Módulo	86
5.2	Controlador Local - Morpheus	87
5.2.1	Descrição	87
5.2.2	Plataforma	87
5.2.3	Tecnologias Utilizadas	87
5.2.4	Características e Recursos	90
5.2.5	Protocolo para Troca de Mensagens com Módulos	92
5.2.5.1	Tópicos	92
5.2.5.2	Regras de Negócio	93
5.2.5.3	Definição de Interfaces	93
5.2.5.4	Definição das Mensagens	94
5.2.5.5	Testes Realizados da Comunicação Morpheus e Módulos .	105
5.2.6	Protocolo para Troca de Mensagens com a Nuvem	109
5.2.6.1	Morpheus	109

5.2.6.2	Nuvem	109
5.2.7	Configurações	112
5.2.7.1	Configuração do Mosquitto	112
5.2.7.2	Guia de Instalação	113
5.2.7.3	Criação dos Certificados	115
5.2.7.4	Senhas	116
5.2.7.5	Casos de Teste para Controle de Acesso nos Tópicos MQTT entre Módulos e Nuvem	116
5.3	Servidor na Nuvem	117
5.3.1	Descrição	117
5.3.2	Características da Implementação	117
5.3.3	Tecnologias Utilizadas	118
5.3.4	Tratamento de Eventos	120
5.3.5	Infraestrutura	123
5.3.6	Segurança	123
5.3.7	Operação	123
5.4	Aplicativo de <i>dashboard</i>	124
5.4.1	Descrição	124
5.4.2	Requisitos	124
5.4.2.1	Requisitos Funcionais	124
5.4.2.2	Requisitos Não-funcionais	126
5.4.3	Tecnologias Utilizadas	126
5.4.4	Interface	128
5.4.4.1	Identidade Visual	128
5.4.4.2	Telas	129
5.4.5	Interações	130
5.4.5.1	Dados	130

5.4.5.2	Ações	130
5.4.5.3	Conectividade	131
5.4.5.4	Aplicativo na Tela Inicial do Dispositivo Móvel	131
5.4.6	Implantação	131
5.4.7	Segurança	131
5.4.8	Performance	132
5.5	Aplicativo Backup	132
5.5.1	Requisitos	132
5.5.2	Tecnologias Utilizadas	133
5.5.3	Navegação	134
5.5.4	Configurações	136
5.5.5	Abertura de Porta do Roteador	137
5.5.6	Controle Remoto	138
5.5.7	Notificações	139
5.5.8	Offset de Temperatura e Umidade	139
5.5.9	<i>Hard Reset</i>	139
5.5.10	Setup Inicial	140
6	Arquitetura: Tecnologia - Coleta e Análise de Dados	141
6.1	Coleta	141
6.2	Análise	146
7	Arquitetura: Tecnologia - Aprendizado de máquina	161
7.1	Conceito	161
7.2	Relação com o Projeto Hedwig	161
7.3	Implementação	162
7.3.1	Linguagem, Ferramentas e Bibliotecas	162
7.3.2	Etapas	163

7.3.3	Algoritmos	163
7.3.4	Tratamento de Dados	163
7.3.5	Seleção de Características	163
8	Metodologia	165
8.1	Gerência do Projeto	165
8.1.1	Gerência de Escopo	165
8.1.2	Gerência de Aquisição	167
8.1.3	Gerência de Processos de Software	167
8.1.4	Gerência de Comunicação	168
8.1.5	Gerência de Riscos	168
8.2	Pesquisa Bibliográfica	168
8.3	Ferramentas e Tecnologias	169
8.4	Método de Testes	169
9	Conclusões	171
Referências		173
Anexo A – Códigos das aplicações desenvolvidas		179
Anexo B – Lista de materiais para montagem dos módulos		181
Anexo C – Script para adição de nova residência ao Mosquitto		183
Anexo D – Configurações do Mosquitto		185
Anexo E – Telas do aplicativo web — dashboard		189
Anexo F – Imagens de configuração para o aplicativo backup		197
Anexo G – Dados coletados de 10/09/2017 a 13/11/2017		201

1 Introdução

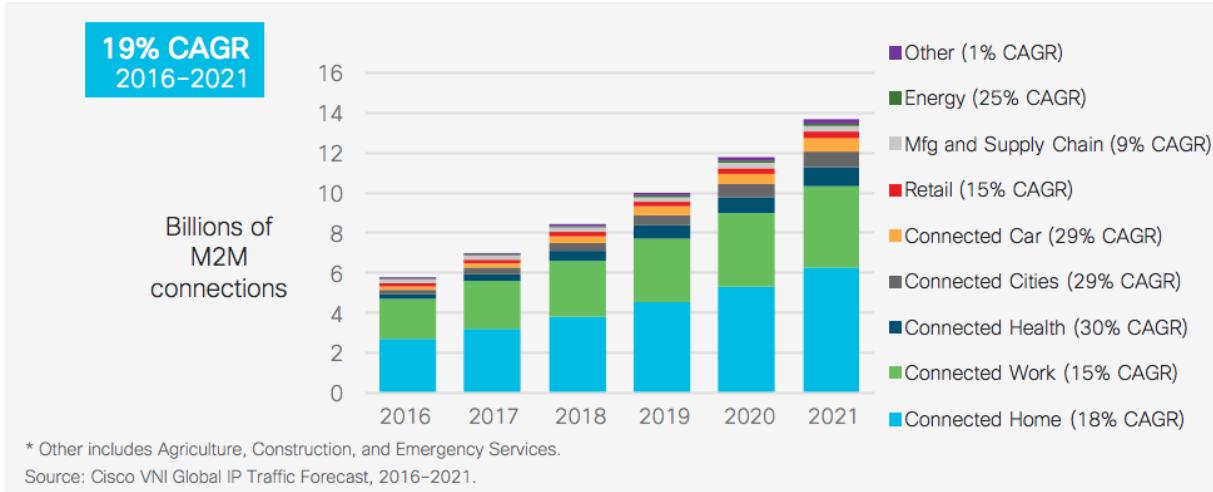
1.1 Motivação

Há uma expectativa de que, no ano de 2017, o número de casas inteligentes aumente em aproximadamente 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 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) aparam 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 nos dispositivos presentes na residência do usuário final do sistema. O projeto disponibiliza módulos físicos, prontos para serem instalados e configurados, 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, são 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.

Aprendizado de máquina

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*, 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 no mercado norte-americano. Alguns dos sistemas mais populares nessa linha são o Amazon Echo e o Google Home.

O Amazon Echo¹ 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 Home² 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

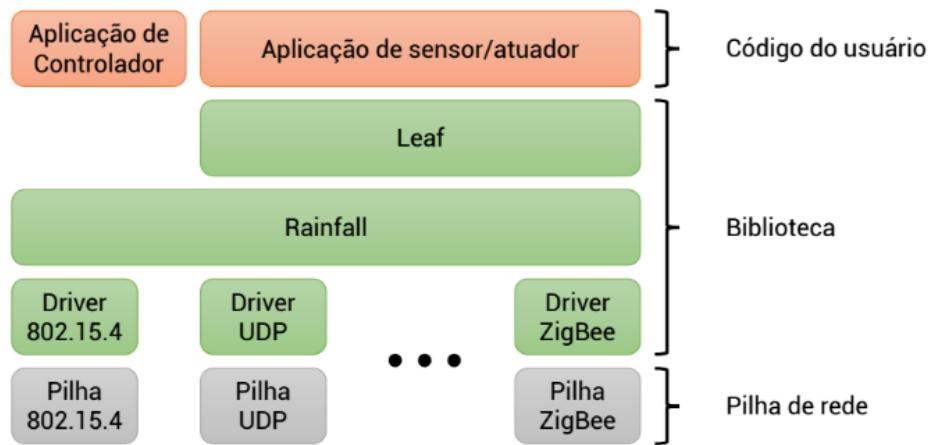
O Projeto HomeSky (MURAMATSU; RODRIGUES; GALLEG, 2016) é um Trabalho de Conclusão de Curso desenvolvido por alunos de Engenharia de Computação na Escola Politécnica da Universidade de São Paulo. Com o objetivo de fomentar iniciativas de desenvolvimento na área de casas inteligentes, o trabalho focou-se 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 Arquitetura: Especificação de Negócios e de Requisitos

3.1 Partes Interessadas

Um dos passos iniciais na elaboração de um projeto é a determinação das partes interessadas (*stakeholders*). 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, podem-se destacar os seguintes grupos dentre os potenciais consumidores:

- Pessoas que moram sozinhas e seus familiares, na sua maioria idosos com boa saúde física;
- 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, obtém-se 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, obtém-se uma estimativa de 35 mil consumidores em potencial;
- Considerando que cerca de 70% das residências reduziram o consumo com campanhas de uso de água em 2015 (G1, 2015), e supondo que 5% desse total se interessaria

em adquirir produtos que auxilia nessa redução, obtém-se 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 controle 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 que 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 deve identificar e se 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 realizada 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 requisições inválidas.

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 local, por exemplo.

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

4 Arquitetura: Solução Técnica - Mecanismos

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 a contratos (interfaces). Com base no requisito RNF-1, optou-se por um projeto modular, no qual 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, a qual foi implementada.

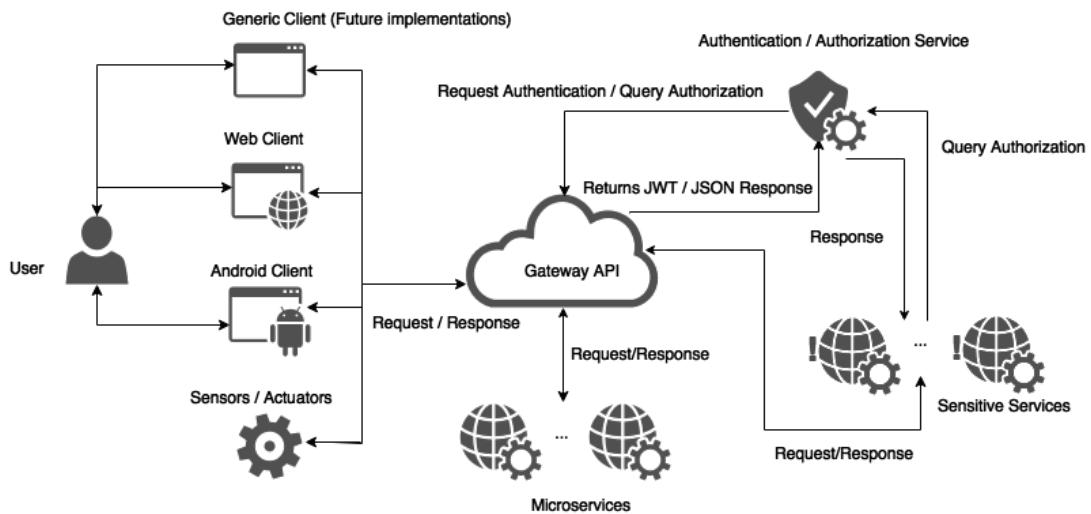
4.2 Evolução Arquitetural

O processo de definição da arquitetura foi iterativo, seguindo 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 uma grande 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 cada requisição necessita solicitar um amplo número de serviços para que possa completar a sua tarefa. Há também um 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 passariam todas as requisições válidas no fluxo de comunicação com a casa. A inserção do *gateway* cria um ponto único de falha, 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, novamente 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. 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. Entretanto, alguns módulos poderiam ficar 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 sua 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, poderia 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 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 aprendizado de máquina (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 do servidor local, os dados que os módulos tentam enviar ao servidor local serão perdidos, entretanto o aplicativo backup continua podendo se comunicar diretamente com os módulos, para ter acesso aos serviços de extrema importância.

4.3 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 é 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 são 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 a conexão não está disponível. Para mitigar o travamento, um sinal de *keep alive* é monitorado, e um circuito antitravamento deve ativar o *hard reset* (*reset* por hardware), ou então uma rotina de *soft reset*, 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 é utilizado em modo *StandAlone* como principal processador e responsável pela conexão dos diferentes módulos de automação. Suas duas principais

plataformas de desenvolvimento são Wemos¹ e NodeMCU². O projeto utiliza o Wemos D1 Mini, versão compacta do Wemos D1 R2.

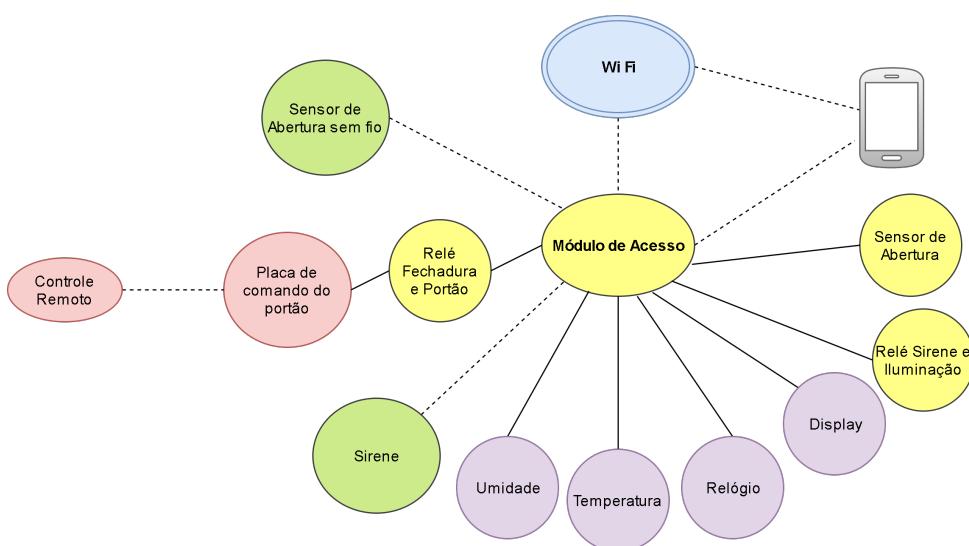
O ESP8266 possui um modo de operação de baixa potência (*sleep mode*) em que o consumo de bateria fica muito menor — em contrapartida, o número de funcionalidades é limitado. Pode-se utilizar 7 portas de E/S digitais e uma porta de entrada analógica. Duas portas não são acessíveis, pois são utilizadas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



¹<https://www.wemos.cc/>

²<http://nodemcu.com/>

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 — são 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) estão 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 ou 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 a 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

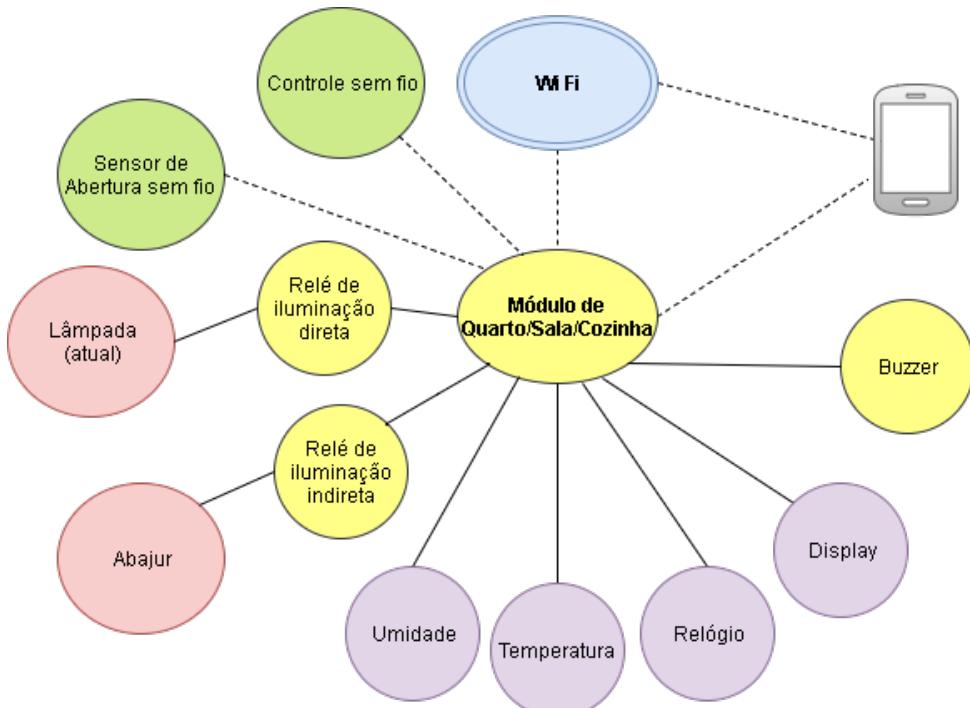
web, mas sim com o aplicativo emergencial usando a ativação do *Access Point*, desenvolvido para operar diretamente com os módulos, sem intermédio do servidor local e dos serviços remotos.

Por meio das credenciais disponíveis no sistema, é possível saber qual dos usuários solicitou a abertura do portão. A persistência destes acessos pode ser analisada e, utilizando-se técnicas de aprendizado de máquina, perfis de acesso podem ser determinados, e evoluir até que o sistema saiba quando houver um acesso em horário inesperado e possa 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 seja confirmado que não se trata de 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 ele 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 (lâmpada e abajur) em vermelho, enquanto o controle e sensor de abertura possuem comunicação sem fio. Em roxo, representam-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 escolher a vestimenta, optar por levar guarda-chuva na ida para o trabalho ou decidir 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 por meio de circuito baseado em LDR para não apresentar brilho muito intenso quando todas as luzes estiverem desligadas. Para o controle da iluminação, há diferentes tempos de desligamento. Por exemplo, quando ocorre controle manual pelo botão presente no módulo, o tempo pode ser maior (e.g. 30 minutos). Já pelo modo automático, quando a luz já foi ligada pelo próprio módulo, o tempo para desligamento pode ser menor (e.g. 4 minutos).

Com o monitoramento da presença, há um reinício 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 acidental	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 para circulação de água e um aquecedor, com cerca de 50 peixes de pequeno porte, que são sensíveis à variação brusca de temperatura.

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, e a adição é realizada pela bomba, que introduz água limpa, sem cloro e com 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.

Figura 8: 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, com disparo de alerta pelo aplicativo quando não estiver no nível esperado — o que 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, evitando superaquecimento devido a mal funcionamento do termostato — isso é feito por meio de ligação em série com o termostato do aquário;
4. Nos casos de perigo acima descritos, também disparar alerta sonoro 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 aspectos específicos: um relativo à presença, possuindo sensores no corredor e na sala da residência, e outro relativo à porta da sala, com um sensor único na porta da residência. O módulo implementado foi instalado em uma residência em Jarinu - SP.

Figura 9: 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 a 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 relé NF (normalmente fechado) em série com a alimentação do roteador (no 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, até um máximo de 30 minutos.

Outra dificuldade encontrada em sua instalação em campo foi a obtenção de endereço IP dinâmico em uma rede em que 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, existe o 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 foi 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 5.2.

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 as de um computador pessoal (PC): navegação na Internet, reprodução de vídeo, processamento de texto, dentre outros. No projeto, é 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 10: 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 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 de 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 11.

A sua arquitetura diminui as latências de comunicação em relação às técnicas de *polling*, que envolvem simplesmente a realização periódica de requisições HTTP, 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 11: Comparação entre WebSockets e *polling*



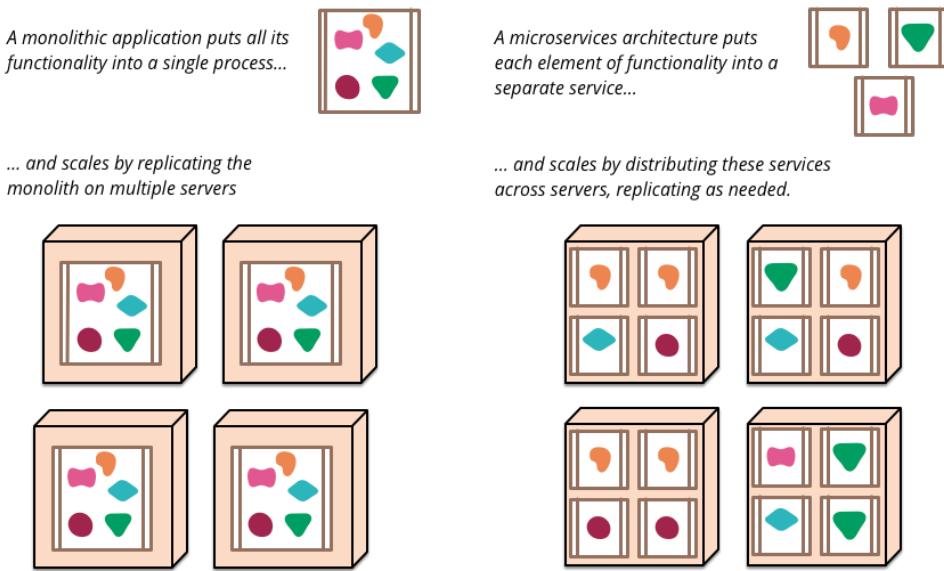
Fonte: (LUBBERS; GRECO, 2016)

4.5.5 Arquitetura de Microsserviços

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 balanceador de carga — 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 12: 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 dela. Contudo, existe uma série de características que projetos usando essa arquitetura compartilham. Detalham-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 a implantação automatizada 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 mais amplo, e muitas de suas implementações podem não seguir certos pontos apresentados como aspectos dos microsserviços. Pode-se citar como por exemplo dessa situação 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 o desenvolvimento de múltiplos aplicativos clientes 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 em diversas plataformas, seja em navegadores ou em sistemas operacionais nativos de smartphones.

A fim de demonstrar como o usuário final poderia interagir com o sistema em sua totalidade, optou-se por desenvolver uma aplicação web. Esse tipo de aplicação foi escolhida devido à vasta quantidade de bibliotecas, *frameworks*, ferramentas e IDEs disponíveis. 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 de 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 melhor 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 downloads 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. Podem-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 diversas plataformas com o mesmo código, apesar de que para obter tal resultado, são impostas diversas 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 móvel 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 - podem 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 URLs.

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 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 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, o resultado da 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 `Authentication` do cabeçalho HTTP. Somente requisições contendo *tokens* válidas 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
    ↪ .TJVA950rM7E2cBab30RMHrHDcEfijoYZgeFONFh7HgQ
```

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 (W3CSCHOOLS, 2017).

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).

Figura 13: Diagrama de interação na autenticação por JWT



Fonte: (JWT, 2016)

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

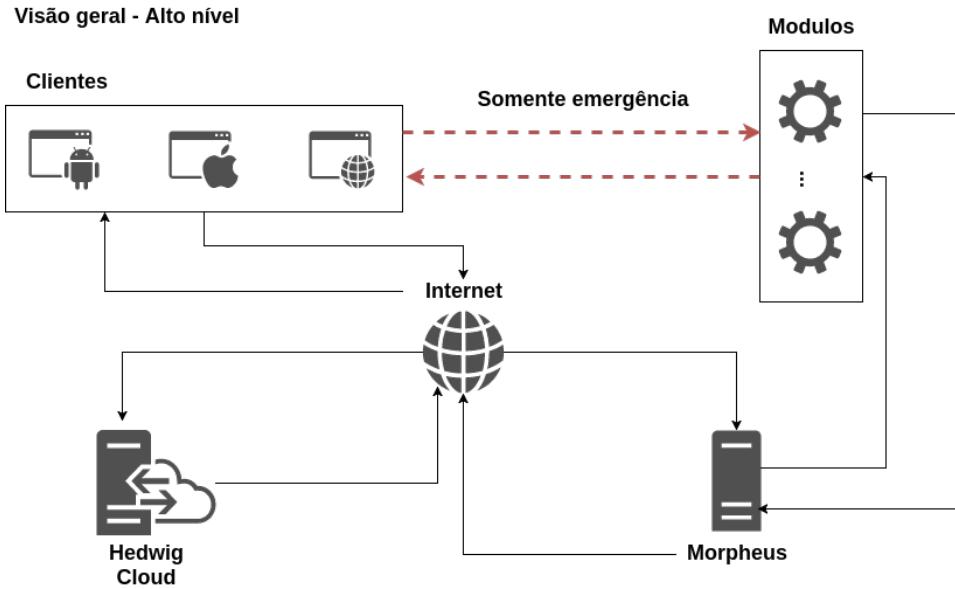
Conforme explicado anteriormente, neste projeto são utilizados tanto protocolos de comunicação próprios quanto os criados por terceiros. A arquitetura desenvolvida aqui busca viabilizar a robustez do sistema, trabalhando em ambos os níveis local e remoto, com o usuário tendo o controle de sua casa por meio de um smartphone ou computador pessoal.

O 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 então realiza a comunicação com os módulos.

A arquitetura de comunicação é representada pela Figura 14 com um alto nível de abstração. Detalhes sobre a implementação e as mensagens trocadas entre servidor local, módulos e nuvem serão vistos com menor granularidade na Seção 5.2.

Figura 14: Visão alto nível da comunicação no Hedwig



4.7.1 Comunicação entre Controlador Local e Módulos

A infraestrutura de comunicação entre o servidor local e os módulos com 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 e é orientado à sessão, diferentemente do protocolo HTTP, localizado na mesma camada (LAMPKIN et al., 2012).

O protocolo MQTT é do tipo *Pub/Sub* (de *publisher/subscriber*) e é estritamente orientado a 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 os *wildcards* * e +, os quais são válidos para mais de um nível e um único nível, respectivamente (MQTT-OASIS, 2014).

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* permite ou nega 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.

O protocolo MQTT também oferece três tipos de QoS (*Quality of Service*), possibilizando: 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 consequentemente aumenta o *overhead*.

As mensagens são transmitidas em texto puro, e é necessário estabelecer um protocolo para a sua utilização. Foi desenvolvido um protocolo de fácil utilização e com baixo *overhead*, mas que pudesse ser expansível e flexível aos casos de uso desejados.

O *broker* Mosquitto³ é utilizado no Hedwig. Ele foi escolhido por ser amplamente adotado em projetos de IoT, além de ser *open-source* e ter licença abrangente (MIT). Entretanto, há outras diversas possibilidades, como o HiveMQ, adotado no projeto HomeSky e com grande uso em aplicações enterprise (HIVEMQ, 2017).

4.7.2 Comunicação entre Controlador Local e Nuvem

Inicialmente, foi proposto um modelo arquitetural para a comunicação com a nuvem no qual existiriam *endpoints* para requisições HTTP tanto do lado da casa quanto do lado da nuvem. Assim, quando o Morpheus precisasse enviar uma mensagem, seria necessário realizar uma chamada ao *endpoint* correspondente. Neste sentido (Morpheus para nuvem) não há nenhum problema, pois é possível garantir as configurações avançadas de segurança adequadas, bem como a utilização de平衡adores de carga e servidores terceiros para lidar com ataques do tipo DoS (AKAMAI, 2017), conforme RNF-6.

O problema, no entanto, está em garantir a segurança e usabilidade do lado da casa. Primeiramente, os IPs residenciais não são fixos, sendo 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 seria 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,

³<https://mosquitto.org/>

ainda, um problema mais grave e mais difícil de ser contornado. Com essa arquitetura, o Morpheus também seria um servidor do ponto de vista da nuvem, e qualquer dispositivo poderia tentar fazer uma requisição em um dos *endpoints* disponíveis. Mesmo que sejam checados os dados da requisição para garantir que ela seja válida, tem-se 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.

Uma forma de contornar o problema se encontra no uso de WebSockets, protocolo detalhado na Seção 4.5.4. Assim, o Morpheus se comporta como um cliente em relação à nuvem e é sempre ele que abre uma conexão. Essa conexão se mantém aberta e forma um caminho *full-duplex*, de 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, ou seja, de forma transparente ao cliente final.

4.7.3 Comunicação entre Nuvem e Aplicativos

Similarmente à comunicação entre controlador local e nuvem, a troca de informações entre nuvem e aplicativos clientes em geral é feita por meio de WebSockets. Como aplicações web podem ter como requisito oferecer suporte à navegadores mais antigos, é usado um *fallback* para um mecanismo de *polling* caso não haja suporte para WebSockets.

4.7.4 Comunicação entre Módulos e Aplicativos

Em caso de indisponibilidade da rede local ou internet, a comunicação é feita diretamente entre módulos e aplicativo backup com escopo local (por meio de HTTP). Esse canal de comunicação fica aberto para uso somente nesses casos, para evitar exposição desnecessária a possíveis ataques. O módulo atua como servidor e o aplicativo como cliente, sendo que cada aplicativo pode se conectar diretamente ao módulo que desejar.

4.7.5 Diagramas de sequência

Seguem diagramas de sequência UML para ilustrar a comunicação fim-a-fim utilizada no projeto:

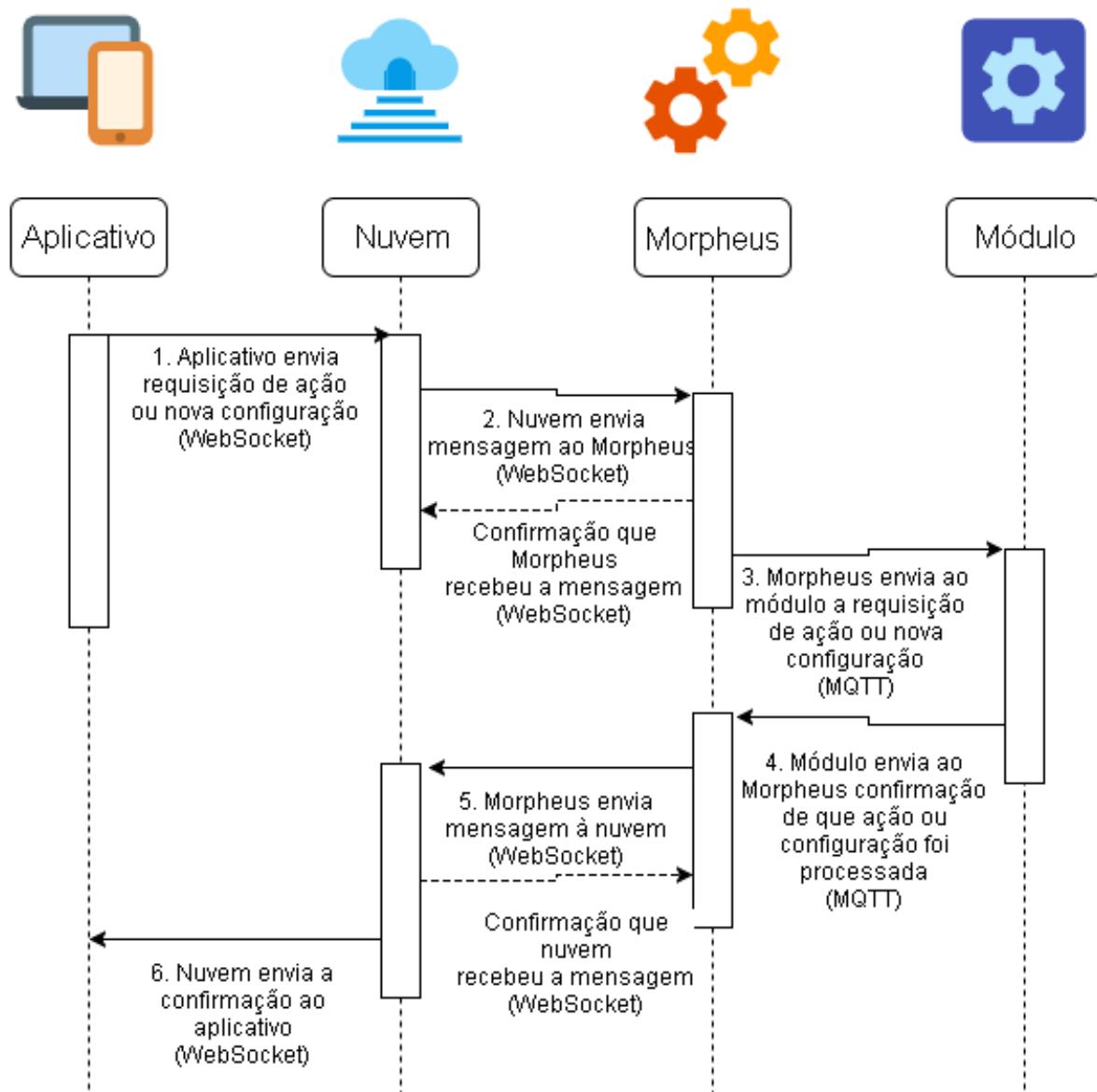
Figura 15: Diagrama de sequência - transmissão de dados



No sistema instalado nas residências para coleta de dados, ocorre atualização a cada 1 minuto. No caso de mudança considerável no estado de algum sensor ou acionamento de algum relé, há uma atualização para o aplicativo com tempo de resposta de 1 segundo.

Para os casos de mudança de configuração e atuação (acionamento de relés), temos:

Figura 16: Diagrama de sequência - configuração e requisição de ação

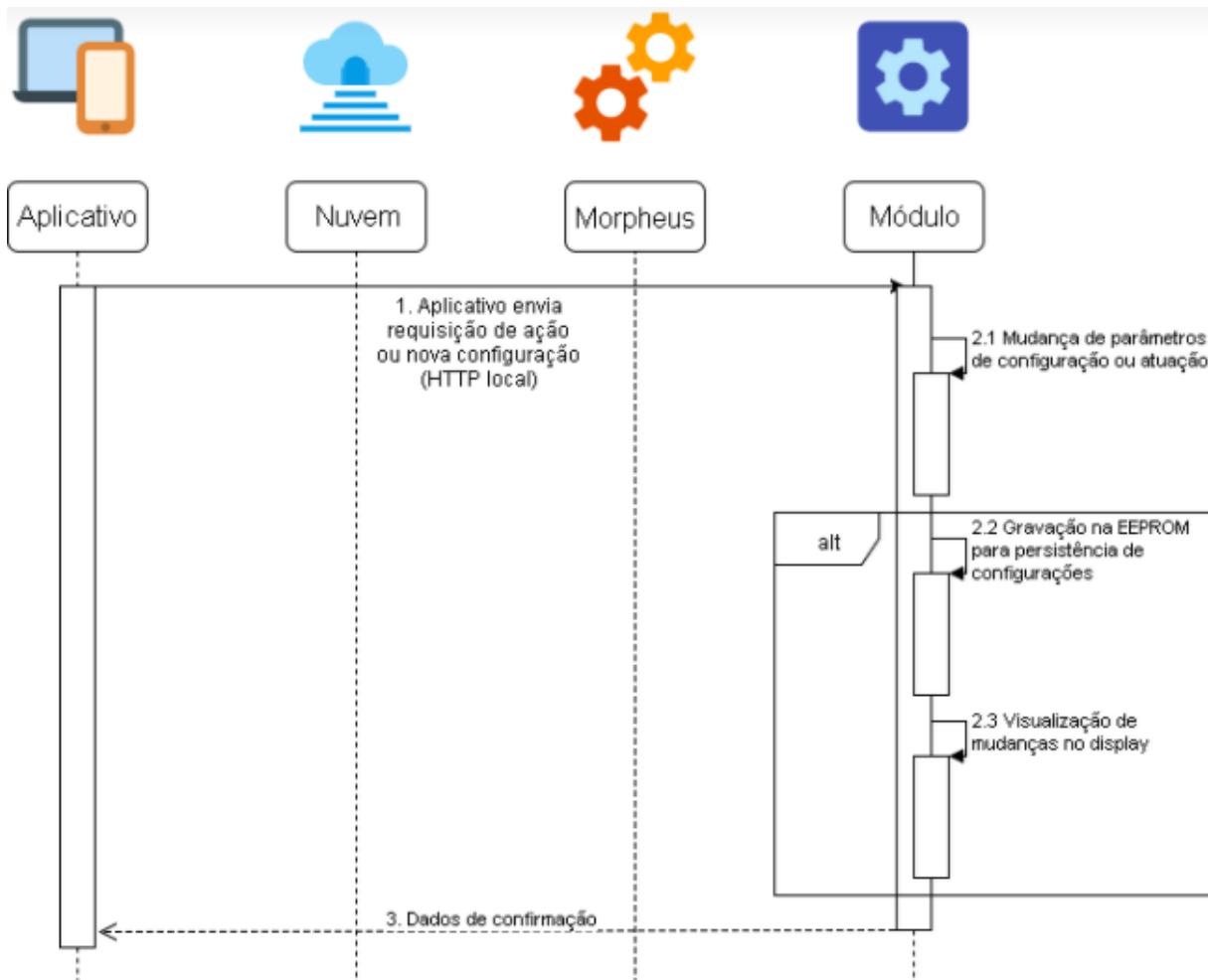


Se não houver conexões disponíveis, os procedimentos de atualização de estado de dados, requisição de ação e configuração ocorrem diretamente entre módulos e aplicativos. O usuário ainda pode atuar no sistema através de controles de rádio frequência e botões físicos, utilizando uma comunicação direta entre usuário e módulo (sem uso do aplicativo).

Figura 17: Diagrama de sequência - transmissão de dados do Aplicativo Backup



Figura 18: Diagrama de sequência - configuração e requisição de ação do Aplicativo Backup



É importante destacar que não há possibilidade do usuário requisitar múltiplos acionamentos (causando muitas trocas de estado em um período pequeno de tempo), uma vez que a requisição é realizada a partir do estado atual do relé (dessa forma, se este estiver ativo, múltiplos cliques do usuário mandam a requisição “desligar”).

Outro ponto é a inclusão de um mecanismo simples para segurança do sistema, mesmo no canal de comunicação emergencial de escopo local, mostrando uma possibilidade de aumento do nível de segurança do sistema (maiores detalhes estão na seção).

Figura 19: Diagrama de sequência - requisição de ação com senha no Aplicativo Backup



5 Arquitetura: Tecnologia e Implementação

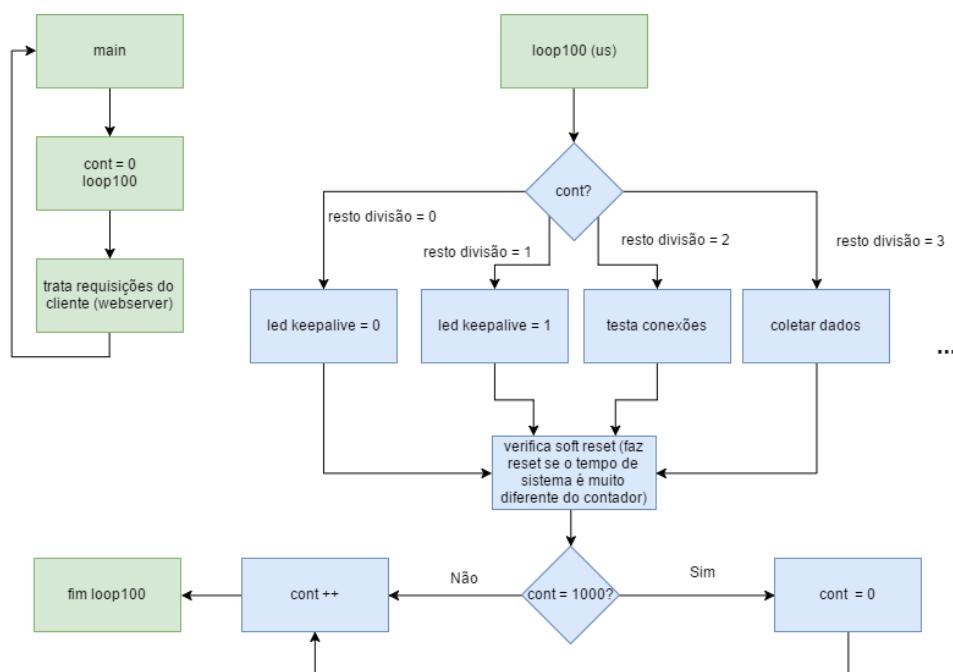
5.1 Módulos

5.1.1 Rotinas

5.1.1.1 Multiplexação no Tempo

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

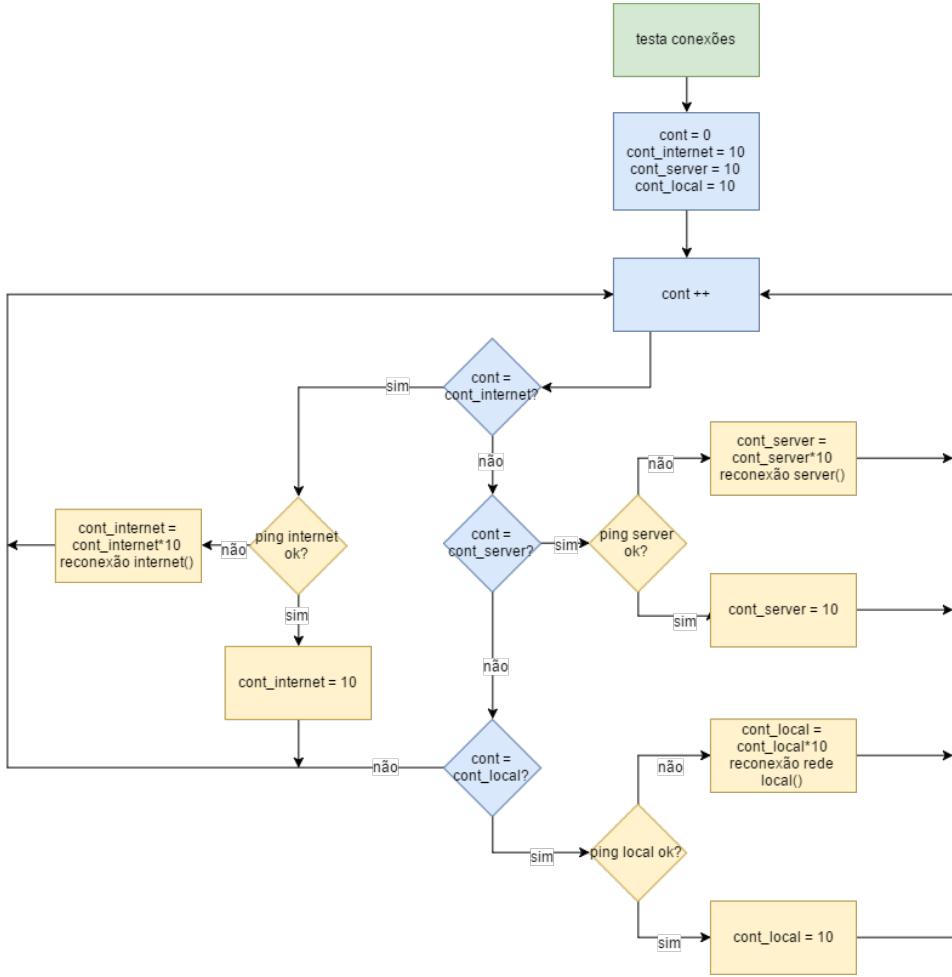
Figura 20: Rotina de multiplexação de procedimentos no tempo



5.1.1.2 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 21: 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 a capacidade de processamento é baixa.

5.1.1.3 DoS Local (*Evil Twin*)

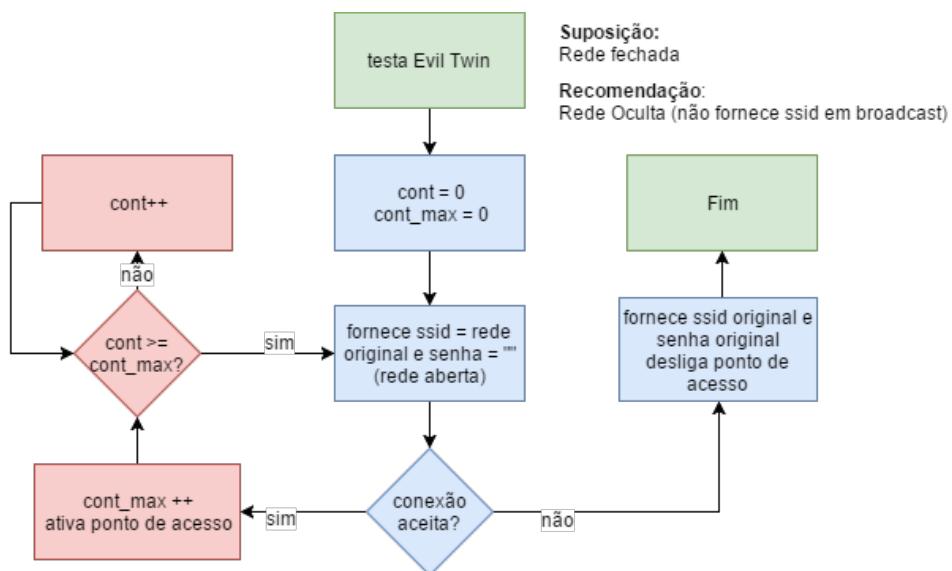
No caso de ataque de *Evil Twin* — no qual uma rede mal-intencionada, usualmente aberta, usa o mesmo SSID da rede original com o objetivo de obter a senha — o sistema pode ficar indisponível até ao nível local. Módulos podem se conectar à rede

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 radiofrequênci ou outro mecanismo utilizado pelo usuário mal-intencionado para que os clientes se desconectem, tentem reconexão e forneçam a senha da rede.

Além do problema de indisponibilidade, no caso do dispositivo ser capturado pela rede mal-intencionada e ser controlado por usuários não legítimos, há a possibilidade dos módulos serem usados para um ataque de DDoS (SECURITY AFFAIRS, 2017).

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

Figura 22: Tratamento de ataque de DoS Local



5.1.1.4 Comunicação por MQTT

Para o desenvolvimento da comunicação por MQTT com o Morpheus no módulo, foi usada a biblioteca PubSub (O'LEARY, 2017) para Arduino.

Para cada módulo existem identificador e senha, que são parâmetros próprios que vêm de fábrica, além de configurações de porta e endereço do Morpheus local padrões.

O controle de comunicação foi realizado da seguinte forma:

1. Só há tentativa de conexão MQTT em caso de haver conexão do Wi-Fi e o horário interno estar configurado — caso contrário, pode-se provocar travamento do módulo ou envio de mensagens sem *timestamp*.
2. É executada configuração de *callback* e de servidor MQTT a cada reconexão – este ponto foi crítico para o bom funcionamento da comunicação.

3. No *callback* ocorre recebimento de mensagens e tratamento (*parsing* da mensagem recebida, o que é feito por meio 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 um 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 e a obtenção dos parâmetros de interesse, ocorre a 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 integrando suas funcionalidades.
6. Ainda em fase final, após a persistência de variáveis na EEPROM, ocorre a inclusão na fila de saída de mensagens de confirmação para o servidor local.
7. A fila de saída é usada para o envio periódico das mensagens de estado.

Figura 23: 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 (O’LEARY, 2017) 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 ou compactação em conjunto com o protocolo desenvolvido.

5.1.2 Diagrama PCB

Abaixo está o diagrama do circuito impresso (PCB).

Figura 24: 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*

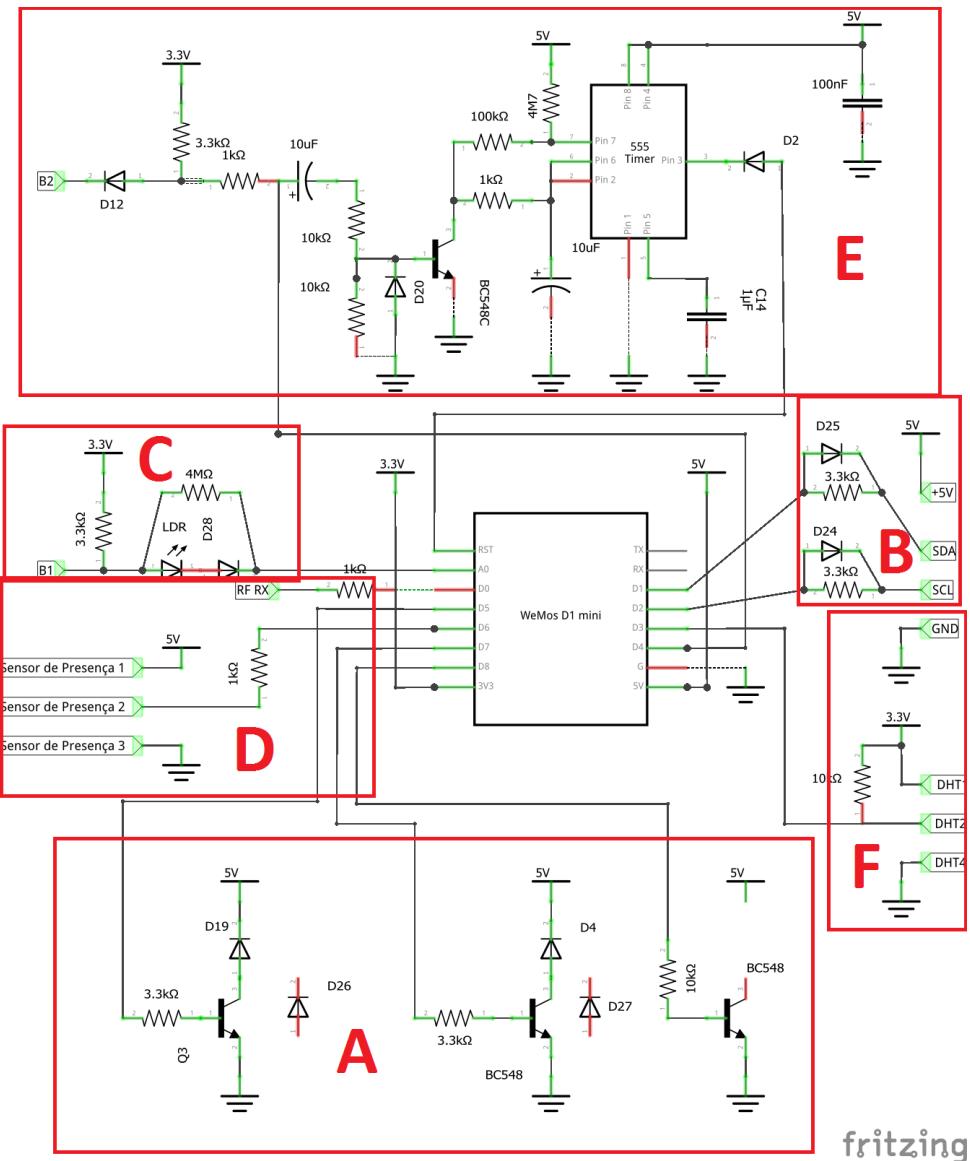
8. Botões

9. Presença

10. RF-RX

11. RF-TX

Figura 25: Diagrama elétrico do Módulo Base

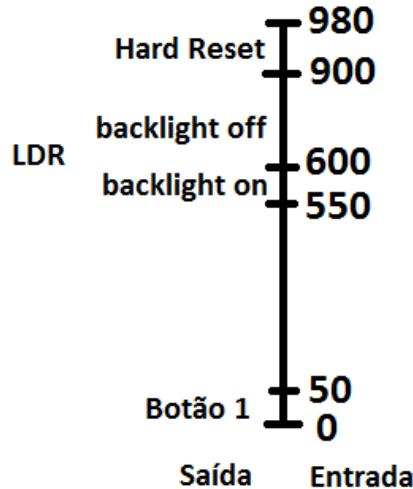


A - Saídas Circuitos simples de transistor para acionamento de relés (para lâmpadas) e *buzzer*.

B - 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.

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

Figura 26: Entradas em A0



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

E - 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, essa saída está conectada 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, permanências 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*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 27: Funcionamento do circuito antitravamento



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

5.1.3 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. Tais versões, por se tratarem de protótipos, não têm sua montagem completamente documentada, tampouco a uniformidade apresentada nos 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 28: Evolução do hardware (de fevereiro/17 a setembro/17)

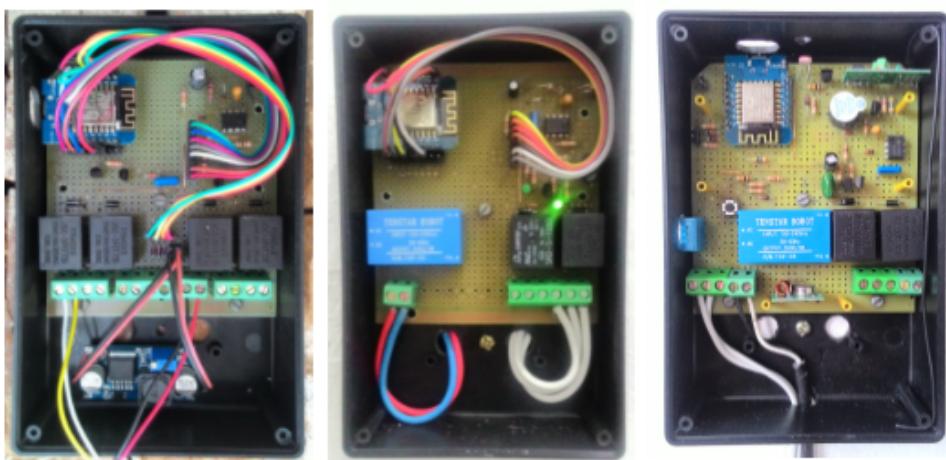
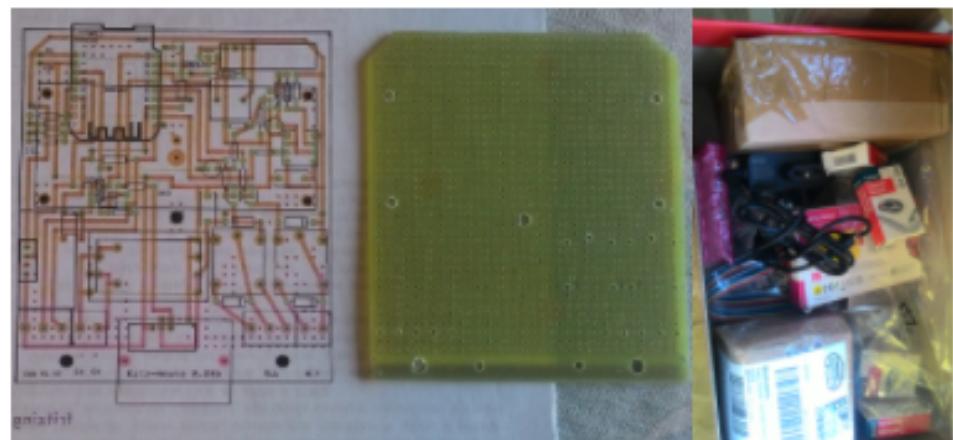


Figura 29: Evolução da caixa de proteção e display

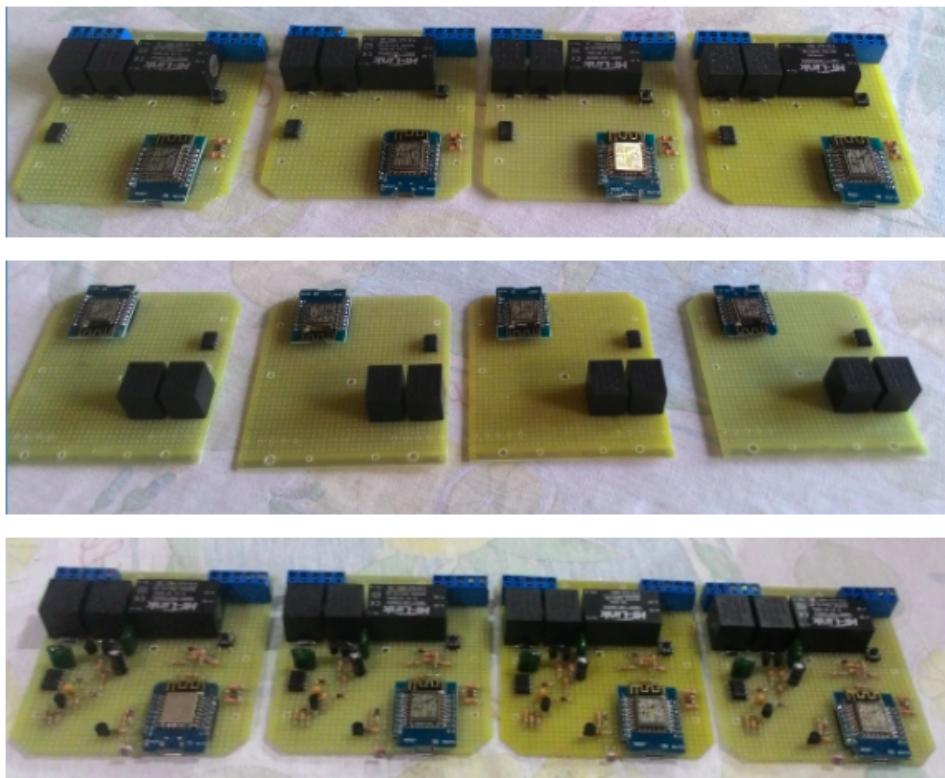


Figura 30: Materiais e preparação da placa padrão



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

Figura 31: Posicionamento dos componentes



2. Em seguida, foram posicionados os componentes.

Figura 32: Preparação dos displays com o I2C e soldagem



3. A próxima etapa foi soldar as trilhas por baixo conforme o diagrama PCB (Figura 24). Também foi necessário soldar o I2C com o display. Essa é a etapa mais demorada, que poderia ser facilmente operacionalizada ao serem adotadas placas de circuito impresso, o que aumentaria muito a capacidade de montagem.

Figura 33: 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 34: 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. Restou 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 de fazer as ligações dos botões, cabos de força e fios dos relés.

Figura 35: Quatro módulos prontos



5.1.3.1 Procedimento de Validação do Módulo

1. Carregar programa para testar módulo;
2. Realizar setup da conexão com a rede Wi-Fi;
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 bipes;
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 serve 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. Verificar o funcionamento dos botões, acionando-os um por um;
11. Checar o acionamento dos relés pelo aplicativo web também;

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

5.2 Controlador Local - Morpheus

5.2.1 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, cujo Deus dos sonhos, Morpheus, era responsável pelo envio de mensagens entre dois mundos diferentes, o dos deuses e o dos mortais (DHWTY, 2014). A principal atribuição do servidor local 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.

5.2.2 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.

Para se comunicar com os módulos, o Morpheus utiliza-se da conexão com um *broker* MQTT. O *broker* Mosquitto foi utilizado por ser uma solução *open-source* largamente utilizada em projetos de IoT. Conforme detalhado a frente, configurações de segurança específicas para o 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, relativa ao requisito não-funcional RNF-6, conforme documentada na Subseção 4.5.4.

5.2.3 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 implementadas em linguagens complementamente diferentes. Os sistemas de nuvem, por exemplo, foram implementados em Node.js. O aplicativo web também em foi feito em JavaScript, com utilização da biblioteca React. Os módulos de hardware foram programados em linguagem *C-like*, própria para Arduino, e o controlador local em Java. Essa flexibilidade permitiu a utilização de recursos e tecnologias que fossem melhor integrados com os requisitos propostos.

Um requisito essencial para o controlador local é a sua robustez (RNF-4). Em um cenário em que este controlador não esteja disponível, a casa passa a funcionar em estado de emergência, no qual 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. Por exemplo, a falta de energia ou de Internet na residência interrompe o seu funcionamento, não sendo possível ter controle sobre tal situação. O mesmo ocorre no evento de problemas de hardware na plataforma que o sistema estiver rodando. Os planos para contenção dos seus efeitos de tais situações são complexos, custosos e fogem do escopo deste projeto, como seria o caso de 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 assincronamente, conforme o requisito não-funcional RNF-10. Parte desta tarefa é facilitada com a utilização do sistema de mensageria MQTT, operado pelo *broker* Mosquitto. Com sua utilização, mensagens podem ser enviadas mesmo que o controlador não consiga recebê-las, pois 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 processar a próxima, de modo que o 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. É uma linguagem utilizada em vasta gama de aplicações, desde complexos softwares comerciais como a IDE Eclipse, até softwares embutidos, como controladores de BlueRay (HOPKINS, 2008). 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.

O *framework* Spring Boot¹ foi utilizado para o desenvolvimento por oferecer diversos recursos facilitadores, configurações de ambiente e um *container* para inversão de controle (*IoC - Inversion of Control*) e injeção de dependência. Essa técnica diminui o acoplamento entre classes e permite a evolução e implementação de novas funcionalidades de maneira fácil (FOWLER, 2004). 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.

Além disso, o gerenciador de dependências Gradle² também foi utilizado por oferecer um poderoso ambiente para configurar, construir e distribuir aplicações. Gradle faz uso de Groovy³, 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. Contudo, frente aos benefícios, ainda é vantajosa a utilização de Java neste caso.

Internamente, o Morpheus é dividido em pacotes, que são responsáveis pela modelagem 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 nas mensagens vindas da nuvem, há o envio de relatório com os problemas encontrados. A partir do reconheci-

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

²<https://gradle.org/>

³<http://groovy-lang.org/>

Figura 36: Arquitetura do servidor local



mento, a mensagem é colocada em uma fila interna, onde outro módulo será responsável por capturá-la e realizar o processamento necessário.

5.2.4 Características e Recursos

Para a concepção do servidor local, foram considerados os requisitos funcionais e não-funcionais, discutidos na Seção 3.2. As características e recursos da implementação são discutidos em seguida.

Configuração dos módulos físicos

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

Envio de dados para a nuvem

Dados provenientes de sensores são enviados para a nuvem, para que possam ser tratados de acordo com as regras de *Business Intelligence* e utilizados em algoritmos de aprendizado de máquina.

Persistência de dados

Quando não houver conexão, o servidor armazena os dados localmente e, quando solicitado, os envia à nuvem.

Tentativas de reenvio

Quando uma mensagem não é enviada com sucesso à nuvem, o Morpheus tenta novamente por um número configurável de vezes em um curto espaço de tempo. Isso ocorre porque, se determinada mensagem não pode ser enviada em uma janela temporal, ela perde o seu sentido (e.g. requisição de abertura de portão).

Verificação do *timestamp*

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

Tomadas de ação

Quando o usuário requisitar uma tomada de ação, esta deve ser enviada por meio de uma mensagem ao Morpheus, por onde será transmitida ao módulo.

Configuração em arquivo

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

Listeners para diferentes tipos de mensagens

Devem haver *listeners* para todos os tipos de mensagens que são recebidos da nuvem e dos módulos.

Processamento concorrente

Toda a infraestrutura do Morpheus permite o processamento concorrente de mensagens. Não é necessário 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 são encriptados na camada de transporte por meio de *TLS*.

Conversão de mensagens

As mensagens enviadas à nuvem são codificadas em JSON após serem convertidas do formato interno, que se refere apenas à troca de mensagens entre os módulos e o Morpheus.

Serialização das configurações

O servidor serializa e persiste as configurações relativas aos módulos que foram configurados para carregá-las em sua inicialização.

Destrução de pools de threads

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

5.2.5 Protocolo para Troca de Mensagens com Módulos

Toda a comunicação entre as partes do projeto é realizada por troca de mensagens. Foi desenvolvido um protocolo específico, leve e expansível, para a codificação dessas mensagens. As subseções seguintes definem e exemplificam o uso do protocolo.

5.2.5.1 Tópicos

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

1. Somente o Morpheus consegue publicar em qualquer tópico ou ser um *subscriber* de qualquer tópico.
2. Cada módulo somente consegue publicar no tópico determinado para ele, o que é 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.

Têm-se 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.

5.2.5.2 Regras de Negócio

O servidor foi desenvolvido com base nas regras de negócio seguintes. Cada regra de negócio tem sua identificação dada por mRN , seguida de um identificador numérico — e.g. $mRN-1$.

mRN-1: Após a compra de cada módulo, o usuário deve registrar online a aquisição. O servidor da nuvem enviará para o servidor local da casa a requisição para configurar o módulo.

mRN-2: Cada módulo envia mensagens para o servidor local com o seus dados por meio do MQTT.

mRN-3: Para a troca de senha do Wi-Fi, o usuário cadastra no aplicativo 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.

mRN-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, também havendo uma senha para se autenticar junto ao *broker* MQTT.

5.2.5.3 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`)
 - Requisição de dados (`data_request`)

5.2.5.4 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 00h00 de 1 de Janeiro de 1970, 64 bits>
@
```

Configuração de nome

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

Configuração de comunicação

```
#configuration
$ts: <timestamp>
$ty: communication_config
@
new_ssid: <novo ssid>
new_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>
```

@

Configuração de RF

```
#configuration
$ts: <timestamp>
$ty: rf_config
@
<nome do sensor | controle | funcao>: <"store" | "clear" | "keep">
@
```

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: Morpheus 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 Morpheus, que 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 do relé 1.

Rele1: 0 = desligar; 1 = ligar

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

Rele2: 0 = desligar; 1 = ligar

Requisição de reinício de software

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

0 = não; 1 = confirmar reinício

Requisição de teste de *auto reset*

```
#action_request
$ts: <timestamp>
$ty: autoreset_test
@
autoreset: <0 | 1>
@
```

0 = não; 1 = confirmar reinício

Confirmação (confirmation)

- Sentido: do módulo para o servidor.
- Uso: confirmação de uma configuração ou requisição de ação vindas do servidor.

Confirmação de hora

```
#confirmation
$ts: <timestamp>
$ty: time_confirm
@
ntp: <segundos desde 00h00 de 1 de Janeiro de 1970, 64 bits>
@
```

Confirmação de nome

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

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 configuração de RF

```
#confirmation
$ts: <timestamp>
$ty: rf_confirm
@
```

```
<nome do sensor | nome do controle | nome do funcao>: <valor_gravado>
@
```

Configuração de configuração de display

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

Confirmação de reinício de software

```
#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, relés, sensor de presença e luminosidade

```
#data_transmission
$ts: <timestamp>
$ty: temp_umi_pres
@
s1: umidade
v11: <value>
s2: temperatura
v12: <value>
s3: presenca
v13: <value>
s4: rl1
v14: <value>
s5: rl2
v15: <value>
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: <timestamp>
$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: <integer de 0 a 23>
final_time1: <integer de 0 a 23>
time_keepon1: <tempo em minutos>
time_deslmanual1: <tempo em minutos>
@
#configuration
$ts: <timestamp>
$ty: auto2_config
@
initial_time2: <integer de 0 a 23>
final_time2: <integer de 0 a 23>
time_keepon2: <tempo em minutos>
time_deslmanual2: <tempo em minutos>
@
```

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 configuração 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>
@
#confirmation
$ts: <timestamp>
$ty: auto2_confirm
@
initial_time2: <integer de 0 a 23>
final_time2: <integer de 0 a 23>
time_kepon2: <tempo em minutos>
time_deslmanual2: <tempo em minutos>
@
```

Confirmação de senha

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

Transmissão de estado de acesso

```
#data_transmission
$ts: <timestamp>
$ty: acesso_estado
@
s1: abertura
vl: <0 fechado | 1 aberto>
s2: alarme
vl: <0 desligado | 1 ligado>
s3: tempo_alarme
vl: <value>
@
```

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

Mensagens próprias para o Módulo do Quarto

Configuração de despertador

```
#configuration
$ts: <timestamp>
$ty: despertador_config
@
despertador: <0 | 1>
despertador_tempo: <tempo em minutos>
@
```

Configuração de luz automática

```
#configuration
$ts: <timestamp>
$ty: acesso_config
@
initial_time: <integer de 0 a 23>
final_time: <integer de 0 a 23>
time_keepon: <tempo em minutos>
time_deslmanual: <tempo em minutos>
@
```

Confirmação de despertador

```
#confirmation
$ts: <timestamp>
$ty: despertador_config
@
despertador: <0 | 1>
despertador_tempo: <tempo em minutos>
@
```

Confirmação de luz automática

```
$ts: <timestamp>
$ty: acesso_config
@
initial_time: <integer de 0 a 23>
final_time: <integer de 0 a 23>
time_kepon: <tempo em minutos>
time_deslmanual: <tempo em minutos>
```

Mensagens próprias para o Módulo do Externo

Configuração de alerta de 1 (água) e 2 (energia elétrica)

```
#configuration
$ts: <timestamp>
$ty: offset_config
@
alerta1: <0 | 1>
alerta1_nivel: <vl>
alerta2: <0 | 1>
alerta2_nivel: <vl>
@
```

Confirmação de configuração de alerta de 1 (água) e 2 (energia elétrica)

```
#confirmation
$ts: <timestamp>
$ty: offset_config
@
alerta1: <0 | 1>
alerta1_nivel: <vl>
```

```

alerta2: <0 | 1>
alerta2_nivel: <vl>
@
```

Transmissão de estado de Módulo Externo

```

#data_transmission
$ts: <timestamp>
$ty: externo_estado
@
s1: agua
vl: <integer>
s2: energia_eletrica
vl: <integer>
s1: agua_alerta
vl: <0 | 1>
s2: energia_eletrica_alerta
vl: <0 | 1>
@
```

Mensagens próprias para o Módulo da Cozinha

Programação para preparo (café ou arroz)

```

#configuration
$ts: <timestamp>
$ty: offset_config
@
initialtime: <vl>
finaltime: <vl>
cooking_time: <vl>
cook_mode: <"auto" | "manual">
@
```

Confirmação de programação para preparo (café ou arroz)

```

#confirmation
$ts: <timestamp>
$ty: offset_config
@
```

```

initialtime: <vl>
finaltime: <vl>
cooking_time: <vl>
cook_mode: <"auto" | "manual">
@
```

Transmissão de estado do Módulo da Cozinha

```

#data_transmission
$ts: <timestamp>
$ty: cozinha_estado
@
s1: gas
vl: <integer>
s2: cooking
vl: <0 | 1>
@
```

Requisição de offset para alarme de gás

```

#data_transmission
$ts: <timestamp>
$ty: gas_offset
@
<vl>
@
```

5.2.5.5 Testes Realizados da Comunicação Morpheus e Módulos

Para que fosse simulado o envio de mensagens, o aplicativo MQTT Fx⁴ foi utilizado. Com o uso deste software, é possível se inscrever em determinado tópico, enviar as credenciais para o *broker*, tanto em forma de usuário e senha, quanto em forma de certificados, além de publicar no tópico desejado.

Requisição de acionamento 1

```

#action_request
$ts:<timestamp>
```

⁴<http://mqttfx.jensd.de/>

```
$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 reinício de software

```
#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, presença e relés

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

Esperado: mensagem no tópico MQTT m2s.

Resultado: de acordo.

5.2.6 Protocolo para Troca de Mensagens com a Nuvem

A troca de mensagens entre Morpheus e servidor na nuvem ocorre por meio de WebSockets. Como explicado anteriormente, caso o servidor local provesse uma API REST aberta à conexões externas, seriam necessárias configurações, software e hardware avançados para a proteção, o que seria inviável para o propósito em questão. Além disso, usuários domésticos possuem endereços IP dinâmicos, de maneira que seria necessária a atualização desse endereço na nuvem a cada mudança. Com a utilização do WebSocket, o servidor local é um cliente que solicita a abertura de uma conexão com a nuvem, que é mantida aberta a partir daí.

A comunicação com a nuvem é baseada em eventos, que são recebidos e enviados com os dados relevantes. Os eventos estão descritos a seguir.

5.2.6.1 Morpheus

O Morpheus ouve os seguintes eventos vindos da nuvem:

- configuration
- action
- data (requisitar informações sobre módulo, e.g. se portão está aberto ou não).

5.2.6.2 Nuvem

A nuvem ouve os seguintes eventos vindos do Morpheus:

- confirmation
- configuration
- data

Definição de mensagens entre Nuvem e Morpheus

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

```

"morphheusConfiguration": <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 é 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>
  }
]
}

```

Requisição de ação As mensagens de `action` seguem o mesmo protocolo estabelecido anteriormente para mensagens de `action_request`.

Transmissão de dados As mensagens de `data` também seguem o mesmo protocolo estabelecido anteriormente para mensagens de `data_transmission`.

Exemplo de mensagem A seguir, um exemplo de uma mensagem completa de configuração.

```

{
  "configurationId": "1",
  "timestamp": "1499717103422",
  "morpheusConfiguration": {},
  "modulesConfiguration": [
    {
      "moduleId": "00765914",
      "moduleName": "teste_wemos",
      "moduleTopic": "hw/00765914",
      "unregister": true
    },
    {
      "moduleId": "000281D0",
      "moduleName": "hugo_basic",
      "moduleTopic": "hw/000281D0",
      "unregister": false,
      "messages": [
        {
          "id": "12345678901234567890123456789012"
        }
      ]
    }
  ]
}

```

```
{
    "controlParameters": [
        {
            "parameter": "ts",
            "value": 1500914158
        },
        {
            "parameter": "ty",
            "value": "rele1_action"
        }
    ],
    "payload": {
        "v1": 5,
        "v2": "auto"
    }
}
]
}
```

5.2.7 Configurações

5.2.7.1 Configuração do Mosquitto

Em situações reais, cada casa terá uma instância do *broker* Mosquitto rodando independente de todas as outras e aceitando somente conexões locais. Entretanto, para que fossem realizados testes e simulações, a instalação e execução de uma instância em cada máquina diferente, localmente, seria inviável. Para tanto, foram realizadas instalações em uma máquina remota — em servidor da Digital Ocean⁵ — com configurações diferentes, uma para cada residência simulada. São executadas instâncias como processos *daemon* vinculados a 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 (COPE, 2017b) (O'LEARY, 2017):

⁵<https://www.digitalocean.com/>

1. Habilitar a restrição de tópicos na instância (COPE, 2017a). A restrição deve levar em conta 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). Essas portas não exigem criptografia, mas devem exigir usuário e senha (que estarão vulneráveis).
5. O Morpheus é obrigado a se conectar a partir da porta 8883 (e.g. primeira casa → 8883; segunda casa → 8884), passando suas credenciais encriptadas.

5.2.7.2 Guia de Instalação

O passo-a-passo descrito aqui foi testado em um ambiente 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`. Devem-se criar 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 no Anexo C.

5.2.7.3 Criação dos Certificados

Por fim, devem-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 haver uma autoridade certificadora independente para garantia da validade e segurança.

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

```
openssl req -new -x509 -extensions v3\_ca -keyout ca.key -out ca.crt
```

2. Criação de chave e certificado para o Mosquitto. 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. Criação de chave e certificado para o Morpheus. O *common name* deve ser 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
```

5.2.7.4 Senhas

Conforme mostrado anteriormente no guia de instalação (item 5.2.7.2), o arquivo de senha deve ser criado no formato `usuario:senha`. Deve-se, então, rodar o seguinte comando para que a senha não fique exposta em formato de texto:

```
sudo mosquitto_passwd -U passwd
```

5.2.7.5 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 um tópico que não pertence ao seu usuário.

Esperado: não receber dados.

Resultado: bem-sucedido.

6. Conectar com credenciais corretas e tentar ouvir 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.

5.3 Servidor na Nuvem

5.3.1 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*.

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.

5.3.2 Características da Implementação

Com base nos requisitos funcionais e não-funcionais, discutidos na Subseção 3.2, foi realizada a implementação na nuvem. Suas características são discutidas em seguida.

Comunicação

- O servidor permite que Morpheus e aplicativos clientes se comuniquem via WebSocket.
- Morpheus pode 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 pode receber mensagens de registro e remoção de módulo para definir quais dispositivos ele gerencia.

- Morpheus pode 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 persiste dados de usuário, de configurações de Morpheus e módulo e de mensagens de dados (`data` e `report`).

Gerenciamento de Dados

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

Coneções

- O servidor permite o estabelecimento de conexões HTTPS seguras e criptografadas.
- O *firewall* bloqueia conexões em portas que não estão sendo utilizadas.

5.3.3 Tecnologias Utilizadas

O servidor foi desenvolvido usando Node.js⁶, 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 MongoDB⁷, 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.

⁶<https://nodejs.org>

⁷<https://www.mongodb.com/>

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 Redis⁸, 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.io⁹, 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*.

A arquitetura possui um proxy reverso que é responsável por enviar as requisições ao servidor em Node.js. Para isso, foi usado o nginx¹⁰, 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.

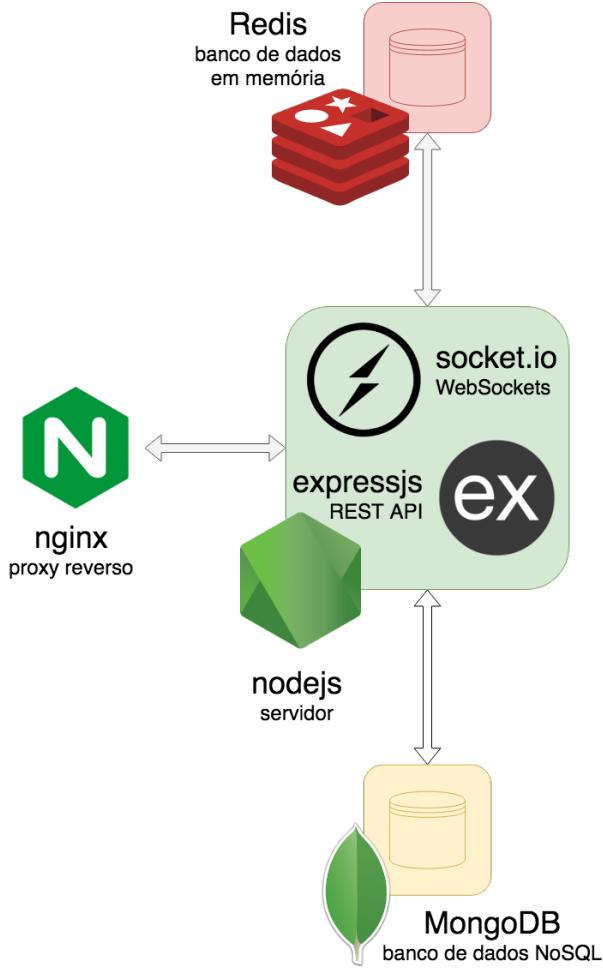
As tecnologias apresentadas aqui favorecem a escalabilidade do sistema. O Node.js possui um modo de execução *cluster*, no qual processos filhos ou *workers* são criados a partir de um processo pai ou *master* por meio de *forking*, sendo possível assim aproveitar todas as CPUs de uma máquina (NODE.JS, 2017). O MongoDB suporta técnicas de *sharding*, que é um método para distribuir dados em múltiplas máquinas para armazenar grandes quantidades de dados e garantir alta vazão de leituras e escritas (MONGODB, 2017). O Redis também pode ser escalado por meio de “clusterização”. Já o nginx possui uma arquitetura que consegue lidar com centenas de milhares de requisições sem bloquear sua máquina de estados, otimizando o número de processos para diminuir o tempo gasto em mudanças de contexto (NGINX, 2017).

⁸<https://redis.io/>

⁹<https://socket.io/>

¹⁰<https://nginx.org/>

Figura 37: Componentes e implementação na nuvem



5.3.4 Tratamento de Eventos

O servidor na nuvem possui uma arquitetura baseada em eventos, que transmitem dados, ações, configurações e confirmações e podem ser provenientes dos Morpheus ou dos aplicativos clientes conectados. O servidor também pode emitir seus próprios eventos para tais entidades conectadas ouvirem. Nessa seção, as rotinas de tratamento para cada tipo de evento são sucintamente descritas.

Conexão (connection)

- **Descrição:** um novo dispositivo se conecta à nuvem.
- **Emitido por:** Morpheus, aplicativos clientes.
- **Tratamento:** é criada uma *string* única de identificação para a nova conexão.

Identificação de Dispositivo Conectado (`hello`)

- **Descrição:** ao se conectar à nuvem, um novo dispositivo se identifica como Morpheus ou aplicativo cliente. Em ambos os casos, é enviado também a identificação serial do Morpheus — no caso dos aplicativos, seria o Morpheus do qual se deseja receber notificações em tempo real. Caso um aplicativo queira receber notificações de múltiplos Morpheus, devem ser enviados múltiplos eventos de `hello`.
- **Emitido por:** Morpheus, aplicativos clientes.
- **Tratamento:** os identificadores de conexão, os tipos de dispositivos e os identificadores seriais do Morpheus são salvos no Redis. Assim, é possível buscar qual é a conexão correspondente a cada dispositivo conectado. Além disso, se já existem dados salvos no Redis a respeito de aplicativos ou Morpheus que referenciam ao mesmo serial de Morpheus, são enviados a eles eventos de `hello`. Um caso de uso para tal funcionalidade é a possibilidade de aplicativos saberem se todos os controladores locais de uma residência estão online.

Ação (action)

- **Descrição:** é enviada uma requisição de ação a um módulo.
- **Emitido por:** aplicativos clientes.
- **Tratamento:** é verificado se o Morpheus responsável por repassar essa requisição ao módulo físico está conectado. Se sim, o evento de `action` é re-emitido somente a ele.

Configuração (configuration)

- **Descrição:** é enviada uma requisição de configuração a um módulo.
- **Emitido por:** aplicativos clientes.
- **Tratamento:** é verificado se o Morpheus responsável por repassar essa requisição ao módulo físico está conectado. Se sim, o evento de `configuration` é re-emitido somente a ele.

Confirmação (confirmation)

- **Descrição:** é enviada uma confirmação de ação ou configuração a um aplicativo.
- **Emitido por:** Morpheus.
- **Tratamento:** é verificado se existem aplicativos correspondentes a esse Morpheus conectados à nuvem. Se sim, o evento de `confirmation` é re-emitido a todos eles. Isso é uma simplificação que obriga os aplicativos a tratarem o recebimento desse tipo de evento por si mesmos.

Dados (data)

- **Descrição:** é enviada uma atualização do estado do módulo físico e seus sensores.
- **Emitido por:** Morpheus.
- **Tratamento:** é verificado se existem aplicativos correspondentes a esse Morpheus conectados à nuvem. Se sim, o evento de `data` é re-emitido a todos eles. Além disso, esses dados são persistidos no MongoDB.

Relatório (report)

- **Descrição:** é enviada um relatório de configurações do Morpheus.
- **Emitido por:** Morpheus.
- **Tratamento:** é verificado se existem aplicativos correspondentes a esse Morpheus conectados à nuvem. Se sim, o evento de `report` é re-emitido a todos eles. Além disso, esses dados são persistidos no MongoDB.

Desconexão (disconnect)

- **Descrição:** um dispositivo se desconecta da nuvem.
- **Emitido por:** Morpheus, aplicativos clientes.
- **Tratamento:** as entradas no Redis relacionada a essa conexão são deletadas. Se o dispositivo que se desconectou foi um Morpheus, é emitido um evento de `bye` a todos os aplicativos conectados que estavam ouvindo atualizações desse Morpheus. Dessa forma, podem ser implementadas mensagens de erro e avisos ao usuário.

5.3.5 Infraestrutura

Para hospedar o servidor do Hedwig, foi utilizado o serviço de computação na nuvem Digital Ocean¹¹. Com ele, foi possível implantar o servidor em uma instância que roda Ubuntu 16.04.3 x64, com única CPU, 512MB de memória, 20GB de armazenamento de disco SSD e 1000GB de cota disponível para transferência de dados. O data center que hospeda essa instância fica em Nova Iorque.

5.3.6 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 Encrypt¹², 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 o *firewall* ufw para bloquear conexões nas portas TCP que não estão sendo usadas.

5.3.7 Operação

O serviço Keymetrics¹³ 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.

¹¹<https://www.digitalocean.com/>

¹²<https://letsencrypt.org>

¹³<https://keymetrics.io/>

Figura 38: Monitoramento do servidor na nuvem



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.

5.4 Aplicativo de *dashboard*

5.4.1 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 4.6, relativo a arquitetura a fim de permitir uma boa experiência de usuário tanto em smartphones como em computadores.

5.4.2 Requisitos

5.4.2.1 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 relés 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 relés 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 sensor e controle 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.

5.4.2.2 Requisitos Não-funcionais

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

5.4.3 Tecnologias Utilizadas

Para criar a aplicação web que demonstra a funcionamento do Hedwig, foi escolhido o React¹⁴, 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 Native¹⁵ — 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 aplicativos nativos.

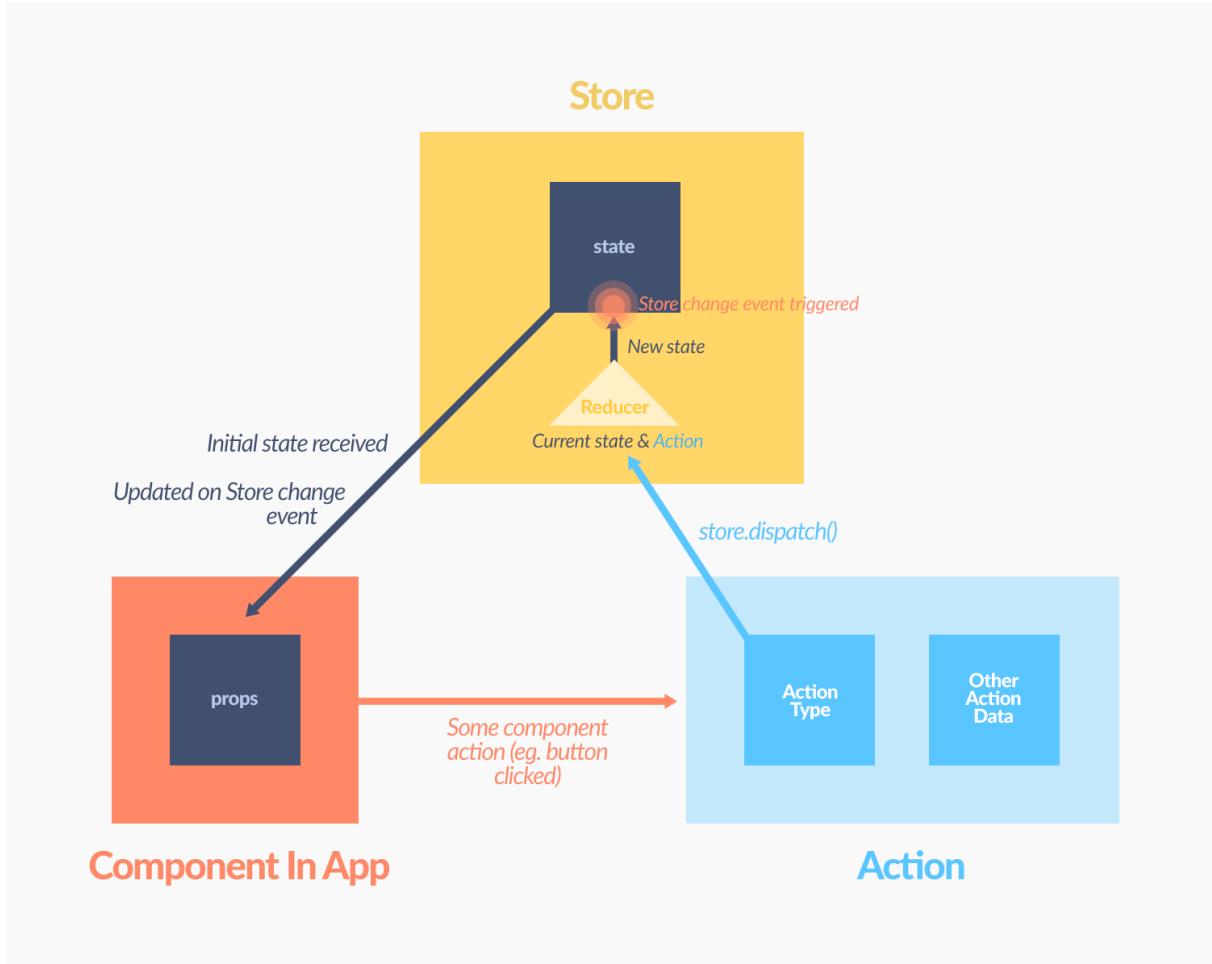
A fim de facilitar a implementação das interações com o usuário, é usada juntamente ao React a biblioteca Redux¹⁶, um gerenciador de estado global. Dessa forma, o compartilhamento de informações entre diferentes componentes fica simplificado. A motivação

¹⁴<https://reactjs.org/>

¹⁵<https://facebook.github.io/react-native/>

¹⁶<https://redux.js.org/>

Figura 39: Diagrama de funcionamento do Redux



Fonte: (FACEBOOK, 2016)

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.
- 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 foram criados para os protocolos do Hedwig.

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, é usado Babel¹⁷, 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, é tirado proveito do *linter* dedicado a JavaScript ESLint¹⁸. *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 Webpack¹⁹ 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.

5.4.4 Interface

5.4.4.1 Identidade Visual

Para facilitar o desenvolvimento, foram usados os componentes do Material Design²⁰ 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

¹⁷<https://babeljs.io/>

¹⁸<https://eslint.org/>

¹⁹<https://webpack.js.org>

²⁰<https://material.io/>

cores distintas.

5.4.4.2 Telas

A seguir, estão as telas principais do aplicativo mostrando os dados e as opções de ação e configuração de um Módulo de Acesso e de um Módulo Básico.

Figura 40: Tela correspondente ao Módulo de Acesso

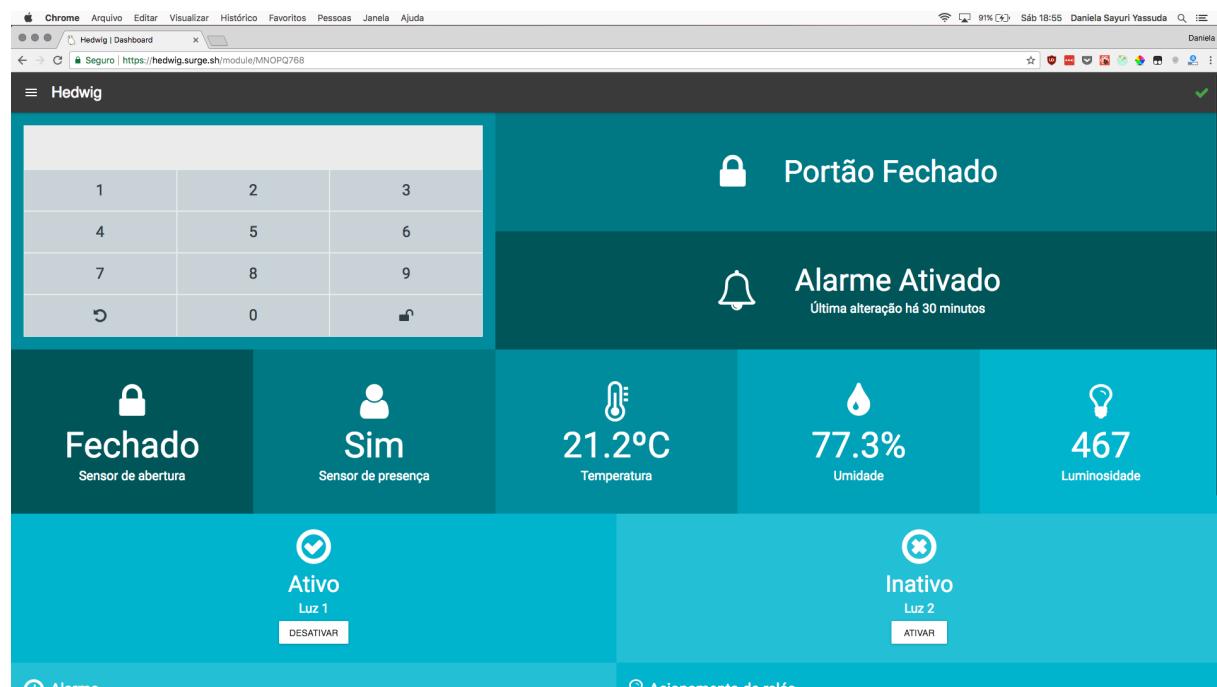


Figura 41: Tela correspondente ao Módulo Básico



Demais telas do aplicativo mostrando tanto a versão desktop como a móvel podem ser encontradas no Anexo E.

5.4.5 Interações

5.4.5.1 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 pela *dashboard* a fim de evitar que o usuário olhe dados antigos demais.

5.4.5.2 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 relé está ligado e o usuário pede para desligá-lo, o aplicativo só vai mostrar que o relé desligou quando o módulo o avisa disso. Isso evita que o aplicativo entre em estados inconsistentes com os módulos.

5.4.5.3 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.

5.4.5.4 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. Diminuir 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.

5.4.6 Implantação

O aplicativo foi publicado com o serviço Surge²¹, 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).

5.4.7 Segurança

O Surge usa a comunicação via HTTPS por padrão, oferecendo o suporte básico a TLS. Dessa forma, os dados transmitidos entre navegador e servidor podem ser criptografados,

²¹<https://surge.sh/>

permitindo uma maior segurança em operações como cadastro, login e recebimento dos dados dos sensores.

5.4.8 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.
- Não há redirecionamentos desnecessários.

5.5 Aplicativo 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).

5.5.1 Requisitos

Destacam-se os requisitos mais críticos do aplicativo backup:

1. Permitir acesso direto ao módulo em caso de indisponibilidade da rede Wi-Fi e Internet;
2. Permitir a visualização de estado de sensores e a atuação em tempo próximo ao apresentado por botão físico;
3. Acesso ao módulo por meio de endereço local no caso de haver rede local disponível, mas sem acesso à Internet;
4. Possibilitar ao usuário configurar rede Wi-Fi, nome do módulo, cor do painel, offset de sensores, nome dos relés e regras de atuação (por radiofrequência, com senha ou não, por horário, por eventos dos sensores de presença e abertura, além do tempo em que deve ficar ligado no caso de configurações automáticas);
5. Permitir configurar múltiplos códigos RF (radiofrequência) 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.

5.5.2 Tecnologias Utilizadas

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. HTML5 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 e I2C, por exemplo), comunicação em geral, além do Wi-Fi e *Access Point* (PubSub para comunicação por MQTT).

5.5.3 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 tela da Figura 42 é própria para celulares, enquanto a tela da Figura 43, com posicionamento configurável e que simula a planta de uma casa (em um cenário real, poderia ser a própria planta da casa do usuário) é própria para desktops.

Figura 42: Telas principais do Aplicativo Backup



Figura 43: Visão geral da planta de uma casa com o Aplicativo Backup



A partir do menu principal, pode-se acessar o menu do módulo desejado (vide Figura 44). 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 recepção de Wi-Fi do módulo e a versão do *firmware*.

Figura 44: Menu principal do Aplicativo Backup

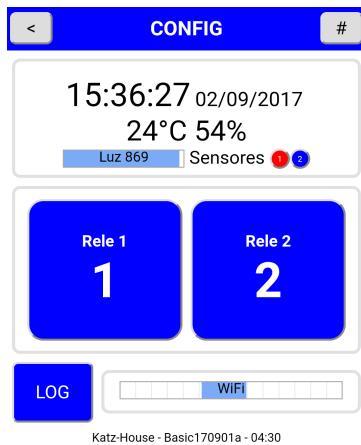


Figura 45: Teclado para digitação da senha



No caso de proteção de controle por senha, é exibido o painel numérico como na Figura 45. A cada requisição da página, o módulo manda um mapeamento das teclas diferente. Por exemplo, as teclas A e B mapeiam para (3,1) e (9,2). Após o usuário entrar com a senha (sequência de teclas do tipo A, B, C, etc.), 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.

5.5.4 Configurações

A partir no menu principal do módulo, pode-se ter acesso ao seu log para depuração, uma opção disponível 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, podem-se configurar alertas e alarmes (sonoros a partir dos módulos e pela Internet através do provedor gratuito Blynk e email), 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 uma opção) e o log (quais parâmetros são persistidos e mandados para a nuvem). Também pode-se acessar o menu de Ferramentas, no qual 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 o arquivo que contém a nova versão, para que seja feito *upload* do mesmo, e o módulo seja atualizado.

É possível também acessar as configurações avançadas (vide Figura 83). Nela, pode-se configurar um offset para temperatura e umidade, de preferência realizados a partir de um medidor confiável para calibração.

Do menu de configurações avançadas, podem ser configurados os sensores e controladores de radiofrequência (sem fio), aspectos de conectividade do servidor gratuito Blynk (usado no envio de emails) e possivelmente trocar a rede Wi-Fi 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 Wi-Fi.

Demais ilustrações referentes às configurações do aplicativo backup estão disponíveis no Anexo F.

5.5.5 Abertura de Porta do Roteador

Para acessar o módulo e o menu remotamente, uma solução é usar a abertura de porta (mecanismo NAT ou *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. Assim, essa alternativa provê a abertura da porta para acesso remoto sem a necessidade de serviços em nuvem, o que é indicado apenas para usuários que podem lidar com um nível de segurança mais baixo.

Figura 46: Página de configuração TP Link

Adicionar ou modificar uma entrada de servidor virtual

Porta de serviço:	8030 (XX-XX ou XX)
Porta interna:	80 (XX, insira um número de porta específico ou deixe em branco)
Endereço IP:	192.168.0.30
Protocolo:	TCP
Ativar:	Ativado
Porta de serviço comum:	--Selecione--

Salvar **Anterior**

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

5.5.6 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. É permitido 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, há um problema.
5. (a) A partir da página inicial do módulo, entre em # → Configurações Avançadas → RF433.

- (b) Aperte + (ao lado do relé 1 ou relé 2), espere até a página indicar “Aguardando” e abra o sensor de abertura. É permitido 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.

5.5.7 Notificações

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

1. Para que seja possível o envio de emails de alerta, acesse # → Configurações Avançadas.→ Blynk.
2. Insira o *Auth Token* (e.g. aa7a6dc1170640f08e951ed8cd2198a1).
3. Selecione Notificações ao iniciar: Sim, e Wi-Fi: Sim.
4. Aperte em OK, no canto superior direito, para salvar suas alterações.

5.5.8 Offset de Temperatura e Umidade

1. Para calibração do módulo, 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.

5.5.9 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 bipes;
5. O módulo retornará para a configuração de fábrica. Veja as seções a seguir para sua configuração inicial.

5.5.10 Setup Inicial

1. Primeiro, conecte à rede Wi-Fi do Módulo, com nome CONFIG (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 rede de sua casa 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-se 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 Wi-Fi 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 < no canto superior esquerdo). Você verá o menu principal da casa, e então poderá controlar relés, visualizar dados coletados pelos módulos e acessar cada módulo. O menu é personalizado na página anterior (ao pressionar o # no canto superior direito da tela).

6 Arquitetura: Tecnologia - Coleta e Análise de Dados

Um dos processos mais críticos para o sucesso de um projeto é a obtenção de resultados e dados, dos quais se obtém conhecimento sobre o comportamento do sistema em atividade e dos usuários que o utilizam.

6.1 Coleta

Para a coleta de dados, foram usados um total de oito 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 e o módulo 8 está localizado na cidade de Jarinu - SP. Já os módulos 9, 10 e 11 estão em casas diferentes na Grande São Paulo, e foram usados para desenvolvimento integrado de *dashboard* e Aplicativo Backup.

Figura 47: Log na página do Aplicativo Backup



Timestamp	Action
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 48: 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 com coleta local.

Figura 49: Módulos instalados em Santo André



O módulo de Jarinu possui uma interface com o sistema de alarmes já instalado no local e tem como objetivo obter dados de presença e abertura de portas, sendo uma prova de conceito para validar possível integração futura com parceiros estratégicos (e.g. Empresas de Segurança Residencial e Empresarial).

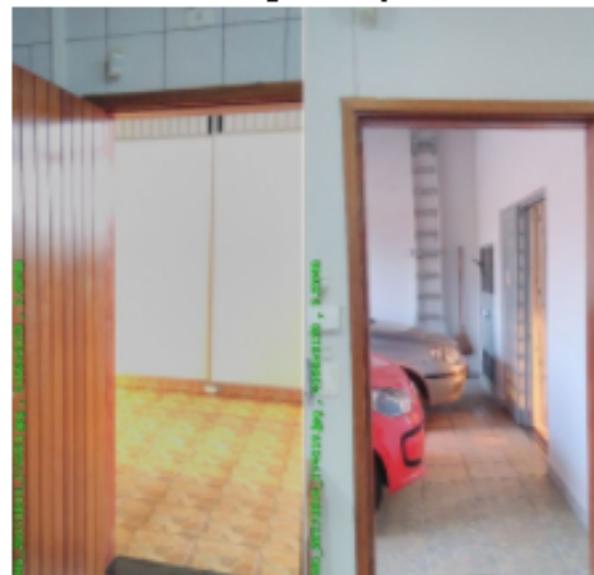
Figura 50: Controle do Sistema de Alarme e Módulo de Interface de Jarinu



Figura 51: Módulo Básico e sensor de presença no corredor de Jarinu



Figura 52: Sensor de abertura das portas da cozinha (à esquerda) e da sala (à direita)



Os demais módulos usados para integração possuem coleta de dados a partir de persistência na nuvem, com as mensagens passando pelo controlador local Morpheus.

Figura 53: 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, há a coleta dos seguintes dados:

Figura 54: 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 os conteúdos 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*”), pois em casos de pouca memória disponível o espaço de dados invade o espaço de programa e ocorre travamento do ESP; monitoramento do contador de loops de 1 segundo que, devido à espera ou demasiado processamento, demoraram mais que 2 segundos para

ocorrerem; monitoramento de desconexões, reconexões, atualizações do *firmware* dos módulos e reinícios;

- **Luminosidade:** monitoramento a partir de dado analógico de um LDR.
- **Temperatura e umidade:** sensor DHT.
- **Presença:** sensor de movimento PIR, que foi usado para monitorar a atividade de presença.
- **Uso dos relés:** 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 e período de coleta, não havia ainda a *dashboard* disponível.
- **Abertura 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, que representa alguma das outras duas pessoas da casa), uso do sistema (aberturas pelo sensor versus 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.

6.2 Análise

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 Wi-Fi)	15	26	24	27	77	

O aspecto de disponibilidade foi analisado a partir da média da memória disponível — quanto maior, melhor —, o número de loops de 1s que ocorreram em 2s ou mais — o que indica a existência de procedimentos que estão atrasando a correta execução de rotinas

periódicas, como aquelas de “*keep alive*” de comunicação —, e o número de reinícios e desconexões da rede Wi-Fi desconsiderando aqueles causados por atualizações 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, um menor número de desconexões da rede Wi-Fi 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 Wi-Fi, 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 relés

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%

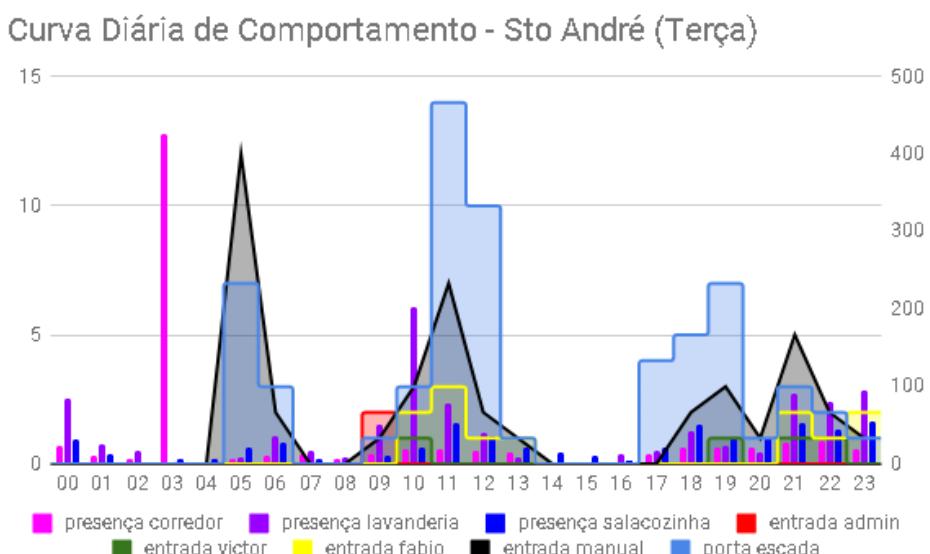
Quanto ao uso dos acionamentos possíveis (botão físico presente no módulo, pelo aplicativo backup — Web, por controle de radiofrequência — RF e por regra automática configurada pelo usuário - Auto), número de acionamentos e porcentagem do tempo ligado, observa-se que:

- As áreas comuns têm 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 em relação ao botão físico para a lâmpada

da cozinha e números de acionamentos por RF e botão físico em valores próximos para a Sala;

- A escada tem 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, cerca de metade dos acionamentos ainda ocorre 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 referindo-se a 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 55: Curva diária Santo André – terça-feira



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

- Há um pico de presença no corredor às 3h. Seria o horário em que os gatos da residência estão mais ativos;
- A curva de presença indica maiores movimentos entre 9h e 13h e entre 17h e 1h, 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 4h às 7h, Nair/Sabrina sai da casa;
 - Das 9h às 13h, Fabio sai de casa, algumas vezes usando seu usuário, outras vezes manualmente;
 - Às 10h, Victor sai de casa;
 - Das 17h às 19h, Sabrina/Nair voltam para casa;
 - Entre 19h e 22h, Victor volta para casa;
 - Entre 21h e 23h, Fabio volta para casa;
- Os picos de porta escada indicam os horários de maior entrada e saída da residência.

Figura 56: Temperatura e aquecedor – Módulo do Aquário



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

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

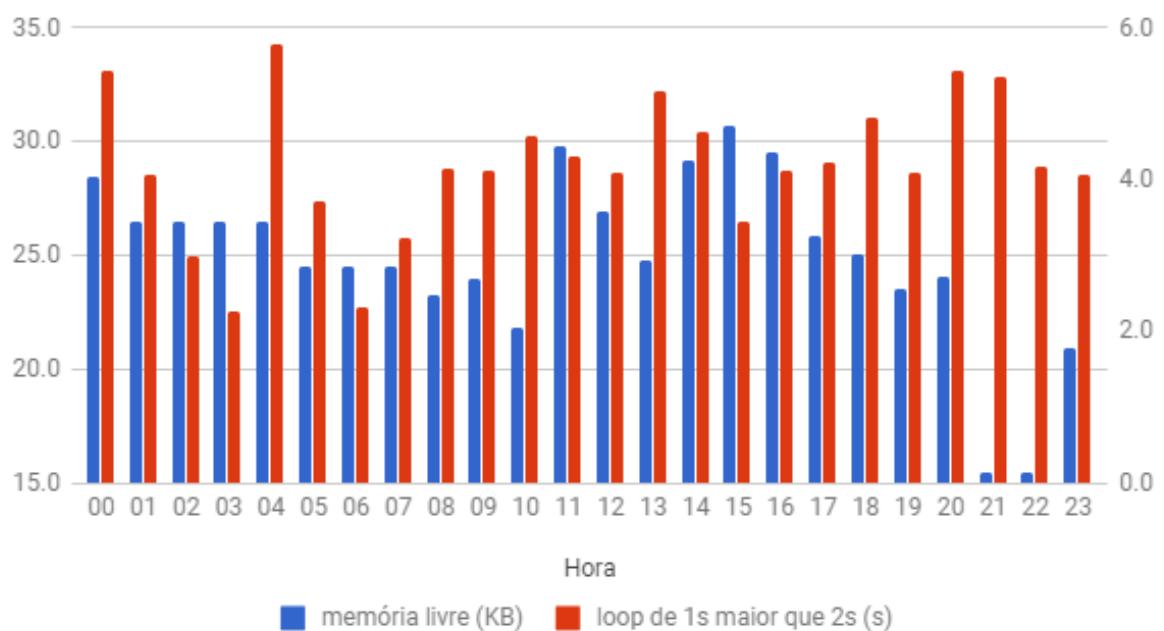
Figura 57: Consolidado Diário – Módulo do Aquário

Consolidado Diário



Figura 58: Memória e loops – Módulo do Corredor

Memória livre (KB) x Loops de 1s maiores que 2s (s)



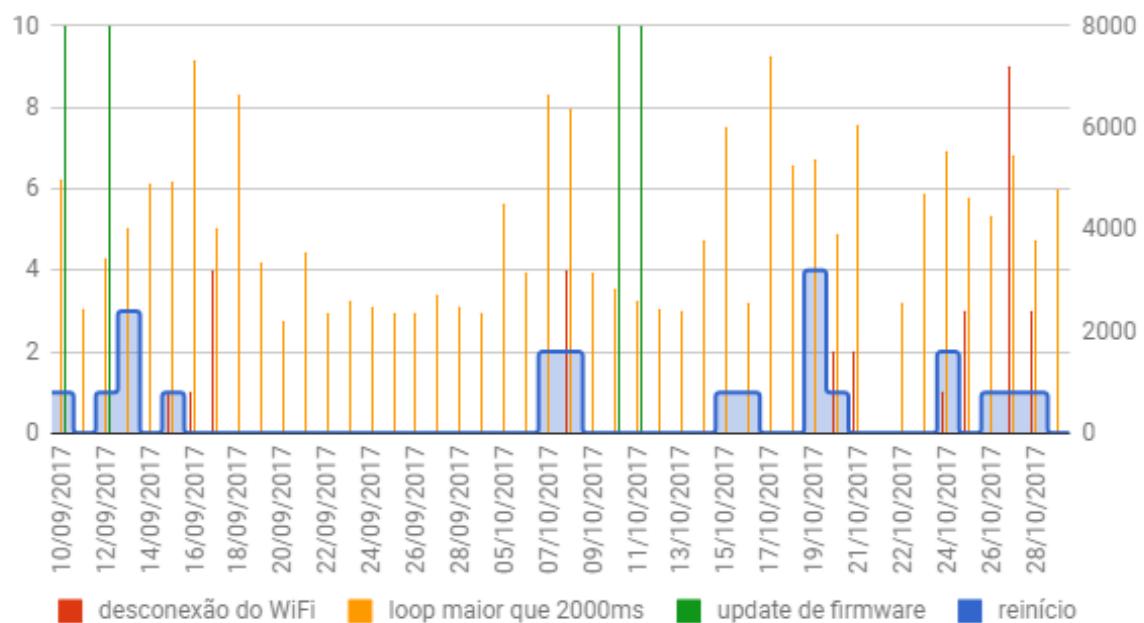
Observando sua curva diária de memória livre e loops de 1s que foram executados em

mais de 2s, observou-se que:

- Os períodos de maior disponibilidade são das 11h às 16h, pois há maior memória livre, que é o parâmetro mais crítico para possíveis reinícios e travamentos;
- Os períodos de menor disponibilidade são das 21h às 22h, nos quais houve picos de loops demorados e pouquíssima memória livre.

Figura 59: Consolidado no período – Módulo do Corredor

Consolidado no Período



São observados em verde, no gráfico acima, as atualizações de *firmware* — o que possibilita acompanhar a melhoria de disponibilidade após a implantação de novas versões — a quantidade de reinícios e ocorrências de desconexão do Wi-Fi. Entre 24/10 e 28/10, ocorreu grande número de desconexões do Wi-Fi, indicando indisponibilidade da rede nesses dias, que causaram uma quantidade considerável de reinícios no mesmo período.

Figura 60: Consolidado diário dos sensores – Módulo do Corredor



No gráfico acima, observa-se inatividade às 4h da manhã e entre as 14h e 16h. A temperatura possui pouca variação, assim como a umidade. Já a curva de luminosidade acompanha a de presença. Apenas das 14h às 16h, considere que há luz natural, por isso não é um valor próximo àquele apresentado de madrugada, entre 0h e 4h.

Figura 61: Uso dos acionamentos no período para a lâmpada – Módulo do Corredor

Uso dos tipos de acionamento para o Rele 1 - Período



No primeiro gráfico, observa-se a evolução do uso dos acionamentos para o relé 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, durante o dia, os acionamentos automáticos ocorrem entre 17h e 22h. Possivelmente, podem-se aplicar regras no período das 23h às 2h.

Figura 62: Uso dos acionamentos por dia para a lâmpada – Módulo do Corredor

Uso dos tipos de acionamento para o Rele 1 - Dia

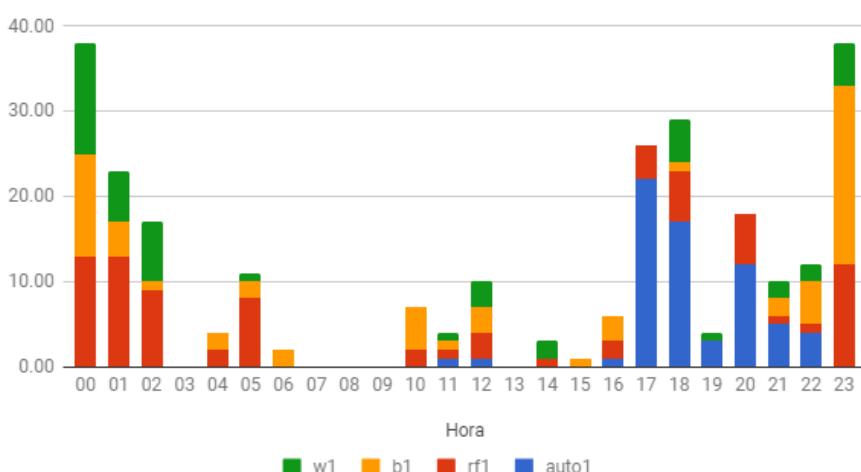


Figura 63: Curva diária – Módulo de Acesso



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

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

Figura 64: 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 9h às 12h e entre 16h e 23h;
- Os horários em que provavelmente há menor disponibilidade de água da rua para preencher a caixa e fazer o nível voltar a ser alto são das 5h às 6h da manhã e às 11h 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 65: Tempo para encher a caixa d'água – período

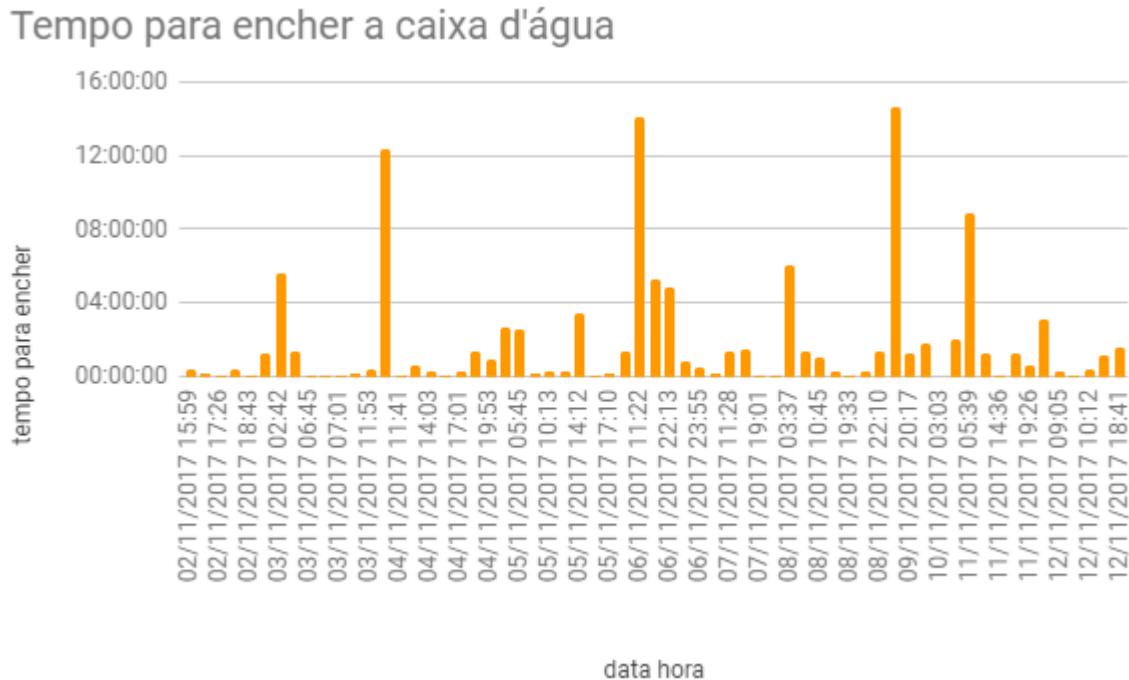
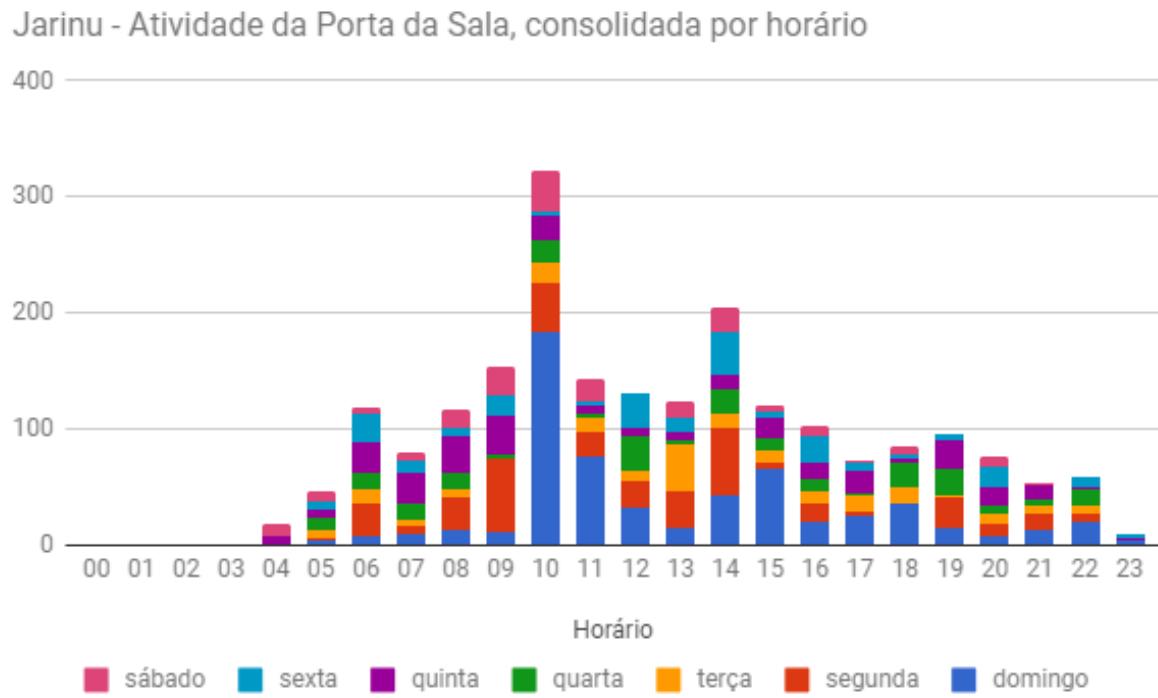


Figura 66: Atividade da porta – Sala de Jarinu



Em Jarinu, um idoso com boa saúde física mora sozinho em sua residência. Ao analisar

as curvas diárias de atividade da porta da sala e presença, notou-se:

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

Figura 67: Atividade da presença – Sala de Jardinu

Jarinu - Atividade de Presença, consolidada por horário

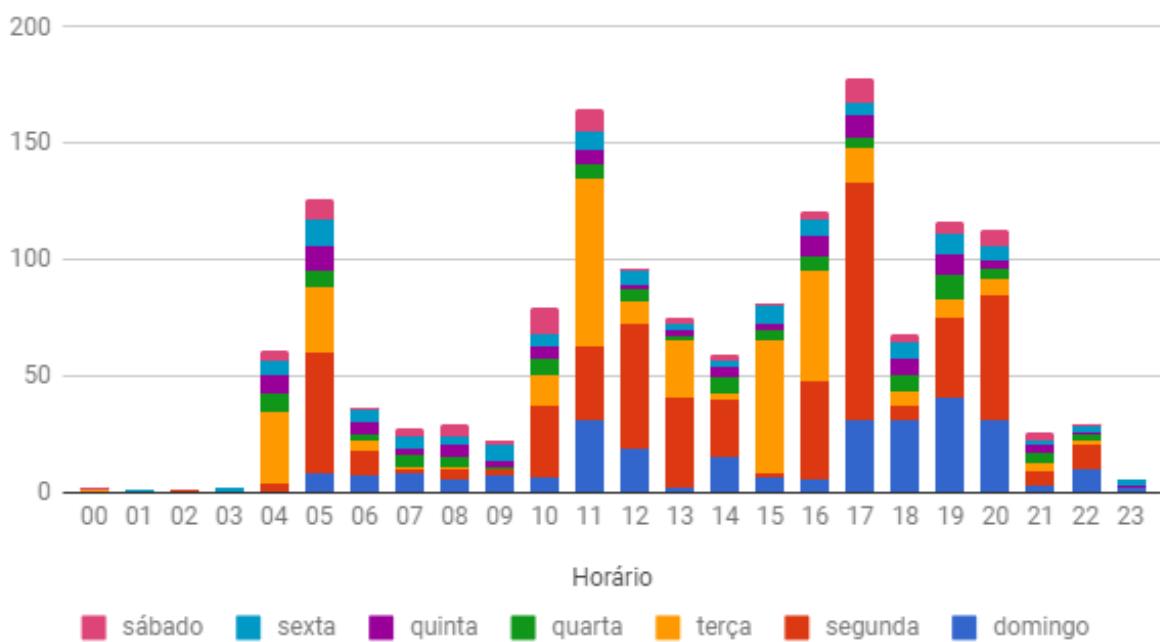


Figura 68: Consolidado no período – Jarinu



Ao analisar por período, pode-se 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 níveis de presença e atividade da porta da sala.

Ainda, a partir do dia 30/11 (último movimento registrado em 29/11 às 16h20), observa-se que não há mais movimentação na residência (seria uma situação para geração de alerta). Nesse caso em específico, foi verificado que o idoso estava de fato fora de casa, através de ligação ao idoso.

É importante destacar que é possível comprovar a continuidade de funcionamento do módulo após 29/11 ao inspecionar seus próprios registros e emails enviados com informações de estado da conexão (envio periódico a cada 6 horas), como forma de mitigar um falso negativo.

7 Arquitetura: Tecnologia - Aprendizado de máquina

7.1 Conceito

O termo Aprendizado de Máquina (*Machine Learning*) surgiu em 1959, sendo utilizado pela primeira vez pelo cientista americano Arthur Samuel. O termo foi por ele definido como ”o campo de estudos que dá a um computador a habilidade de aprender sem ser explicitamente programado” (tradução livre dos autores). Em 1998, Tom Mitchel - outro cientista da computação americano - propôs uma explicação menos abstrata do termo da seguinte maneira: ”um programa de computador aprende com a experiência E com respeito a uma classe de tarefas T e medida de performance P se sua performance nas tarefas T, sendo medida por P, aumenta com o aumento da experiência E”.

Apesar do conceito de aprendizado de máquina existir desde a década de 50, seu crescimento e relevância aumentaram apenas na última década. Alguns fatores levaram a esse crescimento acelerado. Os principais foram o aumento da quantidade de dados - e aplicações de IoT têm propulsionado esse aumento - coletados e disponíveis, a melhora nos algoritmos, e o hardware cada vez mais poderoso dos computadores. Esses últimos dois fatores permitem que a vasta quantidade de dados que possuímos possam ser analisados em tempo viável.

7.2 Relação com o Projeto Hedwig

No projeto Hedwig, a utilização de aprendizado de máquina tem o intuito de trazer melhorias de usabilidade do sistema ao usuário, além de poder trazer medidas de segurança à casa do mesmo.

As melhorias de usabilidade podem ocorrer, por exemplo, quando o sistema aprende alguma rotina do usuário e se torna então capaz de prever quando determinada ação seria solicitada pelo usuário. A partir disso, ele então poderia passar a realizar tal ação auto-

maticamente, ou sugerir a realização de tal ação para o usuário, sem que seja necessário que o usuário proativamente aja para solicitar a ação.

No quesito de segurança, o sistema pode, a partir do aprendizado de rotinas, detectar ações suspeitas na casa (como por exemplo a abertura da porta de entrada em horário não usual), e opcionalmente agir para intervir quando tal tipo de ação é detectada.

7.3 Implementação

7.3.1 Linguagem, Ferramentas e Bibliotecas

Uma das vantagens da popularização que se vê atualmente do uso de aprendizado de máquina é o surgimento de muitas facilidades para o desenvolvimento de aplicações de aprendizado de máquina, devido às vantagens que advém de grande comunidade trabalhando com o tema.

Atualmente as duas principais linguagens sendo utilizadas para ciência de dados e aprendizado de máquina são Python e R. As duas são linguagens interpretadas, e que possuem funcionalidade REPL (Read-Eval-Print-Loop), possibilitando desenvolvimento altamente interativo e de fácil visualização de dados e gráficos, enquanto se programa.

Para o desenvolvimento de funcionalidades de aprendizado de máquina no Hedwig, foi escolhida a linguagem Python. Apesar de R ser uma linguagem que já nasceu voltada à análise de dados e tratamento estatístico, Python possui diversos pacotes e bibliotecas que conseguem adicionar tais funcionalidades à linguagem. Tais pacotes estão altamente popularizados, e portanto é muito fácil percorrer suas documentações e procurar ajuda em fóruns online para acelerar o aprendizado de suas funcionalidades. Python, por sua vez, é uma linguagem de uso geral (*general purpose language*), o que também tem seus lados positivos. Além de ser uma linguagem extremamente bem estabelecida, o fato de ser de uso geral implica que existem também pacotes e bibliotecas que possibilitam o uso de Python para desenvolvimento de aplicativos com funcionalidades de *back-end* para serviços web. Isso se torna de altamente interessante por nos possibilitar facilmente transformar essa aplicação em um microsserviço, que também interaja em tempo real com o servidor em nuvem do Hedwig, permitindo que o serviço de aprendizado de máquina aja em tempo real com toda a aplicação.

Alguns dos pacotes sendo utilizados

7.3.2 Etapas

7.3.3 Algoritmos

7.3.4 Tratamento de Dados

7.3.5 Seleção de Características

8 Metodologia

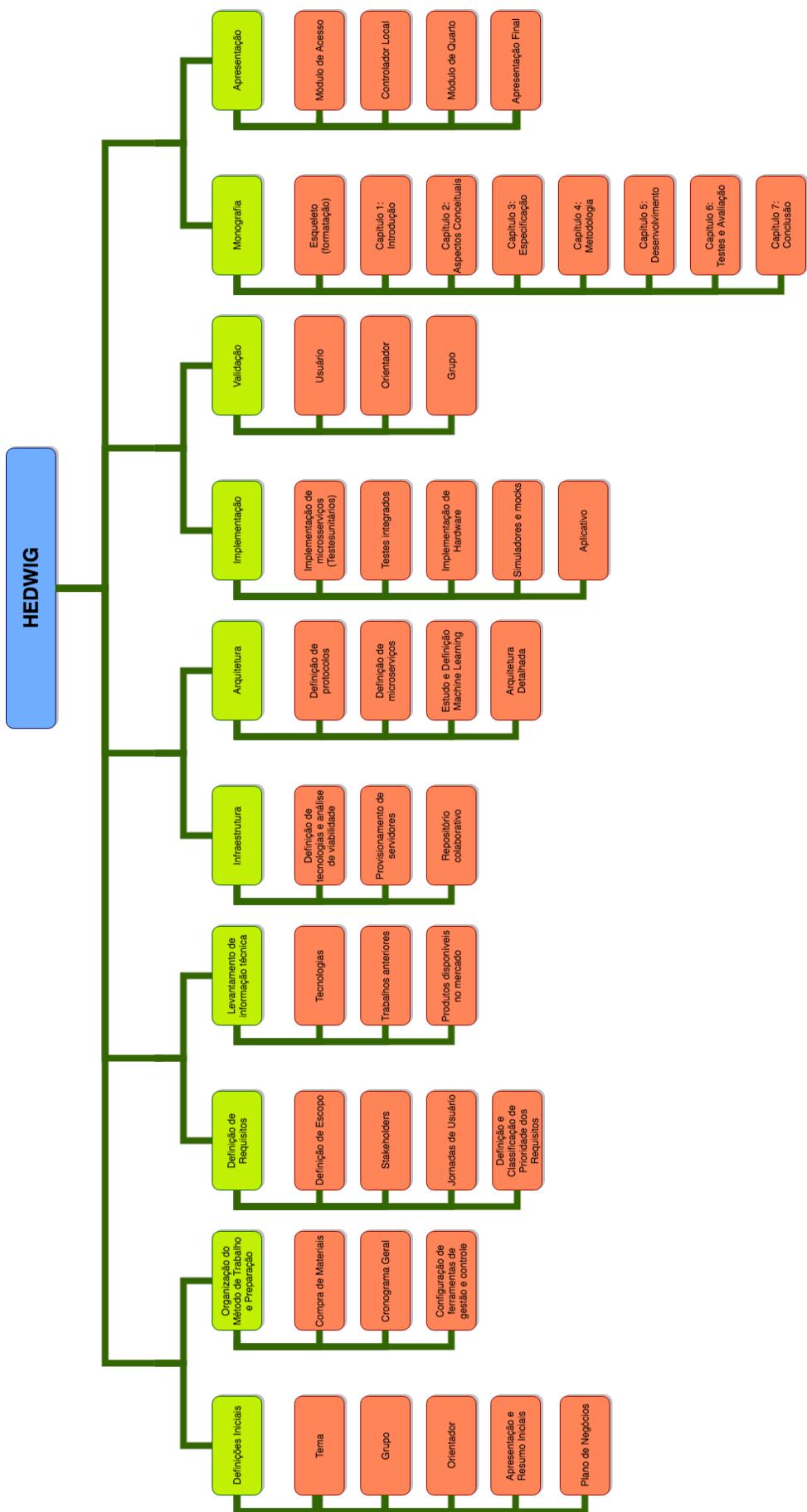
8.1 Gerência do Projeto

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

8.1.1 Gerência de Escopo

Segue a EAP (Estrutura Analítica do Projeto), que orientou as entregas realizadas e escopo do projeto.

Figura 69: EAP do Hedwig



8.1.2 Gerência de Aquisição

O gerenciamento de aquisições envolve primariamente a compra dos materiais necessários para implementação dos módulos físicos. Foi necessário analisar o que adquirir e como fazê-lo, levando em consideração as limitações de tempo do projeto — a demora ou atraso na entrega dos componentes eletrônicos causaria um atraso no cronograma. A escolha dos fornecedores priorizou, então, o custo e o prazo de entrega.

A relação de materiais necessários para a confecção de um módulo básico encontra-se no Anexo B.

8.1.3 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 criou-se a organização Hedwig¹ e os repositórios que contém o código das diversas partes do 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). O GitHub disponibiliza também uma ferramenta para armazenamento de *Issues* — atividades que necessitam ser observadas ou desenvolvidas em um projeto. Esse recurso foi essencial na organização de tarefas e rastreamento de mudanças em curso.

Para a coordenação de trabalho nesses repositórios, foi utilizado o fluxo conhecido como *Feature Branch Workflow* (ATLASSIAN, 2017), caracterizado pela criação de *branches* (ramificações) para o desenvolvimento de cada nova funcionalidade. Ao final do desenvolvimento de cada funcionalidade, é feito um pedido para mesclar à *master branch*, ramificação principal, o código desenvolvido na ramificação atual — conhecido como *Pull Request*. A utilização de *Pull Requests* promove maior controle sobre a atualização do código mantido, e também promove a revisão do código pelos aprovadores.

Já outros tipos de documentação formal do sistema, como textos e planilhas, foram editados e armazenados no Google Drive², ferramenta de armazenamento e backup da Google que também permite colaboração, compartilhamento e controle de versão.

¹<https://github.com/hedwig-project>

²<https://www.google.com/drive/>

8.1.4 Gerência de Comunicação

Para comunicar as tarefas, estudos e pesquisas sendo realizadas durante o projeto, foi utilizado o Trello³, um sistema online para organização de ideias e projetos, que permite listagem e acompanhamento das atividades a serem realizadas, permitindo a adição de prazos, delegação de responsáveis e categorização das tarefas.

Foram realizadas diversas reuniões online usando ferramentas de videoconferência, como o Google Hangouts⁴ e o Skype⁵, que, além do *streaming* de vídeo com áudio, possuem funcionalidades como mensagens de texto, envio de arquivo e compartilhamento de tela, facilitando demonstrações e testes integrados.

8.1.5 Gerência de Riscos

Para gerenciar os riscos do projeto, foram realizadas reuniões periódicas com participação das partes interessadas — grupo e orientadores — nas quais discutiam-se táticas de análise, planejamento e monitoramento. Nessas sessões, foi possível identificar riscos por meio da revisão da documentação e de técnicas de *brainstorming* e definir estratégias de abordagem de riscos.

8.2 Pesquisa Bibliográfica

O estudo dos tópicos relacionados a aprendizagem de máquina foi realizado com auxílio do curso *Aprendizagem Automática*, do Professor Andrew Ng⁶, 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 Hopkins⁷, 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*.

³<https://trello.com/>

⁴<https://hangouts.google.com>

⁵<https://www.skype.com>

⁶<https://www.coursera.org/learn/machine-learning>

⁷<https://www.coursera.org/specializations/jhu-data-science>

8.3 Ferramentas e Tecnologias

Para a aprendizagem da biblioteca React, para o desenvolvimento do *front-end*, foi usada como referência a documentação oficial⁸ oferecida pelo Facebook e o curso *React for Beginners* de Wes Bos⁹. O aprendizado de Redux foi auxiliado pelo curso *Learn Redux*¹⁰, do mesmo autor. O conhecimento necessário à utilização do Spring Boot foi obtido com base no guia de referência oficial¹¹.

8.4 Método de Testes

A estratégia de testes utilizada está fortemente relacionada com a estratégia de desenvolvimento. A implementação do projeto foi realizada em partes, segregadas de acordo a área de aplicação. Assim, as diversas frentes puderam ser desenvolvidas em paralelo, como os módulos, servidor local, aplicativo cliente, etc. juntamente com os seus respectivos testes.

Para cada tarefa específica foram efetuados testes unitários, de pequeno alcance, para validar a funcionalidade ou alteração. O desenvolvimento do servidor local contou também com testes unitários automatizados, entretanto sem cobertura do sistema inteiro. Para testes específicos dos *endpoints* do servidor na nuvem foi utilizada a ferramenta Postman¹², que permite simular requisições a tais *endpoints*, configurando todos os parâmetros necessários e informações de cabeçalho HTTP, e apresenta ao usuário a resposta recebida.

Durante a integração das partes, houveram diversos testes ponto-a-ponto, para verificação do funcionamento completo do sistema. Utilizou-se para isso o Software MQTT Fx¹³, onde é possível obter as mensagens encaminhadas pelo sistema de mensageria, e compará-las com os resultados esperados. Na Subseção 5.2.7.5 são mostrados alguns casos de testes utilizados durante a configuração do Broker Mosquitto, enquanto a seção H contém um exemplo de teste fim a fim realizado com aplicativo, servidor local, nuvem e módulo.

Com a instalação do sistema em duas residências distintas, houve uma validação contínua de premissas e levantamento de requisitos, principalmente aqueles relacionados à

⁸<https://facebook.github.io/react/docs/hello-world.html>

⁹<https://reactforbeginners.com/>

¹⁰<https://learnredux.com>

¹¹<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

¹²<https://www.getpostman.com/>

¹³<http://mqttx.jensd.de/>

usabilidade. Foram realizados diversos testes de conceito em campo, de forma que muitas soluções, embora fizessem sentido experimentalmente em implementações isoladas, não eram adequadas para módulos instalados em uma residência real. Ainda, atualizações e melhorias foram entregues frequentemente, contribuindo para o crescimento iterativo do projeto (destacam-se a aplicação dos princípios 2 e 4 do Agile Manifesto¹⁴, que dissertam sobre mudanças frequentes de requisitos e entregas frequentes de software).

Em especial, o contato do desenvolvedor dos módulos com todos os problemas e desafios encontrados com o uso do sistema em sua residência facilitou sua proximidade com um dos stakeholders principais, possibilitando um *feedback* rápido, reduzindo o ciclo de desenvolvimento consideravelmente.

¹⁴<http://agilemanifesto.org/iso/ptbr manifesto.html>

9 Conclusões

O principal foco deste trabalho foi a criação de uma arquitetura abrangente e robusta para automação residencial, com sua implementação completa — desde os módulos de hardware à aplicação cliente.

A fim de demonstrar todas as funcionalidades dos módulos, por meio do aplicativo web, optou-se pelo modelo de *dashboard*, que promove interação com fluidez e concisão. Porém, a arquitetura permite que outros tipos de aplicação possam se comunicar com as casas. O processamento de linguagem natural com um *chatbot*, por exemplo, ajudaria o morador a interagir com a casa, guiando-o durante o processo de criação de regras para acionamento automático de dispositivos. A área de *Business Intelligence* também oferece um vasto campo a ser explorado, com a criação de um aplicativo capaz de exibir dados da casa em gráficos parametrizados, com relatórios sobre o comportamento dos usuários. Como demonstrado na Seção 6 e Anexo G, são obtidas informações relevantes sobre a vida dos moradores de uma residência mesmo com um número limitado de sensores e atuadores em uso.

Para a implementação atual, não foram realizados testes de carga para o servidor na nuvem, o qual foi implementado como um monolito, em uma única instância. O próximo passo seria avaliar os melhores procedimentos de escalabilidade e estudo de sua performance com vários dispositivos e usuários conectados, de modo que seja possível a identificação de gargalos em seus componentes. Como visto na Seção 5.3, as tecnologias empregadas na implementação do servidor na nuvem — nginx, Node.js, MongoDB e Redis — possuem características favoráveis à sua utilização em aplicações altamente escaláveis. O desafio, a partir daí, se concentraria na redundância para a comunicação com as casas, onde são utilizadas conexões WebSocket.

O monitoramento do servidor de nuvem também pode ser explorado, de maneira que sejam entendidas suas características de uso, peça essencial para o alcance de maior disponibilidade e robustez. A integração realizada durante o projeto foi simplificada e usou um serviço que apenas monitora uso de memória e CPU. Idealmente, deve ser possível

também acompanhar eventos e transações importantes que ocorrem na nuvem, configurar alertas de picos de uso de recursos e poder ter uma visão consolidada dos logs, com busca e estatísticas.

Nos aspectos de hardware, e da infraestrutura de comunicação, podem ser explorados outros canais, transparentes ao usuário, e que seriam capazes de oferecer maior robustez à aplicação. Ao ampliar as formas de comunicação disponíveis, os módulos poderiam fazer uso de redes de dados — 4G e futuramente 5G — e não seriam mais dependentes exclusivamente da internet local. A integração, e também o desenvolvimento, de dispositivos vestíveis, integráveis com a casa, é também um passo futuro, que melhorará a experiência do usuário e expandirá as possibilidades de aplicação —como, por exemplo, um relógio que monitora pessoas idosas, e envia notificações e alertas aos familiares.

A possibilidade de atualização de *firmware* remotamente agregou em segurança e flexibilidade ao projeto, já que torna possível o envio de correções diretamente ao usuário, por meio da internet, sem a necessidade do contato direto com o módulo. Assim, frente a uma potencial ameaça, pode-se desenvolver uma correção e enviá-la aos clientes, que manteriam suas casas atualizadas.

Como outras sugestões aos próximos passos e caminhos futuros, indica-se a criação de testes unitários e automatizados, que verificam pequenas porções de código por vez. Isso facilitaria a adição de novas funcionalidades, garantindo a compatibilidade reversa, além de aumentar as chances de identificar falhas antes de liberar novas versões de software ao público. Com a mesma mentalidade, pode-se implementar uma *pipeline* de integração contínua para identificar erros de integração rapidamente por meio de testes e verificação de código e permitir o lançamento de novas versões de maneira ágil. O fato do código-fonte do Hedwig já estar em um sistema de controle de versão, o GitHub, é um acelerador para o uso de serviços de integração contínua. Outro importante aspecto relacionado às tecnologias de Internet das Coisas, mas que não foi coberto neste trabalho são as questões éticas e relacionadas à privacidade do usuário, as quais são complexas e necessitarão de grandes esforços para a elaboração de uma legislação adequada.

REFERÊNCIAS

- 9 TO 5 MAC. *Jobs' original vision for the iPhone: No third-party native apps.* 2011. Disponível em: <<https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/>>. Acesso em: 07 nov. 2017.
- AKAMAI. *Denial of service attacks (DoS).* 2017. Disponível em: <<https://www.akamai.com/us/en/resources/protect-against-ddos-attacks.jsp>>. Acesso em: 10 nov. 2017.
- ATLASSIAN. *Comparing Workflows.* 2017. Disponível em: <<https://www.atlassian.com/git/tutorials/comparing-workflows>>. Acesso em: 17 abr. 2017.
- BENSON, M. *An end or an error: App downloads are on a dangerous decline in the USA.* 2016. Disponível em: <<https://www.androidauthority.com/end-era-app-downloads-decline-usa-698555/>>. Acesso em: 10 nov. 2017.
- BIBLIOTECA VIRTUAL. *São Paulo: população do estado.* 2017. Disponível em: <<http://www.bibliotecavirtual.sp.gov.br/temas/sao-paulo/sao-paulo-populacao-do-estado.php>>. Acesso em: 20 fev. 2017.
- BROWSER COOKIE LIMITS. *Browser Cookie Limits.* 2016. Disponível em: <<http://browsercookielimits.squawky.net/>>. Acesso em: 08 nov. 2017.
- BUSINESS WIRE. *Internet of Things Spending Forecast to Grow 17.9% in 2016 Led by Manufacturing, Transportation, and Utilities Investments, According to New IDC Spending Guide.* 2017. Disponível em: <<http://www.businesswire.com/news/home/20170104005270/en/Internet-Spending-Forecast-Grow-17.9-2016-Led>>. Acesso em: 17 nov. 2017.
- CHAMPEON, S. *Progressive Enhancement and the Future of Web Design.* 2003. Disponível em: <http://hesketh.com/publications/progressive_enhancement_and_the_future_of_web_design.html>. Acesso em: 11 nov. 2017.
- CHEN, A. *New data shows losing 80% of mobile users is normal, and why the best apps do better.* 2015. Disponível em: <<https://goo.gl/ubcySV>>. Acesso em: 10 nov. 2017.
- CISCO SYSTEMS. *The Zettabyte Era: Trends and Analysis.* 2017. Disponível em: <<https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf>>. Acesso em: 09 nov. 2017.
- COPE, S. *Configuring and Testing Mosquitto MQTT Topic Restrictions.* 2017. Disponível em: <<http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/>>. Acesso em: 28 ago. 2017.

COPE, S. *Introduction to MQTT Security Mechanisms*. 2017. Disponível em: <<http://www.steves-internet-guide.com/mqtt-security-mechanisms/>>. Acesso em: 28 ago. 2017.

DHWTY. *Morpheus, the Greek God of Dreams who delivered messages from the gods to the mortal world*. 2014. Disponível em: <<http://www.ancient-origins.net/myths-legends-europe/morpheus-greek-god-dreams-who-delivered-messages-gods-mortal-world-002318?nopaging=1>>. Acesso em: 02 jul. 2017.

ELECTRONIC FRONTIER FOUNDATION MEDIA RELEASE. *Movie Legend Hedy Lamarr to be Given Special Award at EFF's Sixth Annual Pioneer Awards*. 1997. Disponível em: <<https://w2.eff.org/awards/pioneer/1997.php>>. Acesso em: 15 mai. 2017.

ESPRESSIF SYSTEMS. *ESP8266EX Datasheet*. 2015. Disponível em: <<http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf>>. Acesso em: 22 mai 2017.

FACEBOOK OPEN SOURCE. *Building the F8 2016 App: Integrating Data with React Native*. 2016. Disponível em: <<http://makeitopen.com/docs/en/1-3-data.html>>. Acesso em: 11 nov. 2017.

FACEBOOK OPEN SOURCE. *Reconciliation*. 2017. Disponível em: <<https://reactjs.org/docs/reconciliation.html>>. Acesso em: 09 nov. 2017.

FOWLER, M. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. Disponível em: <<https://martinfowler.com/articles/injection.html>>. Acesso em: 25 jun. 2017.

G1 SÃO PAULO. *Grande SP atinge 83Sabesp*. 2015. Disponível em: <<http://glo.bo/1K5JsQh>>. Acesso em: 20 fev. 2017.

GITHUB. *The state of the Octoverse 2016*. 2016. Disponível em: <<https://octoverse.github.com/>>. Acesso em: 17 mai. 2017.

GOOGLE CHROME. *Managing HTML5 Offline Storage*. 2017. Disponível em: <https://developer.chrome.com/apps/offline_storage>. Acesso em: 08 nov. 2017.

GOOGLE DEVELOPERS. *AliExpress*. 2016. Disponível em: <<https://developers.google.com/web/showcase/2016/aliexpress>>. Acesso em: 08 nov. 2017.

GOOGLE DEVELOPERS. *eXtra Electronics*. 2016. Disponível em: <<https://developers.google.com/web/showcase/2016/extra>>. Acesso em: 08 nov. 2017.

GOOGLE DEVELOPERS. *Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage*. 2016. Disponível em: <<https://developers.google.com/web/showcase/2017/twitter>>. Acesso em: 08 nov. 2017.

GOOGLE DEVELOPERS. *Progressive Web App Checklist*. 2017. Disponível em: <<https://developers.google.com/web/progressive-web-apps/checklist>>. Acesso em: 07 nov. 2017.

GOOGLE DEVELOPERS. *Progressive Web Apps*. 2017. Disponível em: <<https://developers.google.com/web/progressive-web-apps/>>. Acesso em: 07 nov. 2017.

HIVEMQ. *HiveMQ Enterprise MQTT Broker*. 2017. Disponível em: <<https://www.hivemq.com/wp-content/uploads/hivemq-data-sheet-3.3.pdf>>. Acesso em: 05 nov. 2017.

HOPKINS, B. *Blu-ray Disc Application Development with Java ME*. 2008. Disponível em: <<http://www.oracle.com/technetwork/articles/javame/bluray-142687.html>>. Acesso em: 13 ago. 2017.

I-SCOOP. *IoT in smart home automation: the fast growing list of use cases*. 2017. Disponível em: <<https://www.i-scoop.eu/smart-home-home-automation/iot-home-automation-applications/>>. Acesso em: 17 nov. 2017.

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. Disponível em: <<http://cod.ibge.gov.br/6QV>>. Acesso em: 20 fev. 2017.

INTERNET ENGINEERING TASK FORCE. *The WebSocket Protocol*. 2011. Disponível em: <<https://tools.ietf.org/html/rfc6455>>. Acesso em: 19 nov. 2017.

INTERNET ENGINEERING TASK FORCE. *JSON Web Token (JWT)*. 2015. Disponível em: <<https://tools.ietf.org/html/rfc7519>>. Acesso em: 08 nov. 2017.

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. Disponível em: <<https://jwt.io/introduction/>>. Acesso em: 08 nov. 2017.

KENNEMER, Q. *Google Home gets a \$5 million ad spot in the Superbowl*. 2017. Disponível em: <<http://phandroid.com/2017/02/01/google-home-gets-a-5-million-ad-spot-in-the-super-bowl>>. Acesso em: 15 mai. 2017.

LAMPKIN, V. et al. *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. [S.l.]: IBM Red Books, 2012. ISBN 0738437085.

LEWIS, J.; FOWLER, M. *Microservices: a definition of this new architectural term*. 2014. Disponível em: <<https://martinfowler.com/articles/microservices.html>>. Acesso em: 22 mai. 2017.

LUBBERS, P.; GRECO, F. *HTML5 WebSocket: A Quantum Leap in Scalability for the Web*. 2016. Disponível em: <<https://websocket.org/quantum.html>>. Acesso em: 19 nov. 2017.

MCGRAW, G. *Software Security*. 2004. Disponível em: <<http://ieeexplore.ieee.org/abstract/document/1281254/>>. Acesso em: 21 out. 2017.

MCKINSEY & COMPANY. *There's No Place Like A Connected Home*. 2016. Disponível em: <http://www.mckinsey.com/spContent/connected_homes/index.html>. Acesso em: 24 abr. 2017.

MONGODB. *Sharding*. 2017. Disponível em: <<https://docs.mongodb.com/manual/sharding/>>. Acesso em: 02 dez. 2017.

MOZILLA DEVELOPER NETWORK. *JavaScript*. 2016. Disponível em: <<https://developer.mozilla.org/bm/docs/Web/JavaScript>>. Acesso em: 11 nov. 2017.

MQTT. *MQTT Version 3.1.1*. 2014. Disponível em: <<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>>. Acesso em: 21 jun. 2017.

MULLINS, C. S. *What is an In-Memory Database System?* 2017. Disponível em: <<http://www.dbta.com/Columns/DBA-Corner/What-is-an-In-Memory-Database-System-119241.aspx>>. Acesso em: 19 nov. 2017.

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. Disponível em: <https://www.theregister.co.uk/2009/01/16/half_billion_iphone_apps/>. Acesso em: 10 nov. 2017.

NETWORK WORKING GROUP. *The Syslog Protocol*. 2009. Disponível em: <<https://tools.ietf.org/html/rfc5424>>. Acesso em: 20 nov. 2017.

NGINX. *Inside NGINX: How We Designed for Performance & Scale*. 2017. Disponível em: <<https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/>>. Acesso em: 02 dez. 2017.

NODE.JS. *Node.js v9.2.0 Documentation - Cluster*. 2017. Disponível em: <<https://nodejs.org/api/cluster.html>>. Acesso em: 02 dez. 2017.

NOSQL DATABASES. *NOSQL Databases*. 2009. Disponível em: <<http://nosql-database.org/>>. Acesso em: 17 nov. 2017.

O'LEARY, N. *Arduino Client for MQTT*. 2017. Disponível em: <<https://pubsubclient.knolleary.net/index.html>>. Acesso em: 28 ago. 2017.

PEREIRA, T. *Quando utilizar RDBMS ou NoSQL?* 2016. Disponível em: <<http://datascienceacademy.com.br/blog/quando-utilizar-rdbms-ou-nosql/>>. Acesso em: 19 nov. 2017.

PROJECT MANAGEMENT INSTITUTE. *PMI. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos. Guia PMBOK®*. 2004, 3^a ed. 3. ed. [S.I.]: PMI, 2004.

RAIMA. *In-Memory Advantages From RDM*. 2013. Disponível em: <<https://raima.com/in-memory-database/>>. Acesso em: 19 nov. 2017.

RASPBERRY PI FOUNDATION. *Model 3*. 2017. Disponível em: <<https://www.raspberrypi.org/products/raspberry-pi-3-model-b>>. Acesso em: 12 jun. 2017.

REDUX. *Introduction*. 2017. Disponível em: <<https://redux.js.org/docs/introduction/>>. Acesso em: 09 nov. 2017.

RICKER, T. *Jobs: App Store launching with 500 iPhone applications, 25free*. 2008. Disponível em: <<https://www.engadget.com/2008/07/10/jobs-app-store-launching-with-500-iphone-applications-25-free/>>. Acesso em: 10 nov. 2017.

ROTEM-GAL-OZ, A. *Services, Microservices, Nanoservices – oh my!* 2014. Disponível em: <<http://arnon.me/2014/03/services-microservices-nanoservices/>>. Acesso em: 23 mai. 2017.

RUDOLPH, P. *Hybrid Mobile Apps: Providing A Native Experience With Web Technologies*. 2014. Disponível em: <<https://www.smashingmagazine.com/2014/10/providing-a-native-experience-with-web-technologies/>>. Acesso em: 08 nov. 2017.

RUSELL, A. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*. 2015. Disponível em: <<https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>>. Acesso em: 07 nov. 2017.

SAVIT, J. *Availability Best Practices - Avoiding Single Points of Failure*. 2013. Disponível em: <<https://blogs.oracle.com/jsavitz/availability-best-practices-avoiding-single-points-of-failure>>. Acesso em: 19 out. 2017.

SECURITY AFFAIRS. *1Tbps no ataque contra OVH (serviço de hosting), com dispositivos de IoT (09/2016)*. 2017. Disponível em: <<http://securityaffairs.co/wordpress/51640/cyber-crime/tbps-ddos-attack.html>>. Acesso em: 02 dez. 2017.

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. Disponível em: <<http://www.sublimelinter.com/en/latest/about.html>>. Acesso em: 09 nov. 2017.

THOMSEN, A. *Como programar o NodeMCU com IDE Arduino*. 2016. Disponível em: <<https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/>>. Acesso em: 22 mai. 2017.

VISWANATHAN, P. *Cloud Computing and Is it Really All That Beneficial?* 2017. Disponível em: <<https://www.lifewire.com/cloud-computing-explained-2373125>>. Acesso em: 18 nov. 2017.

W3C SCHOOLS. *HTML5 Web Storage*. 2017. Disponível em: <https://www.w3schools.com/html/html5_webstorage.asp>. Acesso em: 22 nov. 2017.

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 – SCRIPT PARA ADIÇÃO DE NOVA RESIDÊNCIA AO MOSQUITTO

```
#!/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_Casa_$count_ao_firewall..."

`sudo ufw allow "$morpheus_port_number"/tcp > /dev/null`
`sudo ufw allow "$modules_port_number"/tcp > /dev/null`
echo "Adicionando_Casa_$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!'
```

ANEXO D – CONFIGURAÇÕES DO MOSQUITTO

Para a demonstração da implementação, foram configuradas instâncias independentes do MQTT Mosquitto Broker, em um servidor virtual. Este anexo documenta as configurações disponíveis para conexão com tais instâncias. Os passos abaixo com o *firewall* são manuais. Eles estão concentrados em um script, que deve ser executado após o cadastro de uma nova casa (necessário alterar o contador do script).

Execução do Script

. start.sh

Passos manuais (não necessários se o script acima for executado)

Para cadastrar novo módulo, primeiro desativar o *firewall*: sudo ufw disable

Cadastrar o módulo e liberar a porta: sudo ufw allow <porta>/tcp

Ativar o *firewall*: sudo ufw enable

Configurações

Mosquitto 1

Morpheus

IP: 138.197.83.143

Porta: 8883

MorpheusId: adf654wae84fea5d8ea6

Password: 123456789

Modulos

IP: 138.197.83.143

Porta: 1883

Passwd file
00164304:359985
adf654wae84fea5d8ea6:123456789

Mosquitto 2

Morpheus
IP: 138.197.83.143
Porta: 8884
MorpheusId: asd561asd5asd984faee
Password: 852456987

Modulos
IP: 138.197.83.143
Porta: 1884

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

Mosquitto 3

Morpheus
IP: 138.197.83.143
Porta: 8885
MorpheusId: e5d5e8f4w8e89wef48s9
Password: 159753

Modulos
IP: 138.197.83.143
Porta: 1885

Passwd file
00163205:958668

e5d5e8f4w8e89wef48s9:159753

Mosquitto 4

Morpheus

IP: 138.197.83.143

Porta: 8886

MorpheusId: eegeo345439do0982o29

Password: 987456321

Modulos

IP: 138.197.83.143

Porta: 1886

Passwd file

01681382:664361

eegeo345439do0982o29:987456321

ANEXO E – TELAS DO APLICATIVO WEB — DASHBOARD

Figura 70: Tela de login

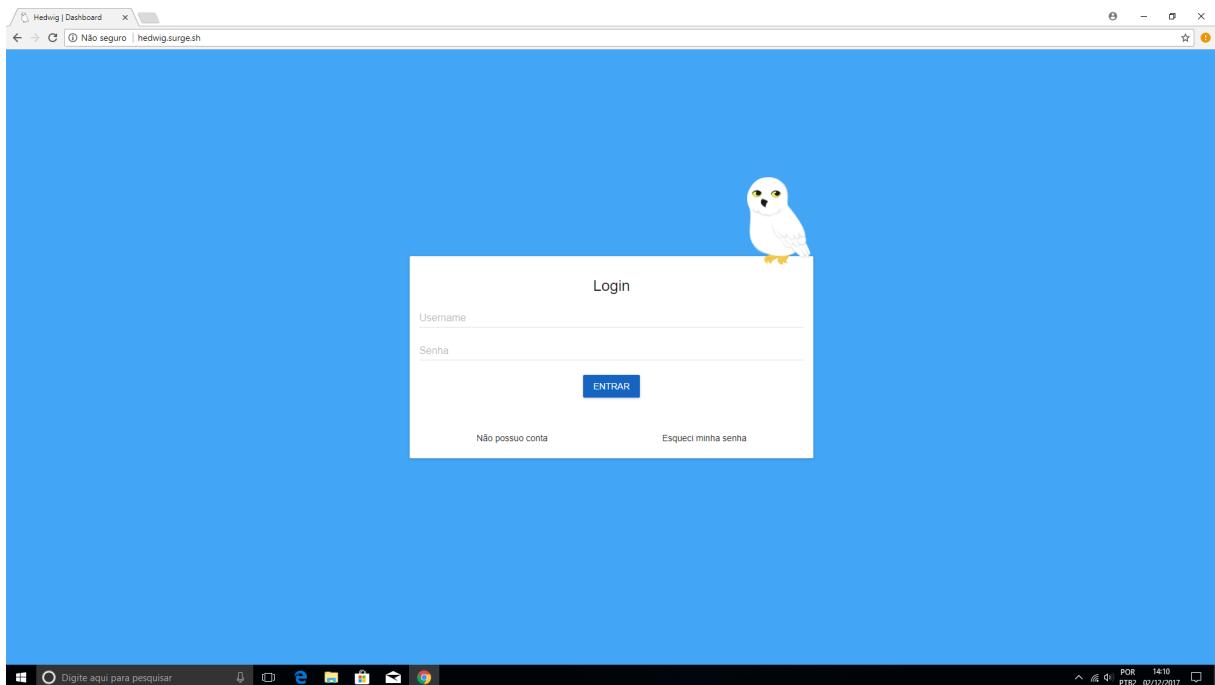


Figura 71: Tela de cadastro

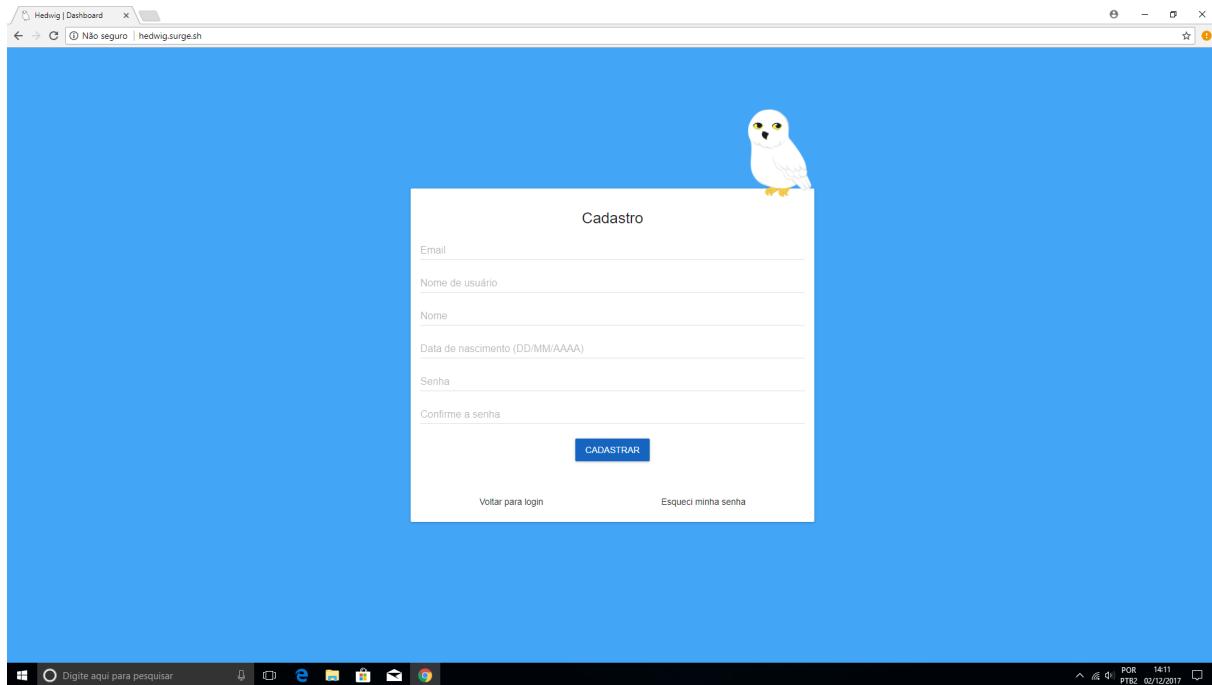


Figura 72: Tela de adição de Morpheus

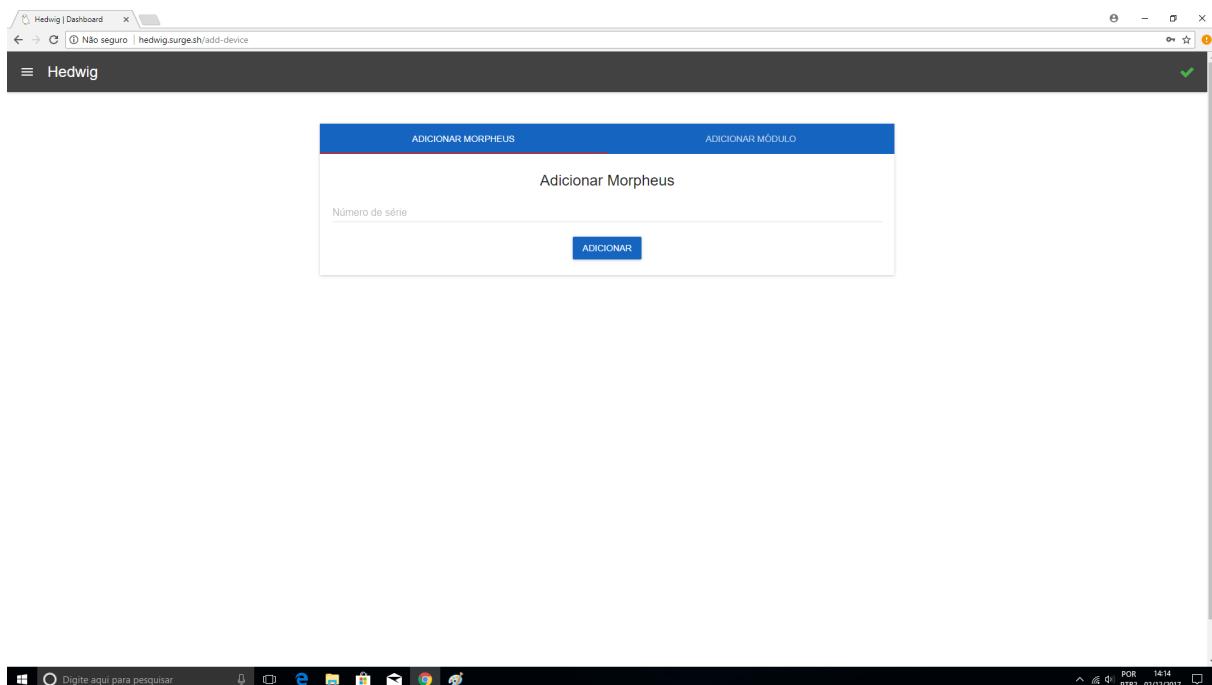


Figura 73: Tela de adição de módulo

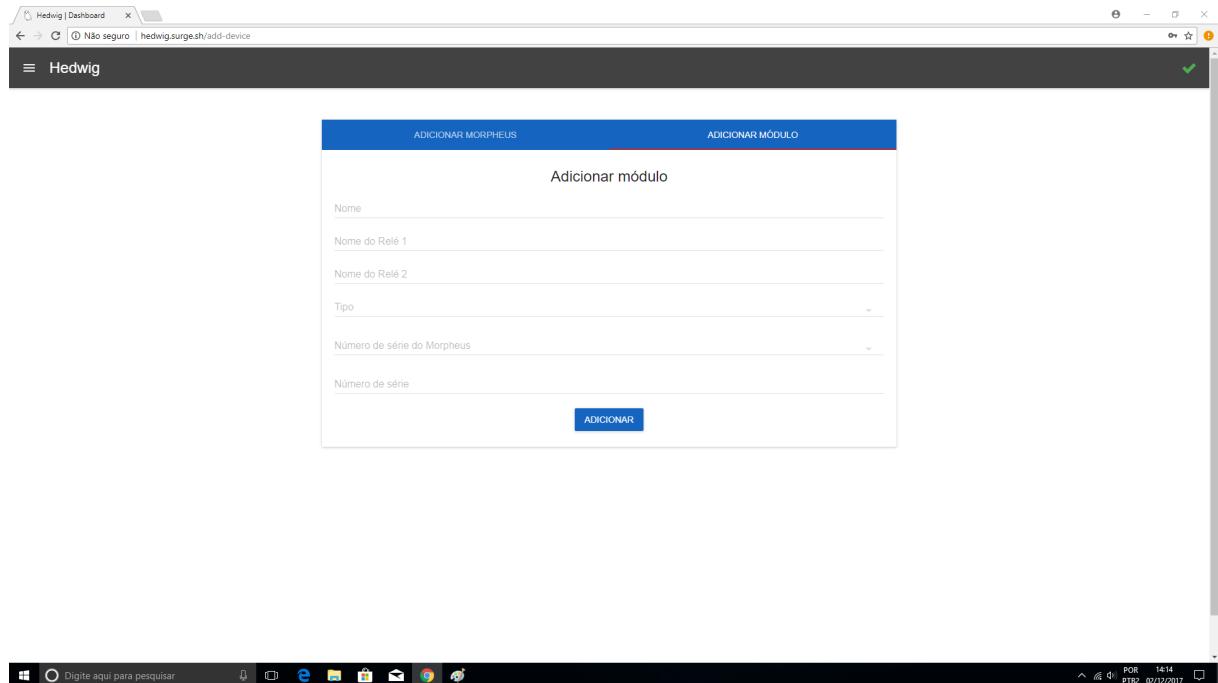


Figura 74: Tela de configuração de Morpheus

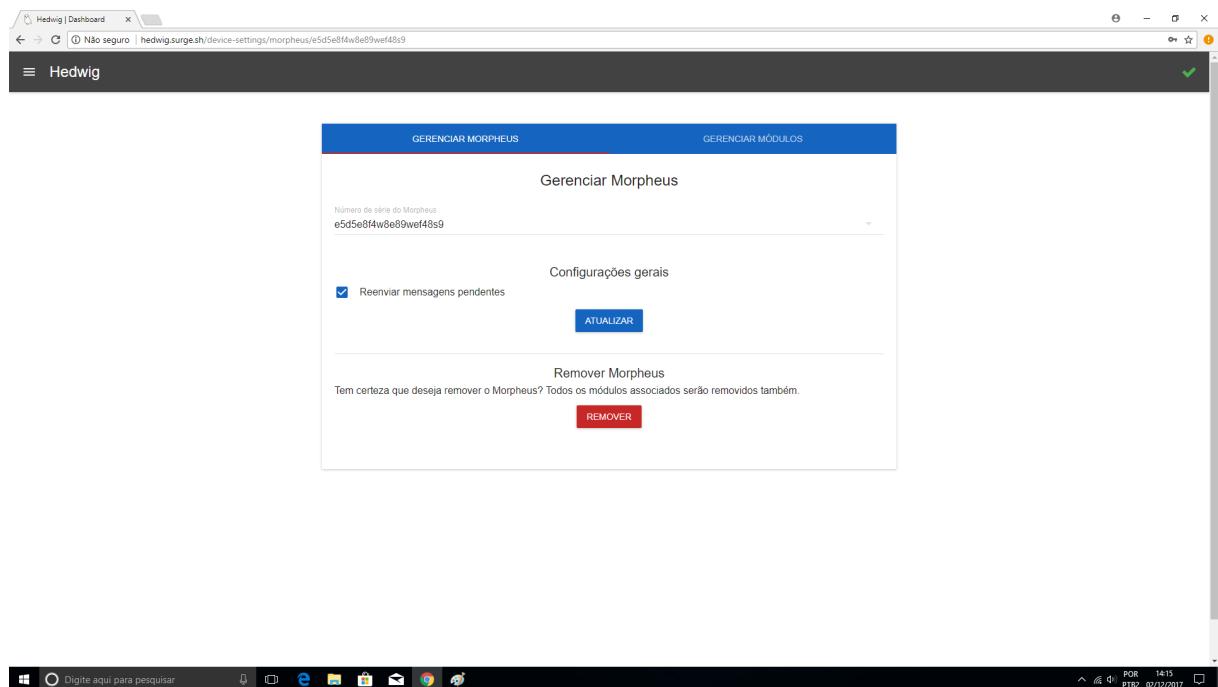


Figura 75: Tela de configuração de módulo

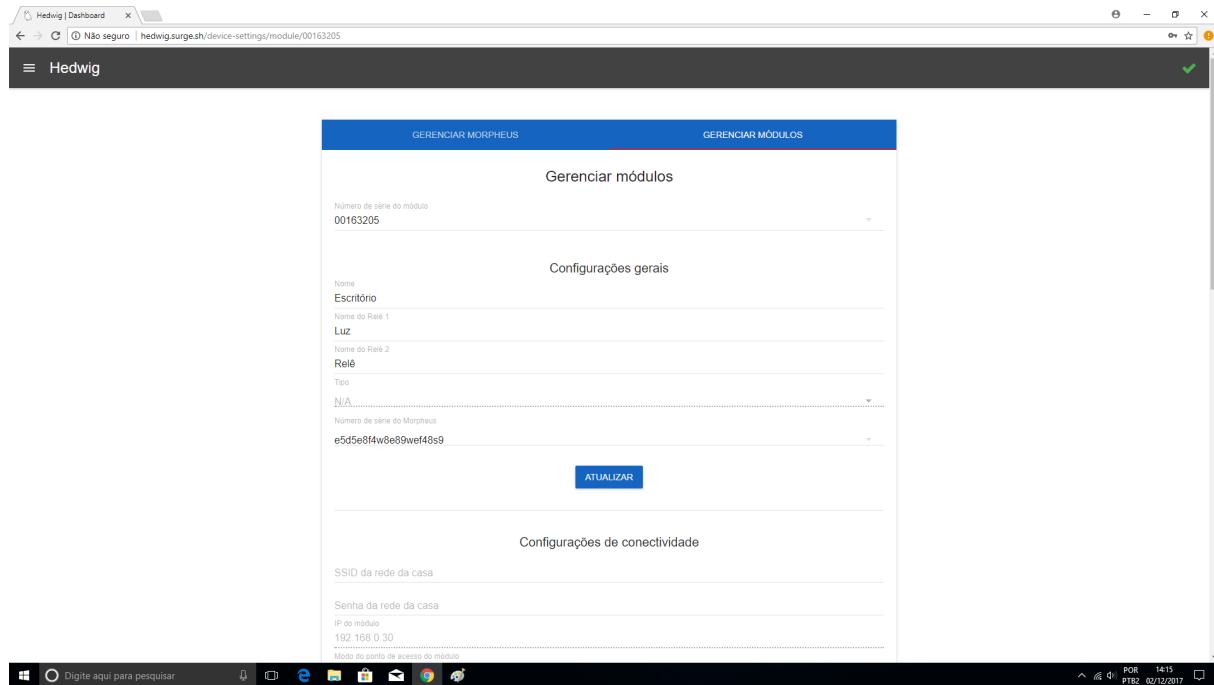


Figura 76: Tela de configurações de usuário

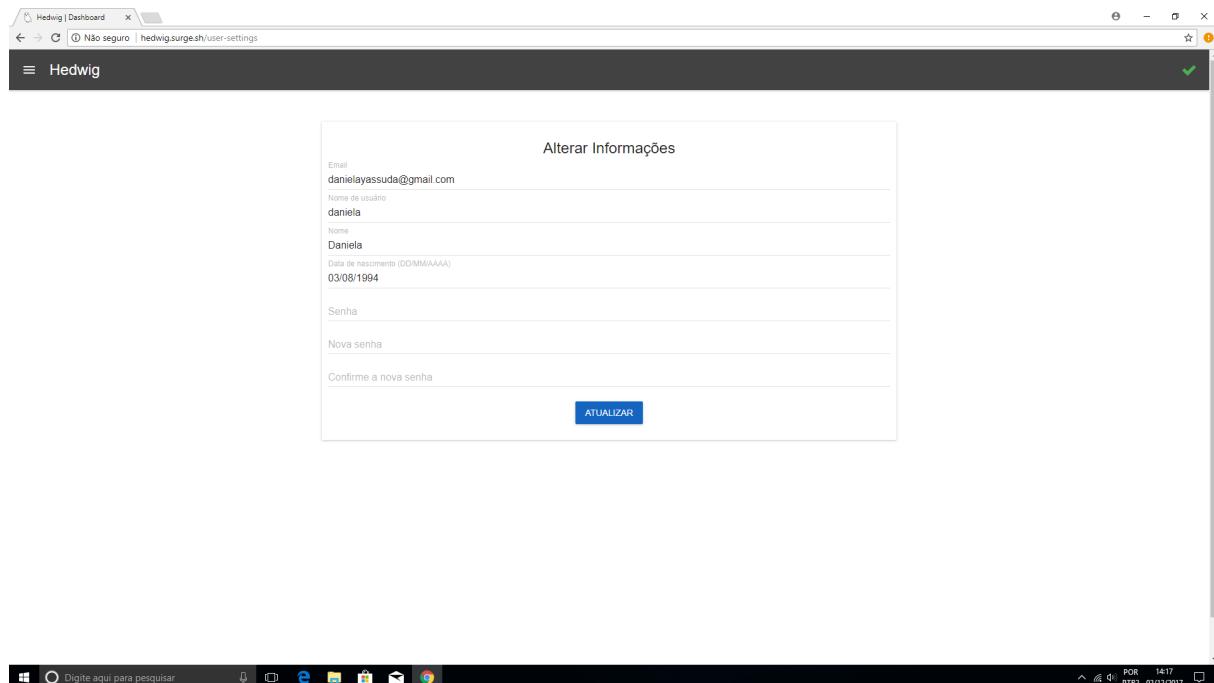


Figura 77: Tela mostrando se os Morpheus do usuário estão devidamente conectados

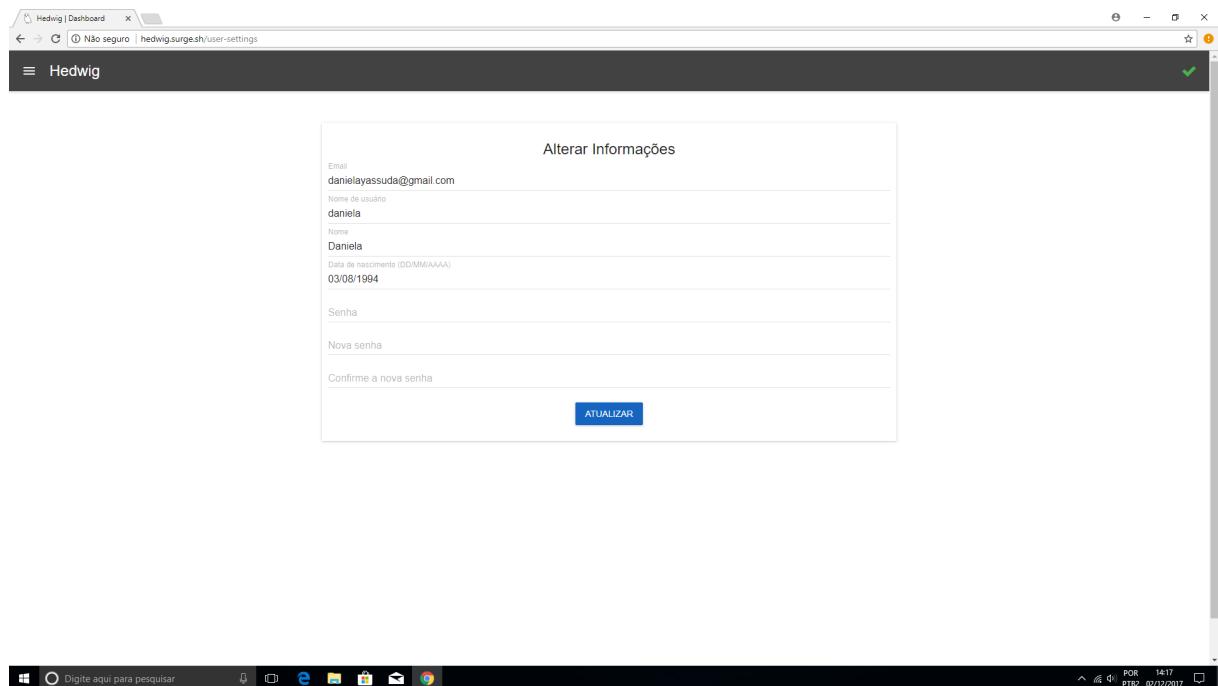


Figura 78: Ícone do aplicativo na tela inicial do celular

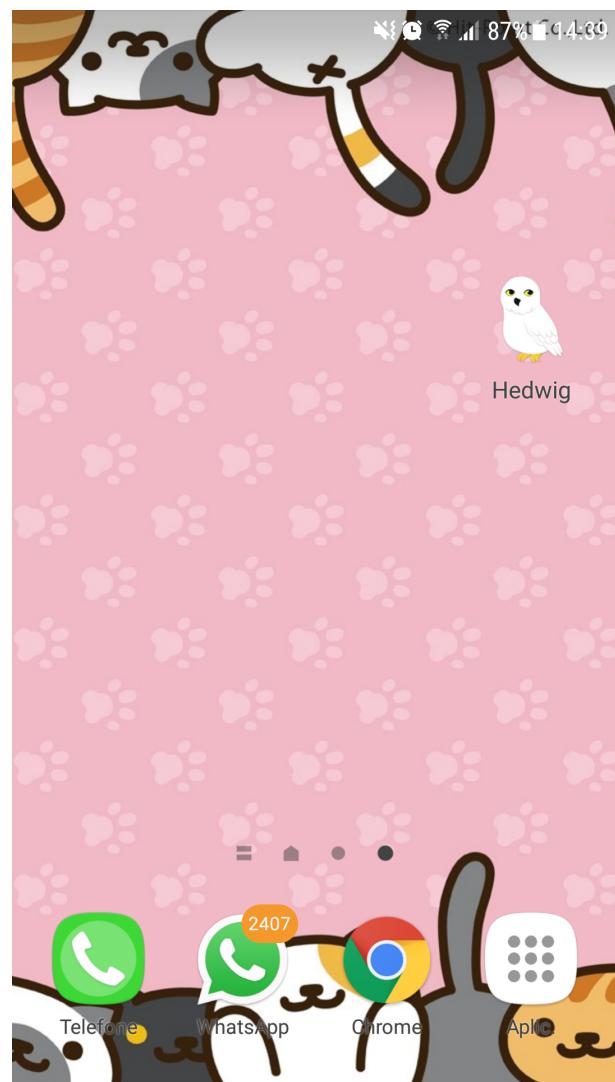


Figura 79: Tela de login e tela principal com menu aberto no celular

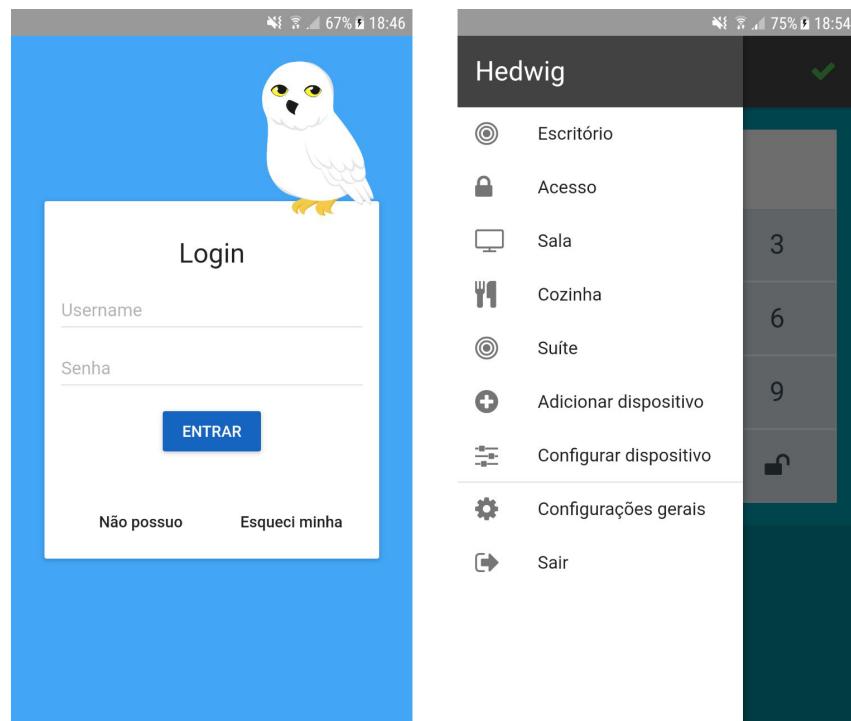
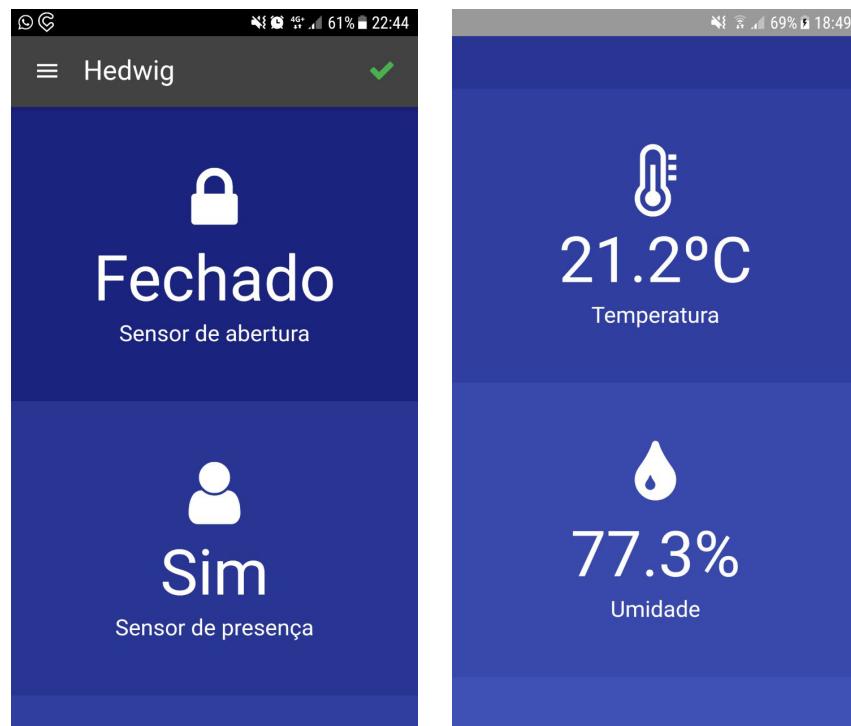


Figura 80: Dados de sensores apresentados no celular



ANEXO F – IMAGENS DE CONFIGURAÇÃO PARA O APLICATIVO BACKUP

Figura 81: Configuração de cores e *layout* do display do módulo

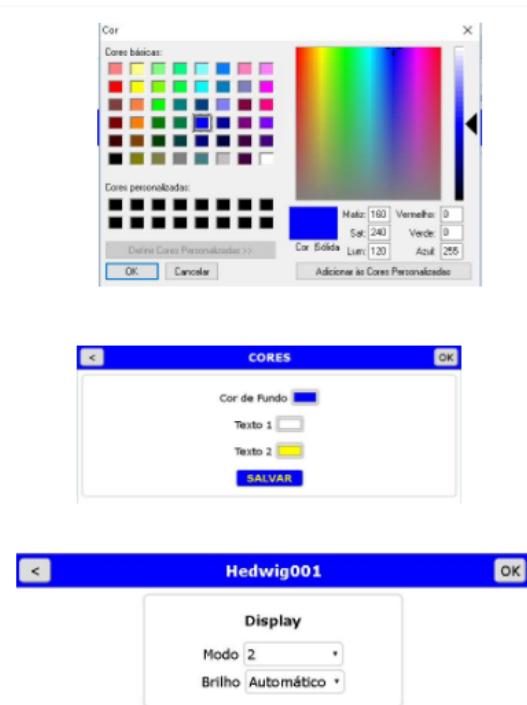


Figura 82: Configurações de alertas, alarmes, relés, log e menu de ferramentas do Aplicativo Backup



Figura 83: Atualização de firmware, menu de configurações avançadas e DHT



Figura 84: Configurações de radiofrequênci, Blynk e Wi-Fi

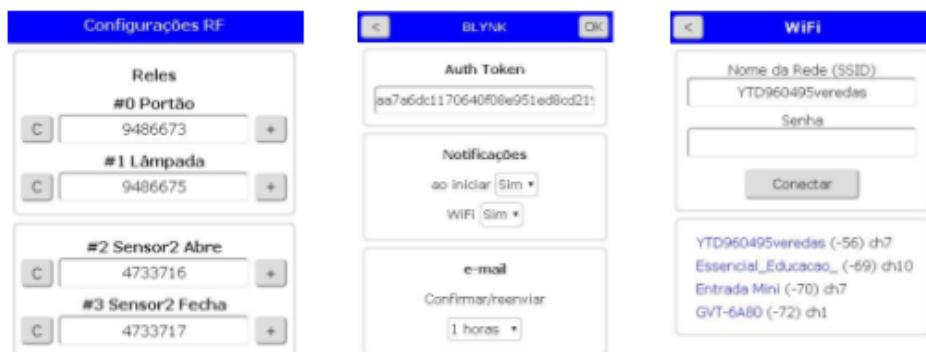


Figura 85: Configurações de nome e ponto de acesso Wi-Fi



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

Figura 86: Resumo do Módulo do Aquário

Módulo	Aquário
Data Início	10/09/2017
Data Fim	29/10/2017
Disponibilidade	Sensores
Total de reinícios*	34
# Updates de Firmware	18
# Erros ping local	87
# Tentativas de conexões WiFi	42
# Desconexões WiFi	15
# Medidas Free Heap (memória livre)	363
Memória livre média (bytes)	30,278
# Tentativas Conexões NTP	60
% de falha NTP	0.00%
# Tentativas Conexões myIP	227
% de falha myIP	60.35%
Tempo Médio para Conexão myIP (segundos)	2.76
# Tentativas Conexões Blynk	74
% de falha Blynk	50.00%
Tempo Médio para Conexão Blynk (segundos)	0.56
*Exclui updates de firmware	

Figura 87: Disponibilidade do Módulo do Aquário – período

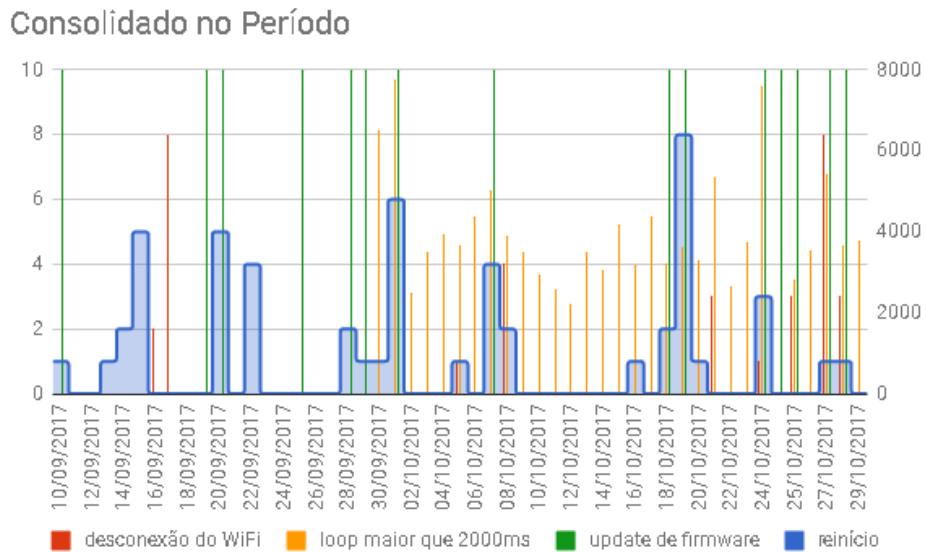


Figura 88: Disponibilidade do Módulo do Aquário – dia

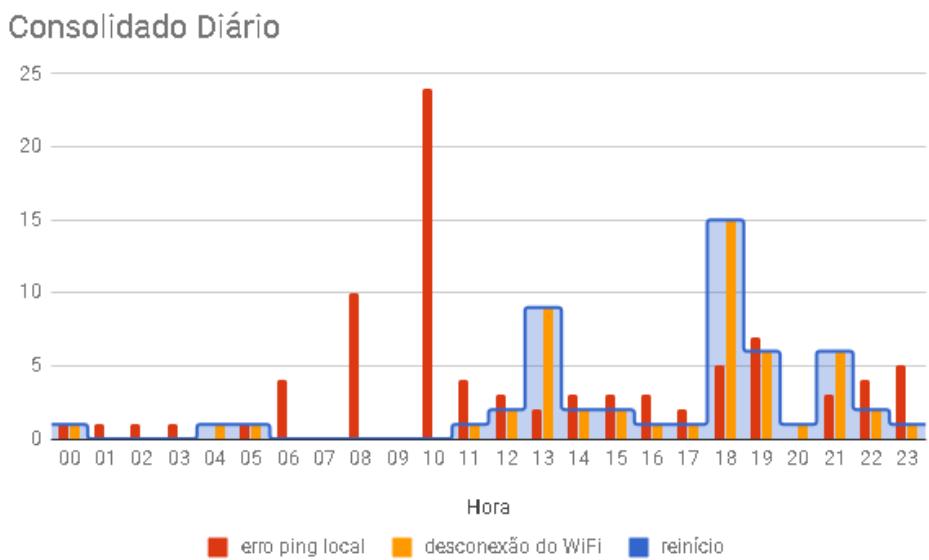


Figura 89: Memória livre do Módulo do Aquário

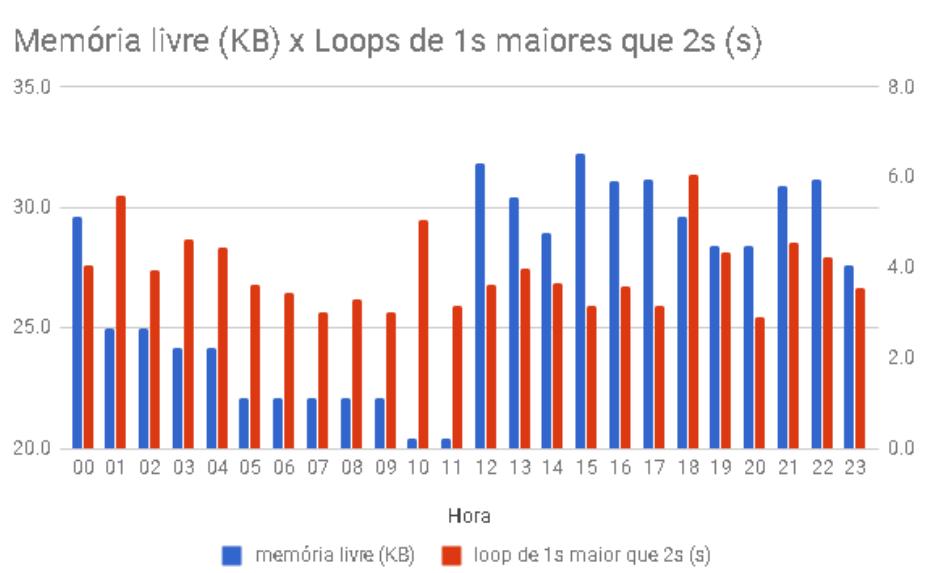


Figura 90: Resumo do Módulo do Corredor

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	Rеле 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	22.91%
Tempo Médio para Conexão Blynk (segundos)	0.82	% Web	# Acionamentos
*Exclui updates de firmware		% RF	11.61%
		% Auto	87.96%
		% Web	202
		37.50%	% Botão Físico
		29.46%	% RF
		21.43%	2.55%
		0.00%	9.49%

Figura 91: Sensores do Módulo do Corredor – período

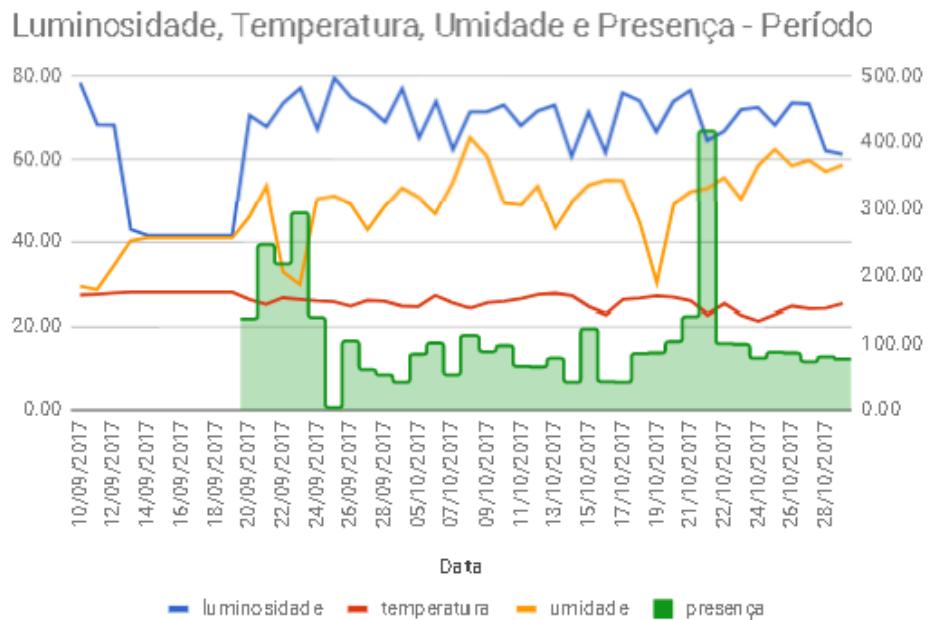


Figura 92: Uso do relé 2 – Módulo do Corredor – dia



Figura 93: Uso do relé 2 – Módulo do Corredor – período

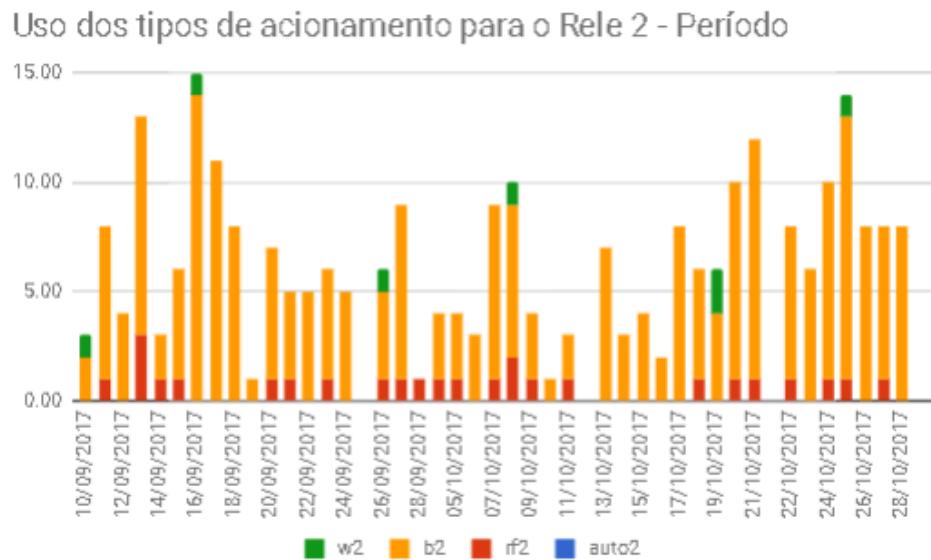


Figura 94: Consolidado diário – Módulo do Corredor

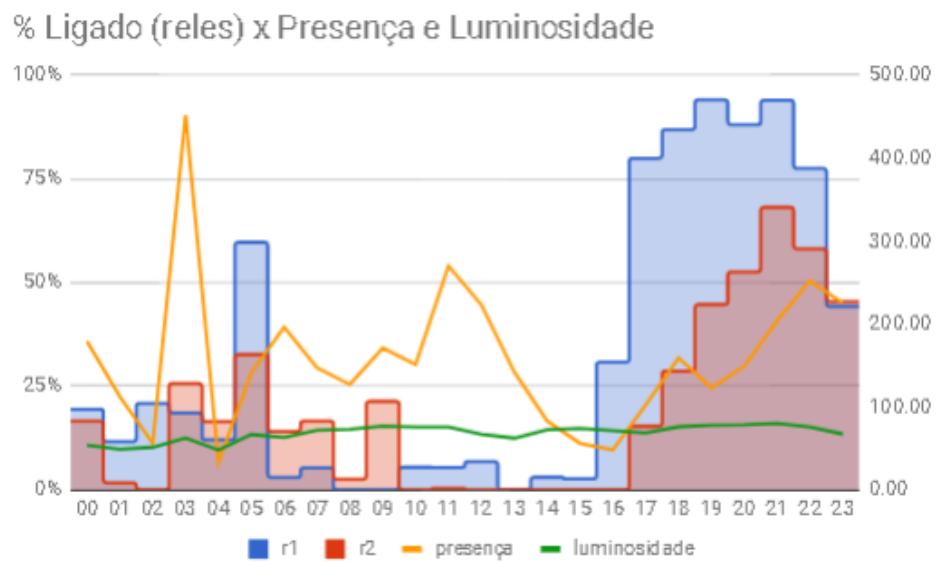


Figura 95: Resumo do Módulo da Lavanderia

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 96: Disponibilidade do Módulo da Lavanderia – dia

Consolidado Diário

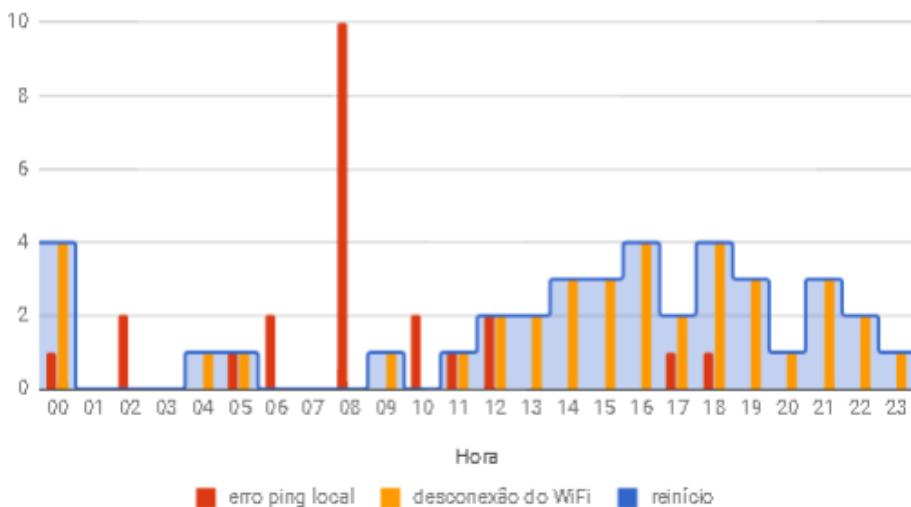


Figura 97: Disponibilidade do Módulo da Lavanderia – período

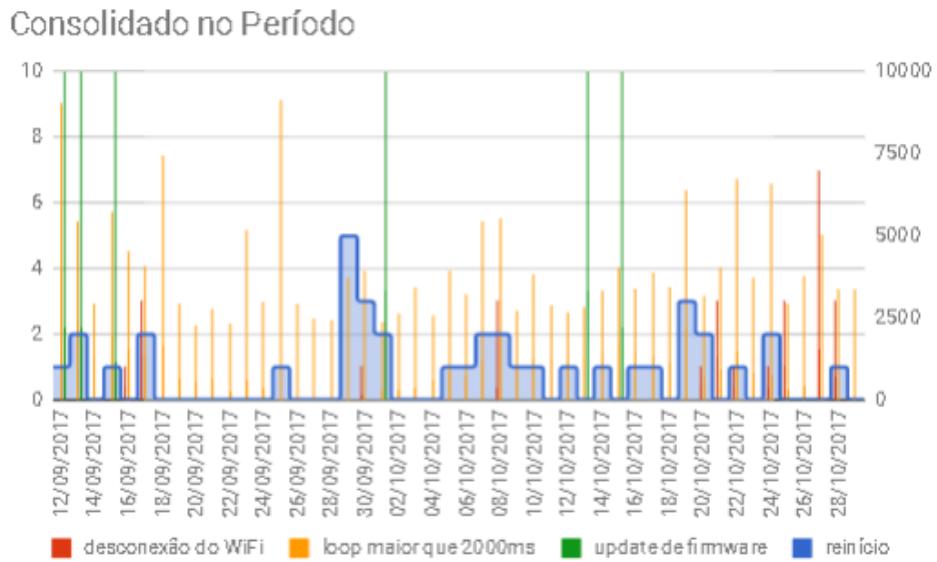


Figura 98: Memória livre do Módulo da Lavanderia

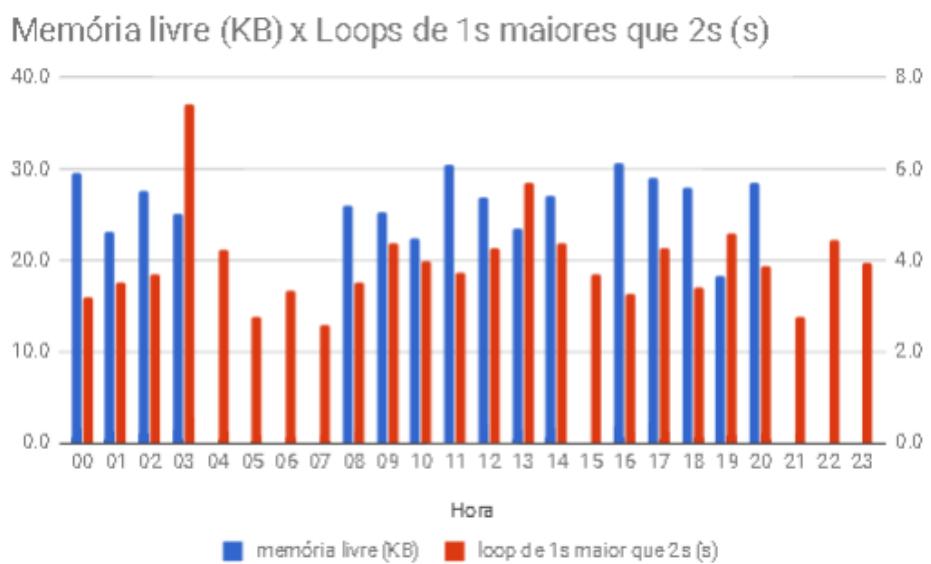


Figura 99: Sensores da Lavanderia – período

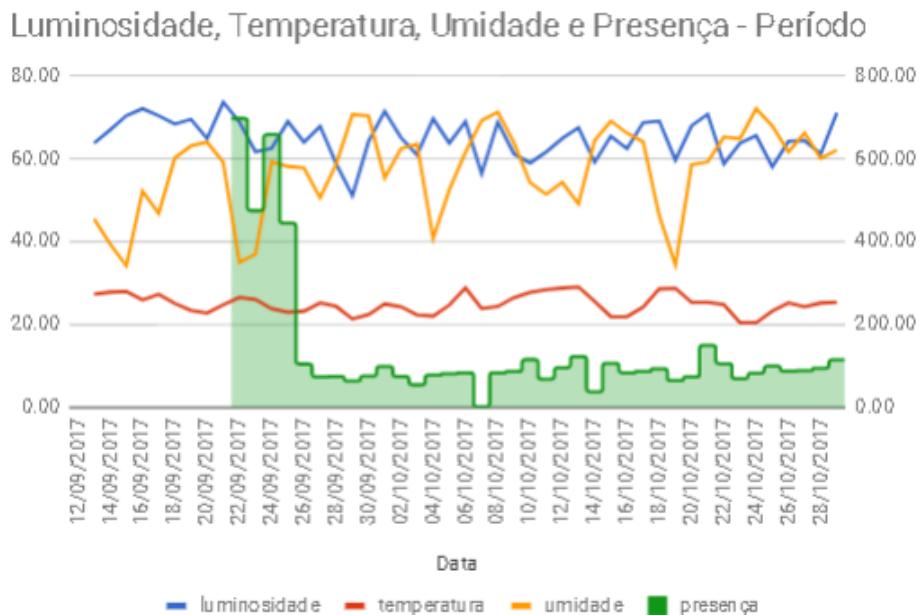


Figura 100: Sensores do Módulo da Lavanderia – dia

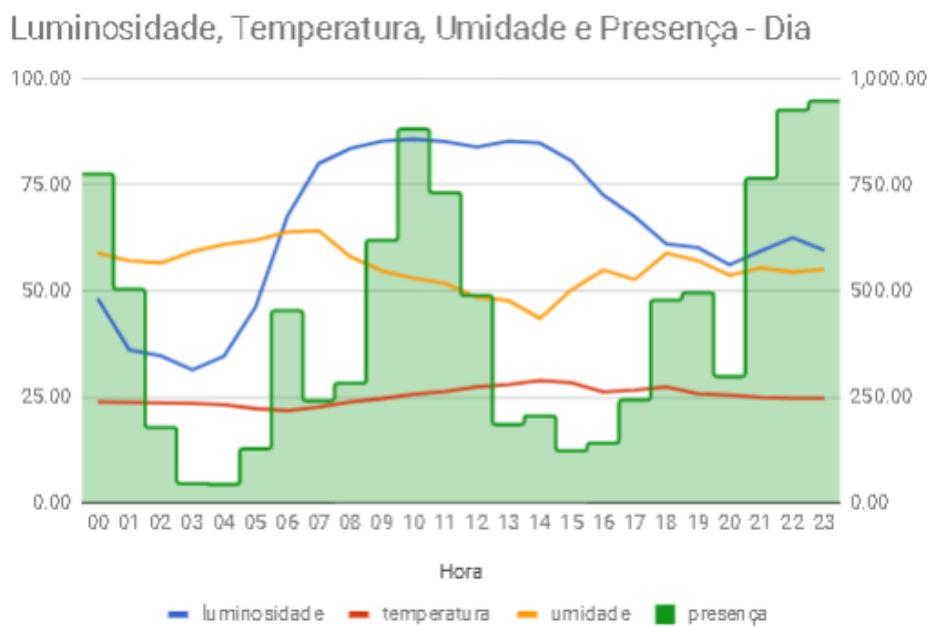


Figura 101: Uso do relé 1 do Módulo da Lavanderia – dia



Figura 102: Uso do relé 1 do Módulo da Lavanderia – período

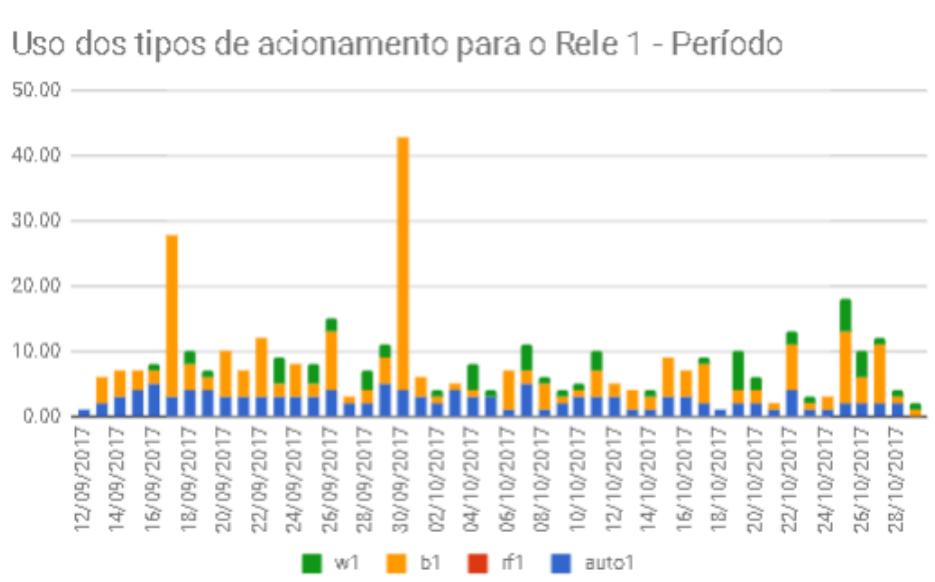


Figura 103: Uso do relé 2 do Módulo da Lavanderia – dia

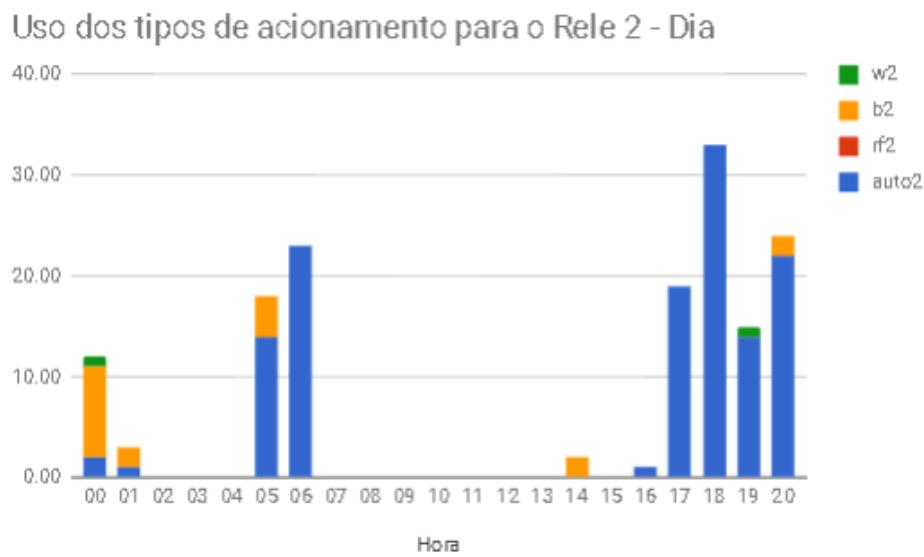


Figura 104: Uso do relé 2 do Módulo da Lavanderia – período

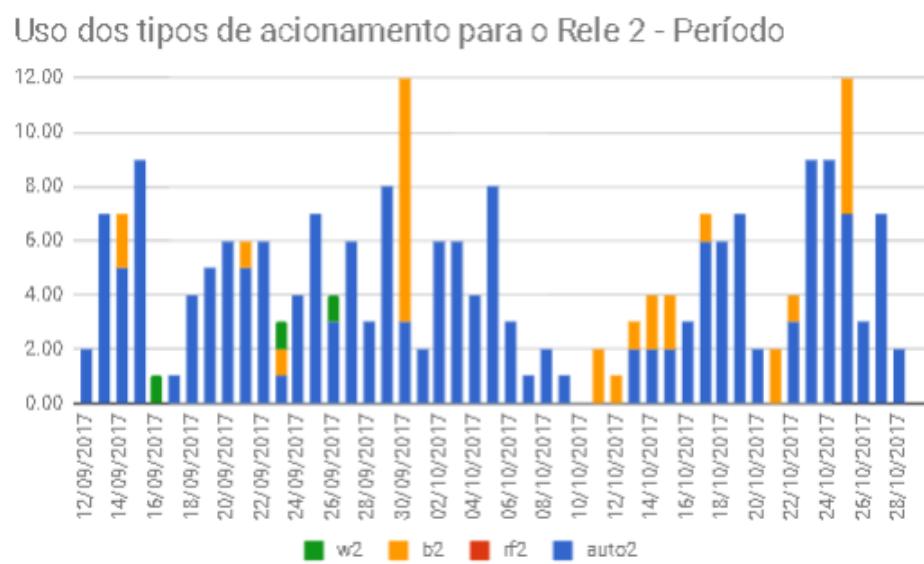


Figura 105: Consolidado diário do Módulo da Lavanderia

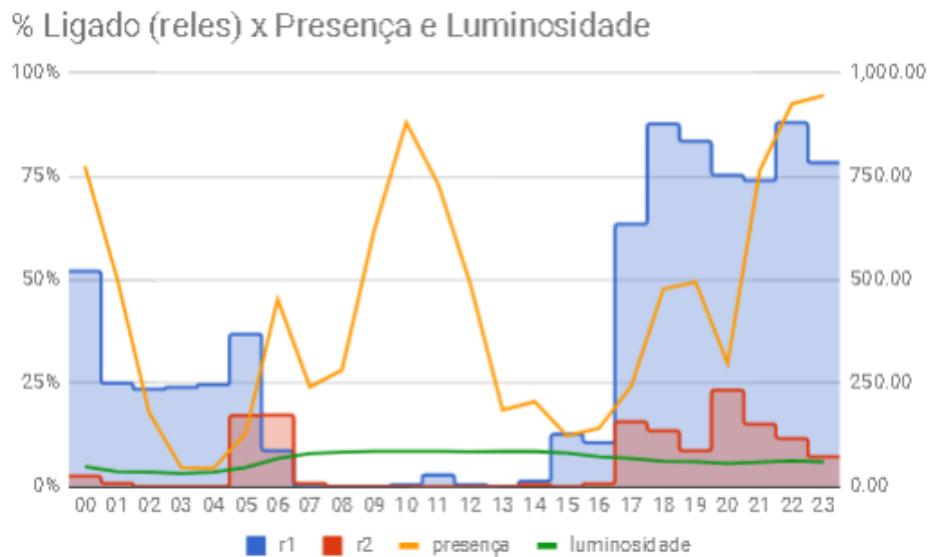


Figura 106: Resumo do Módulo da Sala/Cozinha

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
# Descconexões 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 107: Disponibilidade do Módulo da Sala/Cozinha – período

Consolidado no Período

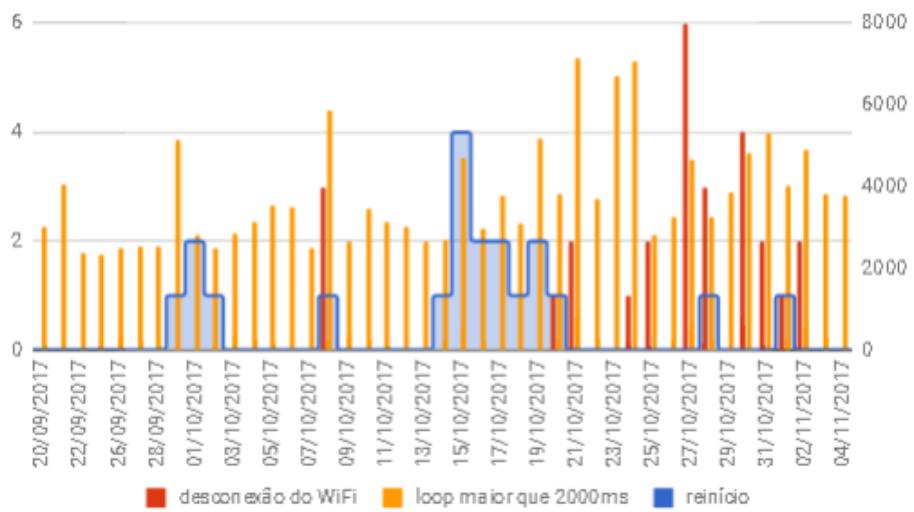


Figura 108: Memória livre do Módulo da Sala/Cozinha

Memória livre (KB) x Loops de 1s maiores que 2s (s)

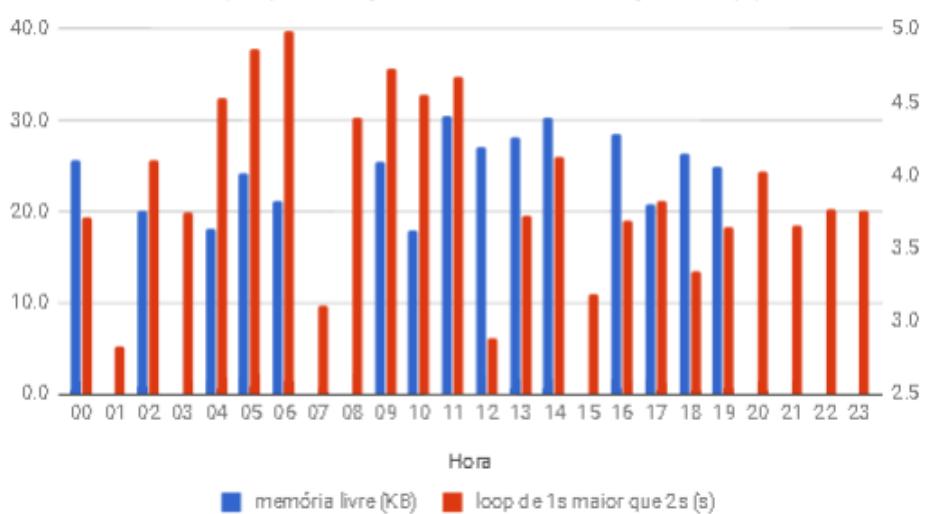


Figura 109: Sensores do Módulo da Sala/Cozinha – dia

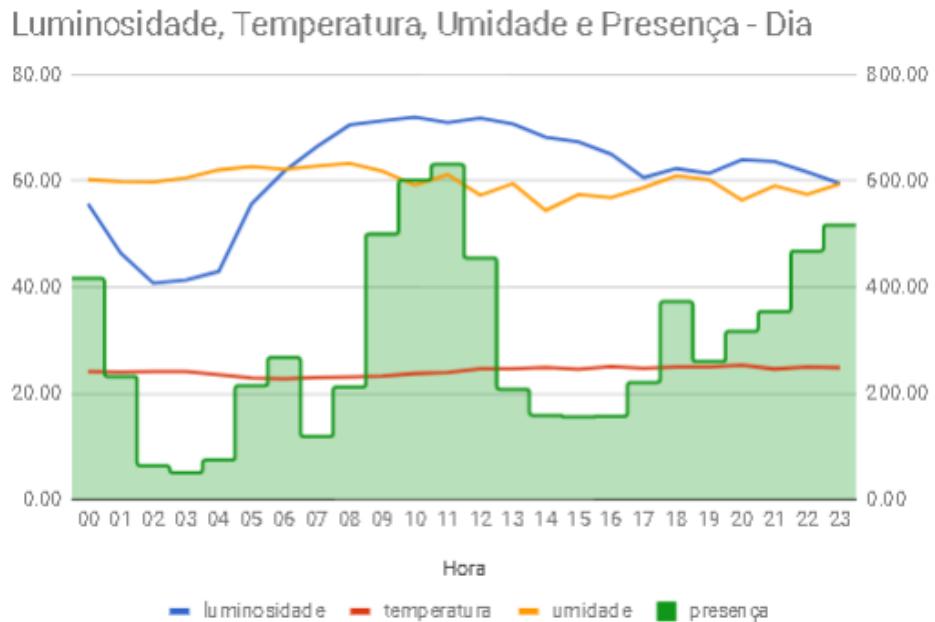


Figura 110: Sensores do Módulo da Sala/Cozinha – período

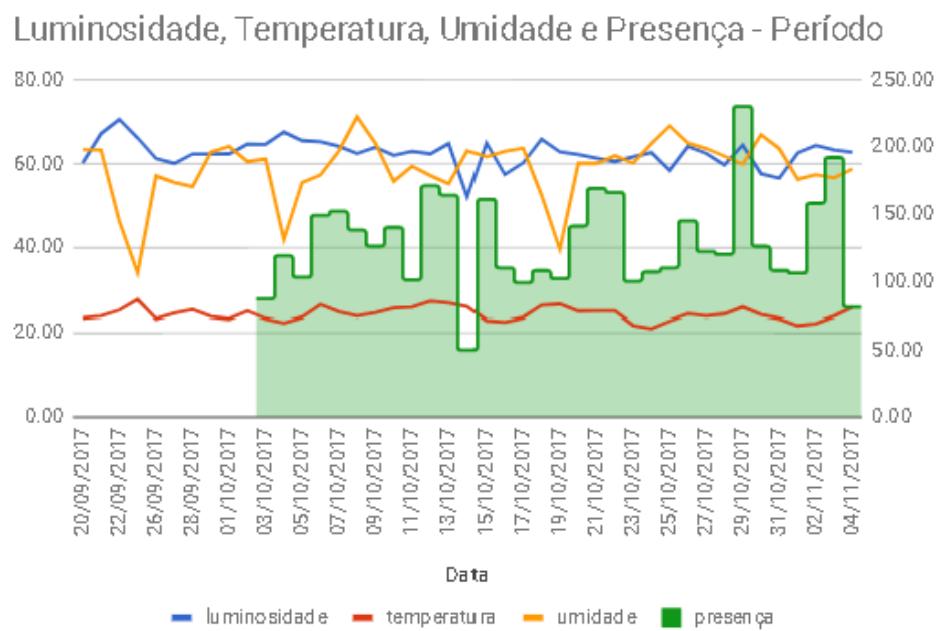


Figura 111: Uso do relé 1 do Módulo da Sala/Cozinha – dia

Uso dos tipos de acionamento para o Rele 1 - Dia



Figura 112: Uso do relé 1 do Módulo da Sala/Cozinha – período

Uso dos tipos de acionamento para o Rele 1 - Período

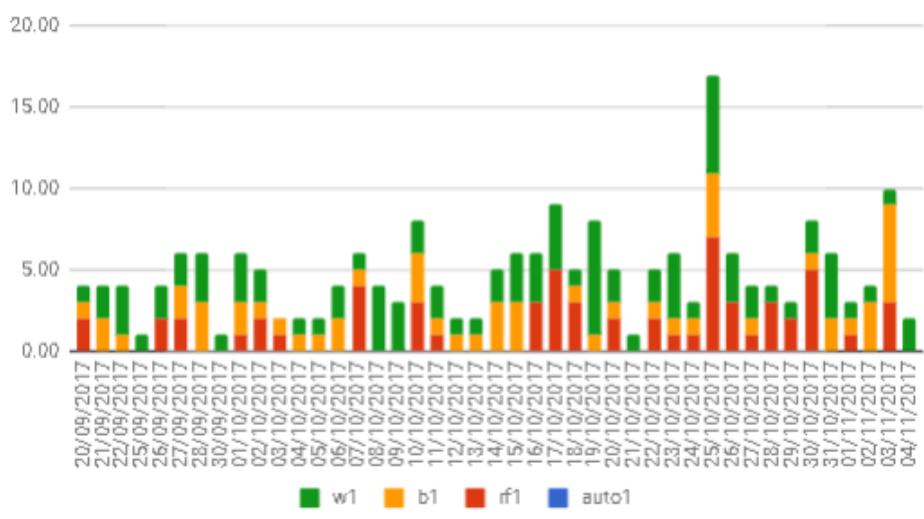


Figura 113: Uso do relé 2 do Módulo da Sala/Cozinha – dia



Figura 114: Uso do relé 2 do Módulo da Sala/Cozinha – período

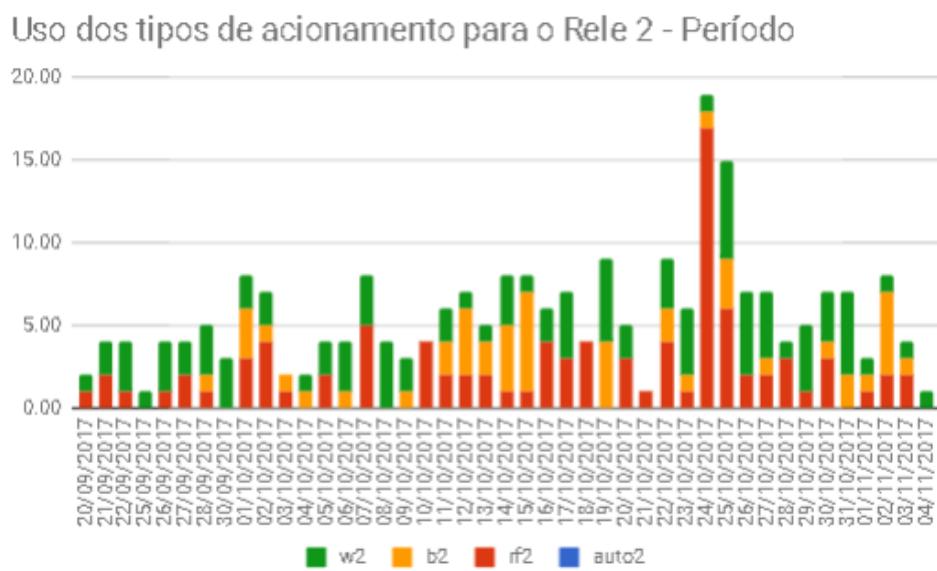


Figura 115: Porta do Módulo de Acesso – dia

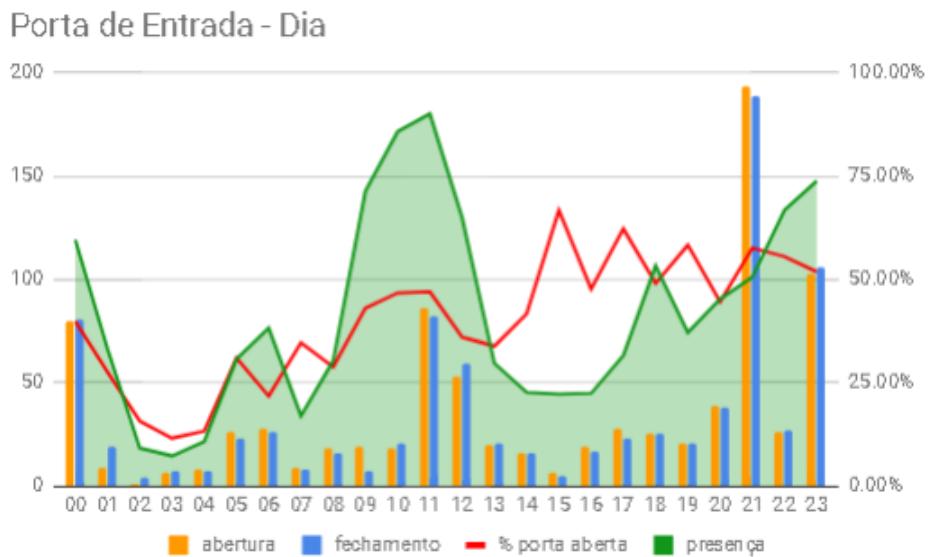


Figura 116: Porta do Módulo de Acesso – período

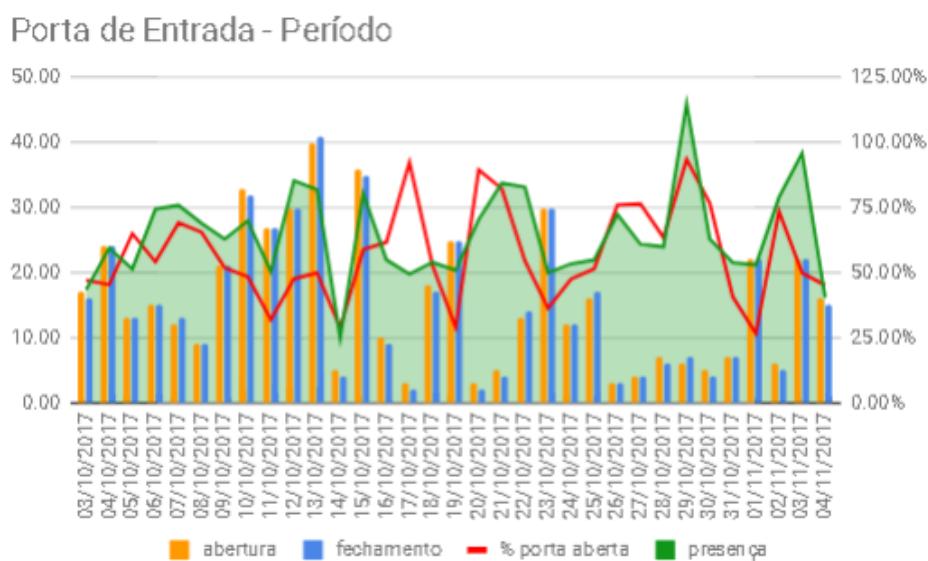


Figura 117: Resumo do Módulo da Entrada

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 118: Consolidado do Módulo da Entrada – período

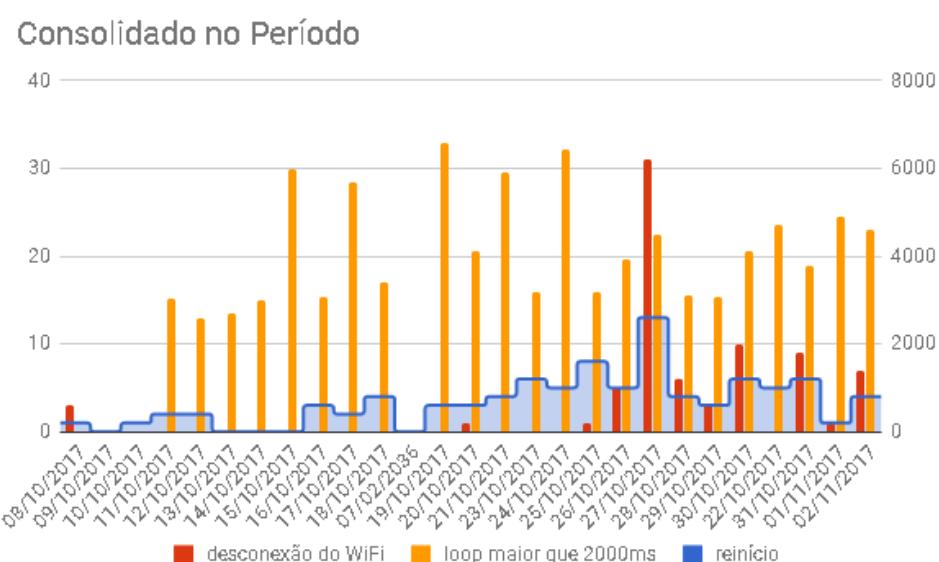


Figura 119: Memória livre do Módulo da Entrada

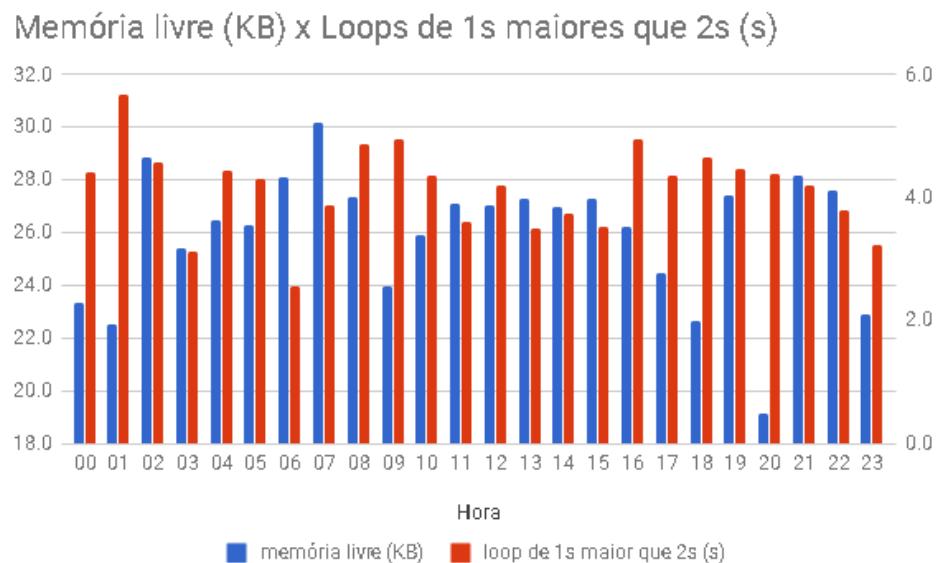


Figura 120: Sensores do Módulo da Entrada – dia

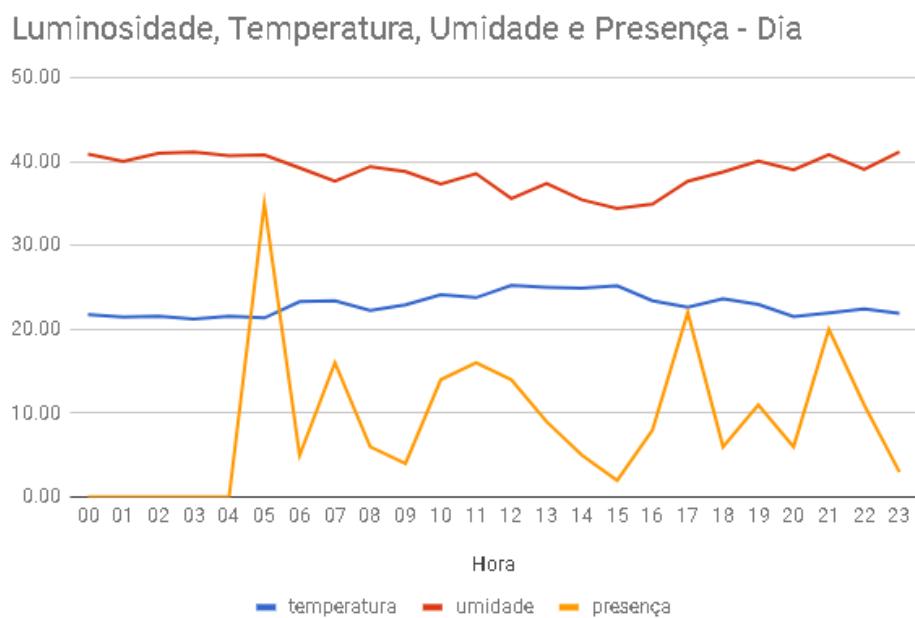


Figura 121: Sensores do Módulo da Entrada – período

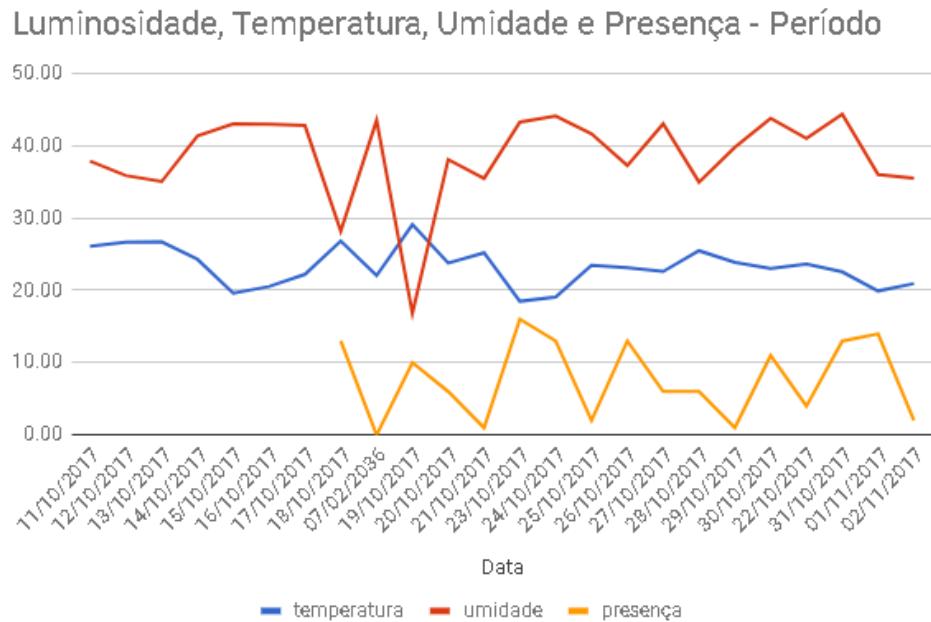


Figura 122: Uso da lâmpada do Módulo da Entrada – dia

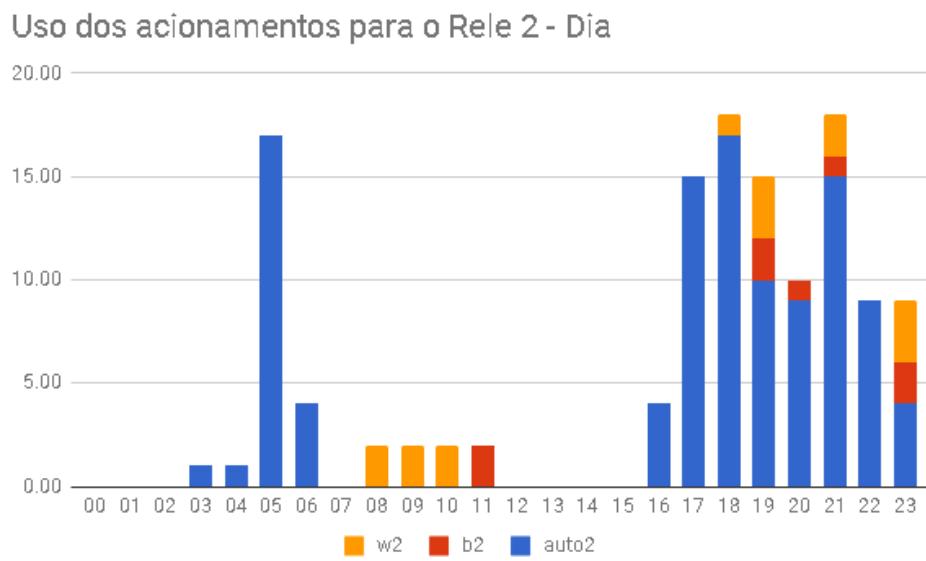


Figura 123: Uso da lâmpada do Módulo da Entrada – período

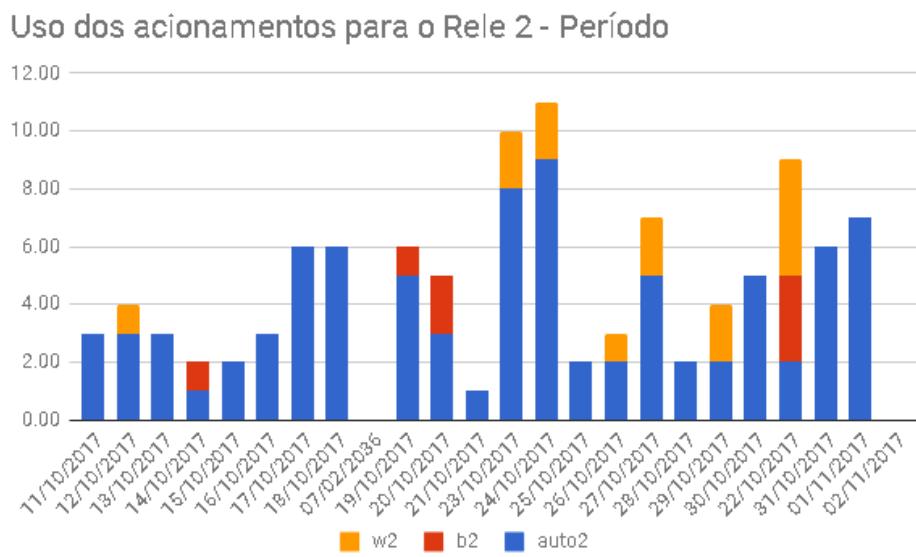


Figura 124: Uso do Módulo da Entrada – dia

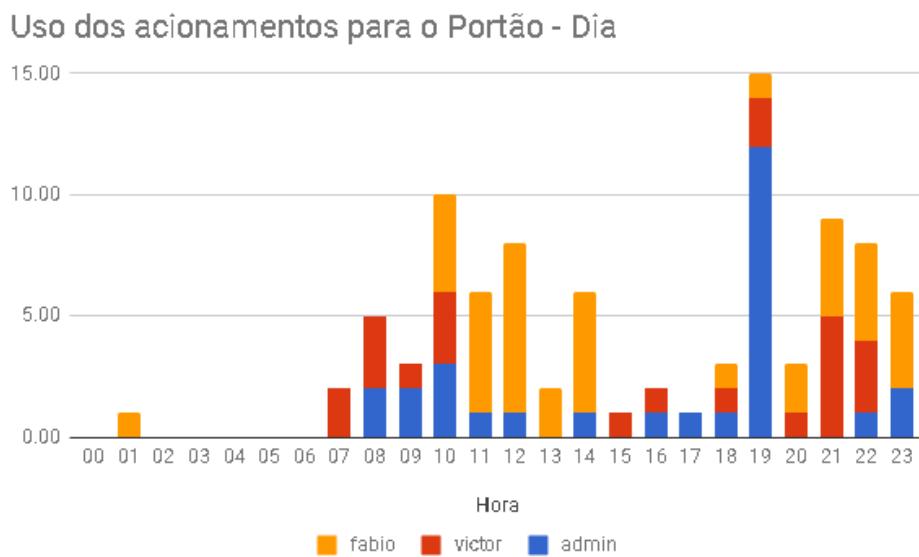


Figura 125: Uso do Módulo da Entrada – período

Uso dos acionamentos para o Portão - Período

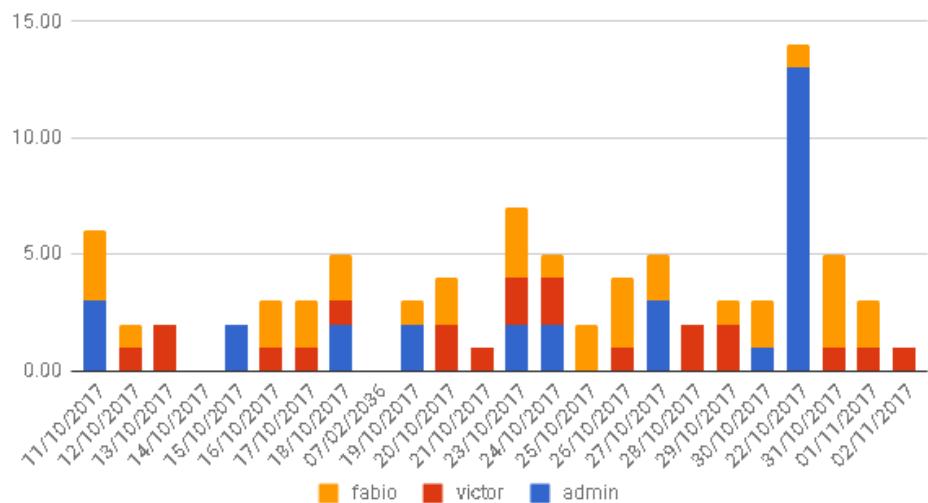
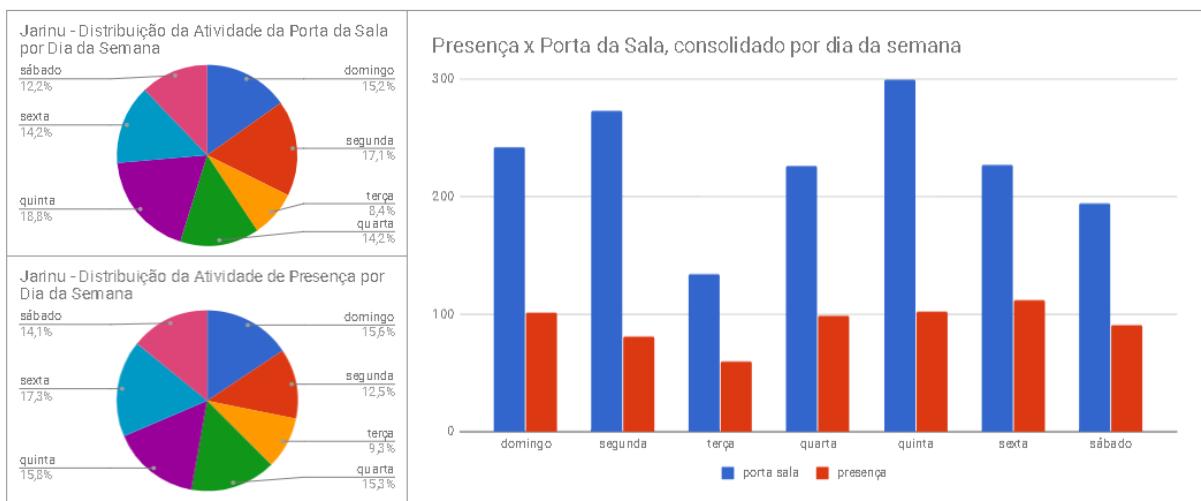


Figura 126: Jarinu – consolidado por dia da semana



ANEXO H – TESTE INTEGRADO (APP+MÓDULO+MORPHEUS)

- **Data:** 15/11/2017
- **Ferramentas usadas:** MQTT fx para acompanhar msgs MQTT do módulo e morpheus, tela e app backup para acompanhar mudanças no módulo e dashboard (<http://hedwig.surge.sh>)
- Reinício: implementado e testado. Mudar enunciado na dashboard para “Emite um sinal que permite simular um travamento do módulo e verificar se o circuito antitravamento age” - a questão é que o circuito antitravamento está sempre ativo, não é bem que ele é ativado quando realizamos o teste. *implementado e testado*
- Auto Reset Test: implementado e testado. Mudar enunciado na dashboard: “Reiniciar módulo por software.” *implementado e testado*
- Configuração de LCD: dashboard ok, msgs enviadas estão corretas, e o módulo responde aos três tipos de tela. Victor ainda deve tratar a parte do backlight; *implementado e testado*
- Sincronização de hora: dashboard confirma para o usuário? (check de timestamp enviado pelo módulo na confirmação próximo da hora do sistema é necessário?); Victor deve chamar procedimento de sincronização com NTP quando recebe a msg *implementado e testado*
- Configurações gerais: dashboard deveria estar mandando msgs de mudança de nome de módulo e relés, para que o app backup reflita os novos nomes? A troca de nome de módulo ocorre corretamente na dashboard, mas por enquanto não está sendo refletida no app backup - *implementado e testado*

- Reduzir tempo para confirmação após receber requisição (demora para acionar e desligar um rele em seguida) *implementado e testado*, melhorado para resposta em 1s;
- Msgs de Comunicação: Victor fazer os procedimentos no módulo *implementado e testado*
- Config RF: msgs ok (requisição pela dashboard e confirmação de recebimento de requisição pelo módulo), mas falta Victor chamar procedimento no módulo *implementado e testado*
- Config de acionamento de reles: incluir na dashboard do basic (possivelmente também parte de alarme) a fazer
- Logout: aumentar timeout de sessão *implementado e testado*
- Config de Comunicação: 8 caracteres na senha do módulo, deixar sempre setado “Automático” no modo do ponto de acesso do módulo e 192.168.0.20 no ip local fixo *implementado e testado*
- Msgs específicas módulo de entrada: Victor implementar *implementado e testado*