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 obtenção do Título de Engenheiro Eletricista com ênfase em Computação.

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 obtenção 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

RESUMO

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

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

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

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

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

ABSTRACT

The increasing development of the Internet of Things (IoT) technologies 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 this scenario, this work aims to provide a complete system for Smart Houses. It explores communication technologies and connectivity between devices, creating an accessible and expandable platform for residency automation - all of that with low production cost.

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

LISTA DE TABELAS

LISTA DE SIGLAS

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

SUMÁRIO

1 Introdução

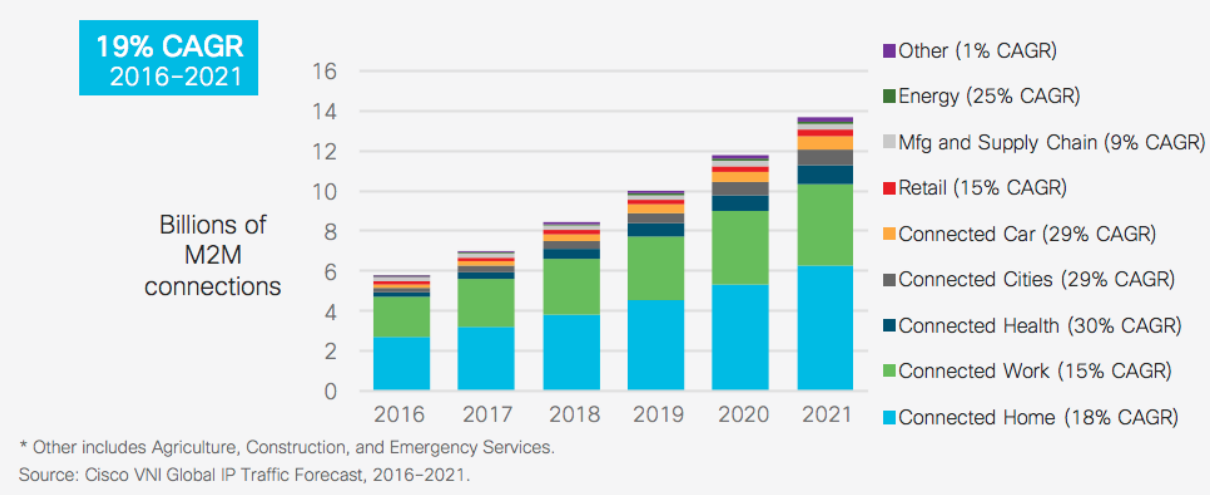
1.1 Motivação

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

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

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

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



Fonte: (CISCO, 2017)

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

1.2 Projeto Hedwig

1.2.1 Objetivo

A contribuição do projeto para o avanço das tecnologias de *Internet das Coisas* fundamenta-se na criação de um sistema com arquitetura modularizada, e em camadas, com funcionalidades locais e de nuvem, provendo uma *API* que permita seu acesso por diversos clientes - como *websites* ou aplicativos para *smartphones* - que seja capaz de monitorar e agir em diversos módulos presentes na residência do usuário final do sistema. O projeto irá disponibilizar módulos físicos, prontos para serem instalados e configurados na residência, sem que seja necessário 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, foram previstos três níveis de operação para o sistema: Online, Local e Offline, os quais dispõem de diferentes requisitos de funcionalidades, para garantir a 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, em 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 módulos 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 exercita um conjunto de tarefas específicas (de transporte, análise, tomada de ações, etc), e seus efeitos são traduzidos em entradas para o nível seguinte.

Machine Learning

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

Segurança

A proteção da privacidade do usuário é tão importante, ou talvez mais, quanto a proteção física da casa. Assim, necessita-se que o sistema seja seja seguro, e que o fluxo de dados trocados entre as partes ocorra em canais protegidos. Foram utilizadas tecnologias de criptografia assimétrica para comunicação entre servidor local e serviços de nuvem, juntamente com conexão por *WebSocket*; a autenticação e autorização de usuários são realizados por meio de *JSON Web Tokens* e é feito uso de canais *Publisher/Subscriber* protegidos para troca de mensagens.

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, e 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 um 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 *Machine Learning* incrementa o sistema, permitindo trazer facilidades e promover maior adaptação. O sistema torna-se capaz de aprender com *feedbacks* do usuário, seja pelo monitoramento dos módulos ou por respostas dadas por meio do aplicativo, e pode atuar em questões de segurança (*safety*), personalizações e mesmo em possíveis sugestões de produtos relacionados à rotina do cliente.

2 Projetos Relacionados

2.1 Sistemas Existentes no Mercado

2.1.1 Sistemas Comerciais

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

O Amazon Echo[1] consiste em um *smart speaker* (alto-falante inteligente) conectado ao assistente pessoal Alexa, também da Amazon, que é capaz de entender comandos de voz. Inicialmente, funcionava como uma maneira de encomendar produtos por voz. Atualmente, além de funcionar como assistente pessoal, também é capaz de controlar diversos *smart devices* da casa, funcionando como um *hub* de automação residencial. Uma limitação deste produto é que funciona apenas com uma conexão *wireless* de Internet, não sendo capaz de operar em nenhum nível sem a mesma.

Algumas características interessantes do Alexa são que desenvolvedores são capazes de adicionar novas *skills* (habilidades) por meio de documentação da API que está pública 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 consiga conectar-se a ele pela Internet.

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

[1] http://www.amazon.com/oc/echo/
[2] https://madeby.google.com/home/

para o estabelecimento de parcerias que permitam integração com eles, de forma que o Google Home também 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 online. Alguns desses projetos analisados para o desenvolvimento do nosso projeto Hedwig foram o OpenHAB e o Home Assistant.

O OpenHAB[3] possui como objetivo principal o estabelecimento de uma plataforma em software de integração que seja capaz de solucionar o problema atual de que os diversos *devices* de uma residência não são capazes de se comunicar devido à falta de uma linguagem comum com a qual eles possam estabelecer tal comunicação. Por ser independente de hardware específico, é extremamente flexível e personalizável, porém isso implica em certa complexidade para o usuário, no momento de sua instalação. Apresenta interface para o usuário em cliente web e aplicativos nativo para iOS e Android.

O Home Assistant[4] é uma plataforma de automação residencial capaz de controlar e monitorar os diversos devices 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 a dificuldade de que é necessário que o usuário possua conhecimentos técnicos para utilizá-los.

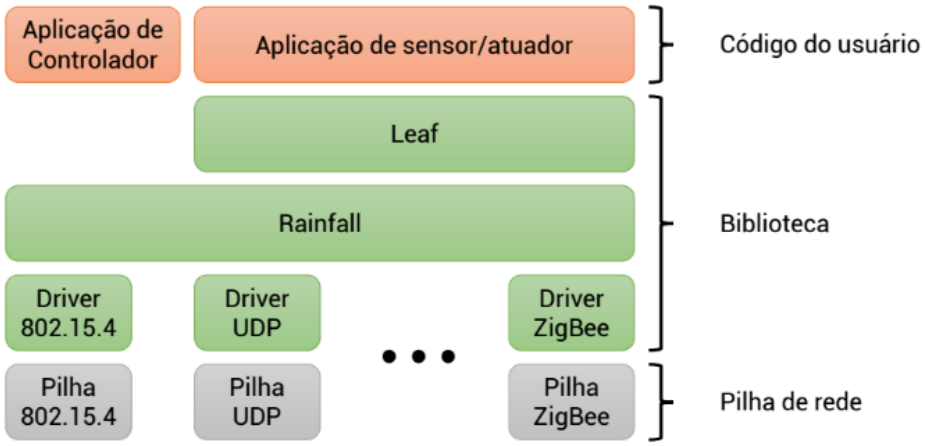
2.2 Projeto HomeSky

O Projeto HomeSky (MURAMATSU; RODRIGUES; GALLEGO, 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 protocolo Rainfall, um protocolo em código aberto a nível de aplicação para ser usado

[3] http://www.openhab.org/
[4] https://github.com/home-assistant/home-assistant

na coordenação de uma rede de sensores. Isso permitiria aos desenvolvedores ter uma maior flexibilidade 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, deixando possíveis implementações de mecanismos de reconhecimento de entrega e retransmissão a cargo do 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.

O sistema Hedwig será uma evolução natural do Projeto HomeSky, buscando aperfeiçoar seu protocolo e arquitetura quanto à robustez e expandir a aplicação de aprendizado de máquina, além de viabilizar seu conceito, desenvolvendo soluções voltadas ao mercado brasileiro.

3 Especificação

3.1 Componentes

3.2 Stakeholders

Com as funcionalidades e os módulos apresentados, podemos destacar os seguintes grupos dentre os potenciais consumidores:

- Pessoas que moram sozinhas e suas famílias, que podem estar interessadas em monitoramento;
- Pessoas que desejam comodidade de controlar seus aparelhos numa interface única, pelo celular, e/ou conforto maior em casa;
- Pessoas preocupadas com o consumo de água e energia elétrica.

Considerando o Censo de 2010 (IBGE, 2010), podemos estimar grosseiramente as classes de consumidores para a cidade de São Paulo:

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

3.3 Requisitos

3.3.1 Requisitos Funcionais

- O sistema deve permitir o monitoramento de aparelhos do dia a dia, dentro de uma residência, em módulos independentes;
- O sistema deve ser capaz de enviar notificações aos usuários, por meio de um aplicativo web ou *mobile*;
- O sistema deve poder ser personalizável pelo usuário, o qual pode adquirir novos módulos ou retirar algum já existente;
- O sistema deve ser capaz de aprender a respeito de cada usuário, utilizando conceitos de Machine Learning. O aprendizado de máquina é responsável por detectar padrões no comportamento do usuário, e a partir disso, sugerir ao usuário ações a serem tomadas automicamente pelo sistema;
- O sistema deve manter backup de dados do controlador local na nuvem;
- O sistema deve permitir a usuários se cadastrarem no mesmo, pela plataforma que melhor lhe convier;
- O usuário poderá cadastrar sua casa na plataforma, podendo ter uma ou mais casas cadastradas;
- O usuário poderá cadastrar os módulos dentro de uma casa, sendo que uma casa pode ter vários módulos, e cada módulo só poderá existir em uma casa;
- O usuário pode efetuar as operações de remoção e modificação nos seus módulos e casas;
- O sistema deve possuir uma função de reset de fácil utilização.

3.3.2 Requisitos Não-Funcionais

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

- Os módulos 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 ampliação e a inserção de novos módulos, com outras funcionalidades. Haverá validação com o desligamento de um módulo e verificação do comportamento dos demais;

- O sistema deve garantir segurança dos dados por meio de protocolo de comunicação seguro, tanto para o controle de acesso à API por usuários autenticados quanto para impedir que dados sejam interceptados em sua transmissão;
- O banco de dados deve possuir acesso restrito e estar hospedado em servidor de alta segurança;
- O sistema deve ser robusto, de modo a continuar operando, mesmo com menor nível de funcionalidades, quando da ocorrência de falhas na comunicação com a nuvem (indisponibilidade parcial devido a problemas com os servidores remotos, ou total com perda da conexão com a Internet) ou falhas na rede local (indisponibilidade da conexão com a rede local). Também deve se recuperar em caso de travamento total do módulo e continuar funcionando em caso de DoS Local. Para validação, haverão testes de indisponibilidade de servidor, conexão com a internet, rede local e DoS local, e observação da continuidade de serviço de atuação na iluminação da casa e abertura do portão em menos de 10 minutos;
- O sistema deve apresentar disponibilidade de 99,9% - cerca de 8 horas de indisponibilidade por ano -, não levando em consideração problemas com a conexão de internet da residência;
- O sistema deve ser escalável a até 10 mil usuários, sem perdas de desempenho consideráveis, ou aumento na latência para as requisições serem atendidas;
- O sistema deve possuir instalação intuitiva e simplificada.

3.3.3 Requisitos por Nível de Conectividade

Nas Tabela 1 são descritos os requisitos, bem como a técnica de avaliação utilizada para demonstrá-lo. A Tabela 2 mostra tais requisitos de acordo com o nível de funcionalidade.

Tabela 1: Descrição de requisitos

ID	Requisito	Avaliação	Descrição
1	Automação Residencial	Demonstração das funcionalidades listadas	Controle por App ou no módulo
2	Monitoramento Residencial	Demonstração das funcionalidades listadas	Monitoramento no App e displays dos módulos
3	Modularidade	Funcionamento de módulo "stand-alone"	Soluções modulares
4	Escalabilidade para n residências	Testes de carga, acessos simultâneos	Suporta carga, acessos simultâneos
5	Machine Learning	Tratamento de séries de dados reais	Análise dos dados para levantamento de rotinas e reconhecimento de padrões
6	Backup de dados	Funcionalidade de Backup de dados no controlador local	-
7	Tolerância a falhas	Estatísticas de keep alive, ping local, ping internet de módulo base com e sem as melhorias	Manter funcionalidade Online, Local e Offline
8	Segurança da informação	Protocolos seguros, controle de acesso	Protocolos seguros, controle de acesso

Tabela 2: Níveis de funcionalidades

ID	Online	Local	Offline
1	Usar parâmetros para controle inteligente de lâmpada e despertador (Quarto)	Configurar lâmpada automática (Quarto) Configurar despertador (Quarto)	Ligar e desligar lâmpada (Quarto) - Módulo
2	Estado do portão (Acesso) Temperatura, Umidade (Todos) Presença (Quarto)	Estado do portão (Acesso) Temperatura, Umidade (Todos) Presença (Quarto)	Desativar despertador (Quarto) - Módulo Destravar porta (Acesso) - App
3	-	Inserção automática/reconhecimento (Todos)	Temperatura, Umidade (Todos) - Módulo Presença (Quarto) - Módulo
4	-	-	-
5	Derivações de Regras Perfil Esperado de Comportamento	Alerta de Portão Aberto (Acesso)	-
6	Redundância na nuvem	Armazenamento de dados temporariamente num cartão SD, no controlador local	Alerta de Portão Aberto (Acesso) - Módulo
7	-	Notificações e alertas no App (Monitoramento estado módulos)	Circuito "Keep Alive" - Hard Reset e Soft Reset
8	Autenticação para as funcionalidades descritas nesta coluna	Autenticação para as funcionalidades descritas nesta coluna	Autenticação para as funcionalidades descritas nesta coluna - Módulo

4 Arquitetura

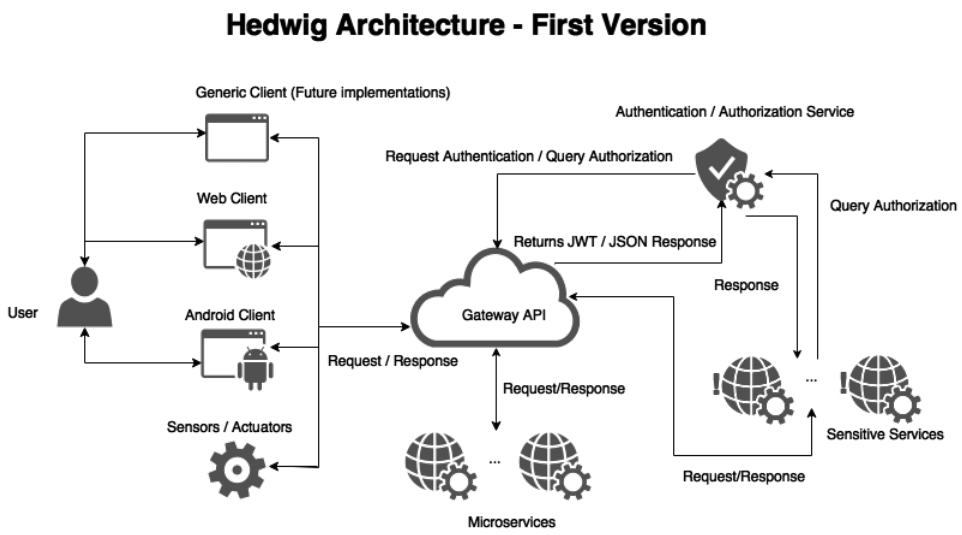
4.1 Visão geral

4.2 Evolução arquitetural

O processo de escolha para a arquitetura utilizada foi iterativo, e foram analisados os pontos fracos e as vantagens de cada nova sugestão.

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 fundamentalmente desenvolvida assim, também é possível utilizar quantas tecnologias forem necessárias ou desejáveis para cada um dos serviços sem efeitos colaterais nos outros, ou seja, transparentemente. Por outro lado, cria-se grande uma complexidade na integração entre todos os serviços disponíveis, que pode ser gerenciada por técnicas conhecidas e também explicadas aqui, como a coreografia e a orquestração. No entanto, há um aumento do *overhead* para a comunicação, e os serviços do Hedwig necessitam de um meio rápido e robusto, que implemente qualidade de serviço para padrões diferentes de mensagens. Foi proposto um *gateway* para os serviços da nuvem, por onde passaria toda a comunicação com a casa. A inserção do *gateway*, no entanto, cria um ponto único de falha.

Figura 4: Primeira versão da arquitetura do projeto Hedwig

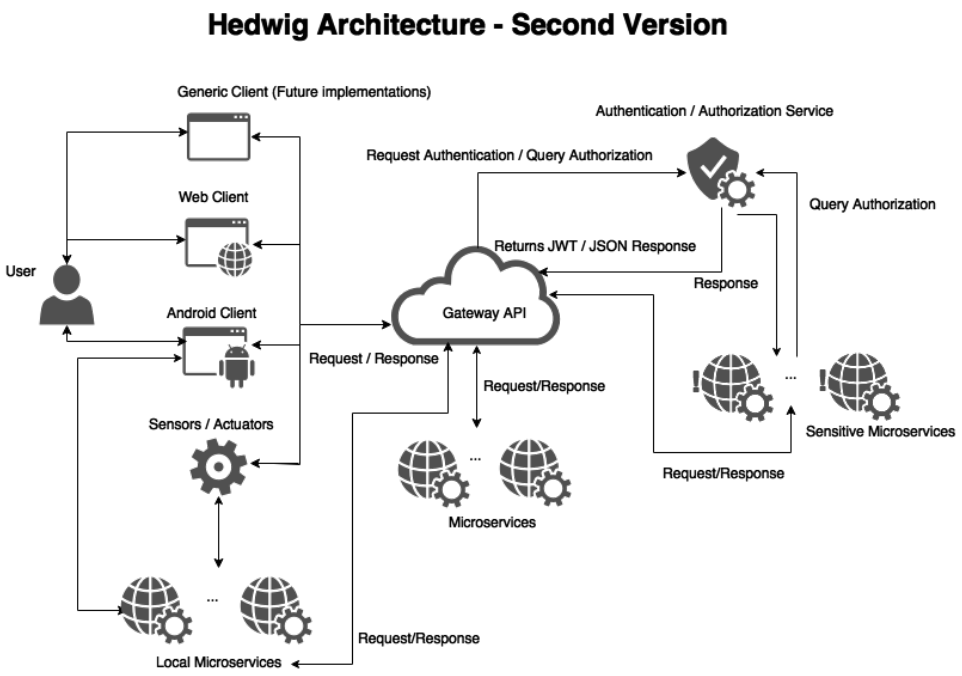


É possível observar que alguns microsserviços são classificados como sensitivos, os quais dependem de nova consulta ao serviço de autenticação e autorização para garantir a segurança. Esses serviços são todos aqueles responsáveis por tomar uma ação em relação à casa que envolva riscos. Os microsserviços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

Quando uma requisição chega à nuvem, ela deve ser autenticada, e caso passe nos critérios de autenticação e autorização, é retornado um JWT (*JSON Web Token*), necessário para os passos seguintes. O JWT é discutido aqui, na seção MARCAR SEÇÃO AQUI.

De extrema importância, e não cobertos pela arquitetura anterior, são os requisitos de disponibilidade do projeto. 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 não estarem online no momento necessário, também não seriam alcançados requisitos mais fortes de disponibilidade. Contudo, é uma versão que chega mais próxima de obedecer às necessidades do projeto.

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

Assim, começamos o desenvolvimento de um modelo arquitetural modularizado, onde

cada módulo teria inteligência para realizar as tarefas necessárias e, ao mesmo tempo, pode enviar dados à nuvem e ser avisado quando deve realizar uma tarefa. Além disso, no aspecto comercial, módulos inteiros poderiam ser vendidos, substituídos e aumentados.

A arquitetura escolhida 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, inteligência e autonomia para realizar todas as atividades necessárias, e com comunicação a um servidor local, que realizará, por último, a comunicação 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 uma eventualidade de a comunicação entre o servidor local e a nuvem ser perdida, os aplicativos web ou *mobile*, podem se comunicar diretamente com o servidor local da casa, para acessar uma quantidade mais restrita e essencial de ações - como, por exemplo, a liberação de acesso à casa. Nessa situação, o servidor local armazena as mensagens, que serão enviadas ao servidor remoto posteriormente. Essas mensagens, no caso, seriam de dados, advindas de sensores em módulos. Como não há urgência para o processamento de tais dados, os quais serão utilizados para análise de comportamento e Machine Learning, não há prejuízo com o eventual envio tardio. O único prejuízo em tal situação seria no caso de o usuário não estar conectado à sua rede local, situação na qual não seria possível visualizar por meio de aplicativo web ou *mobile* informações sobre o status dos sensores e atuadores em seus módulos.

Além disso, no caso de perda de comunicação tanto com a nuvem como com o servidor local, os aplicativos poderão se comunicar diretamente com os módulos para terem acesso aos serviços de extrema importância.

Por ter sido escolhida, essa arquitetura será extensivamente detalhada e discutida aqui, com seus benefícios e limitações.

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 será intermediada por um servidor local, que rodará 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, de modo que, para contornarmos essa limitação, utilizaremos módulos ESP8266 para transmissão sem fio por meio de Wi-Fi. Esses módulos serão responsáveis pela transmissão das informações recebidas para o servidor local. Toda a arquitetura para essa transmissão será detalhada mais à frente. Os outros módulos a serem utilizados, como sensores DHT11, LM555, etc. podem ser vistos em uma lista completa no Anexo X.

Em geral, esses 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 e/ou trabalhando como pontos de acesso. Para casos de falha de conexão, há um algoritmo de novas tentativas com tempos progressivamente maiores conforme as falhas ocorrerem, que busca deixar o módulo disponível para outras funções enquanto o serviço não está disponível. Para evitar o travamento, um sinal de *keep alive* é monitorado, e um circuito anti-travamento deve ativar um *hard reset* (reset por hardware), ou então uma rotina de *soft reset* deve ser acionada. No entanto, observe que a segunda alternativa é a mais fácil de implementar, mas a 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 conexão Wi-Fi 802.11 integrada (ESPRESSIF, 2015). Pode ser programado usando a Arduino IDE, já muito 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 utilizaremos em modo *StandAlone* como principal processador e responsável pela conexão dos diferentes módulos de automação. Suas duas principais plataformas de desenvolvimento são Wemos[1] e NodeMCU[2]. O projeto utilizará o Wemos D1 Mini, versão compacta da Wemos D1 R2.

Possui um modo de operação de baixa potência (*sleep mode*) em que o consumo de bateria fica muito menor - em contrapartida, o nível de funcionalidades fica limitado.

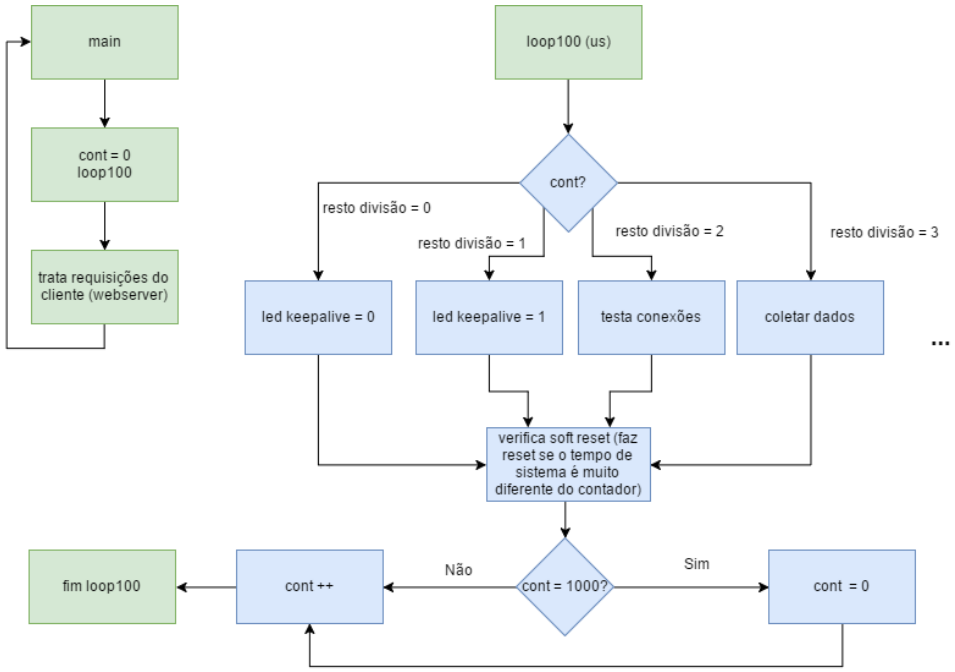
[1] https://www.wemos.cc/
[2] http://nodemcu.com/

Podemos usar 7 portas de E/S digitais e uma porta de entrada analógica. Duas portas são inutilizáveis, pois são usadas para programação e outras tarefas do sistema integrado do ESP8266. Alternativas para extensão de portas são: utilizar três níveis de sinal análogico para detectar três tipos de acionamento, através de um circuito dedicado, com priorização de entrada; usar interface I2C, como o usado para o display; ou usar radiofrequência, por meio de um par receptor-transmissor integrado no módulo, controles, atuadores e sensores sem fio.

4.3.1.2 Multiplexação no tempo

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

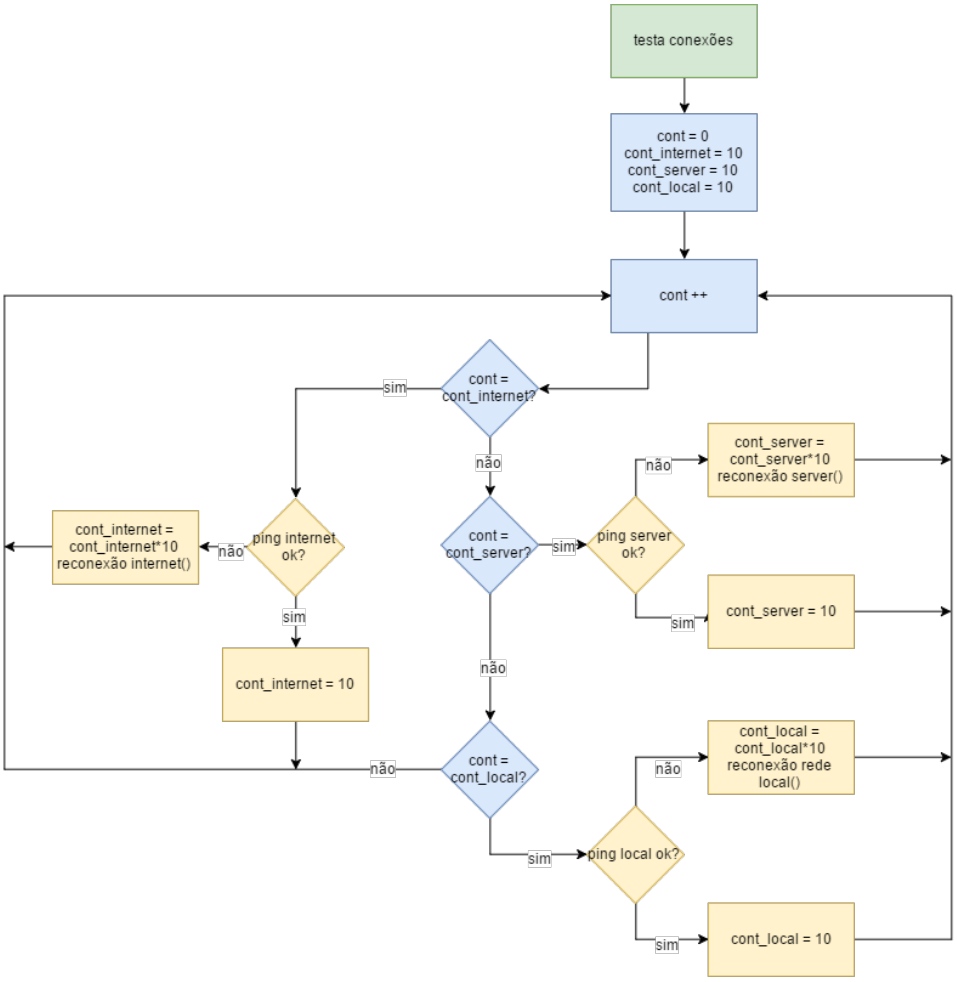
Figura 6: Rotina de multiplexação de procedimentos no tempo



4.3.1.3 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 7: Tratamento de indisponibilidade de recursos



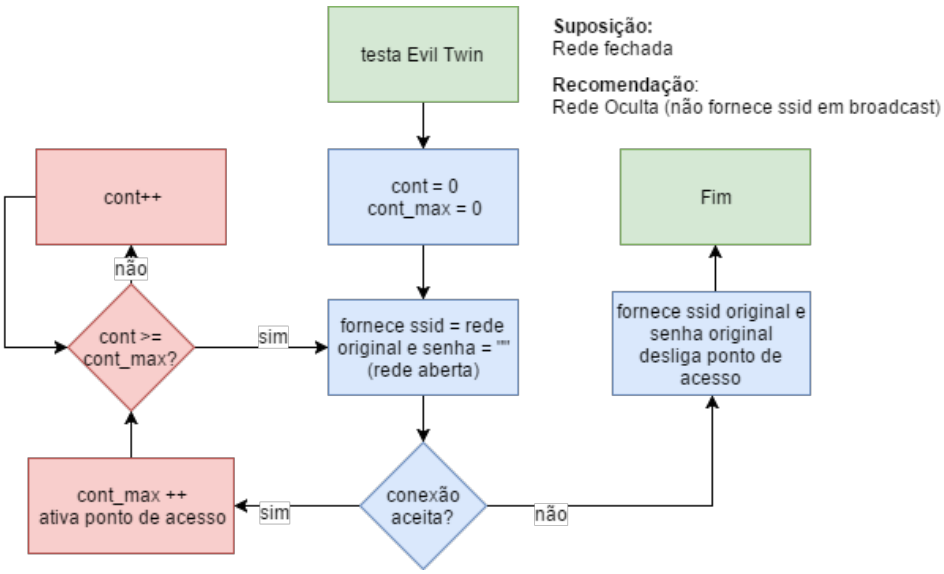
Com esse procedimento, as tentativas de reconexão à Internet, servidor e rede local estão segregadas e com tentativas realizadas em intervalos de tempo sucessivamente maiores. Desta forma, conseguimos gerenciar esses procedimentos, já que o nível de processamento é baixo.

4.3.1.4 DoS Local (*Evil Twin*)

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

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

Figura 8: Tratamento de ataque de DoS Local

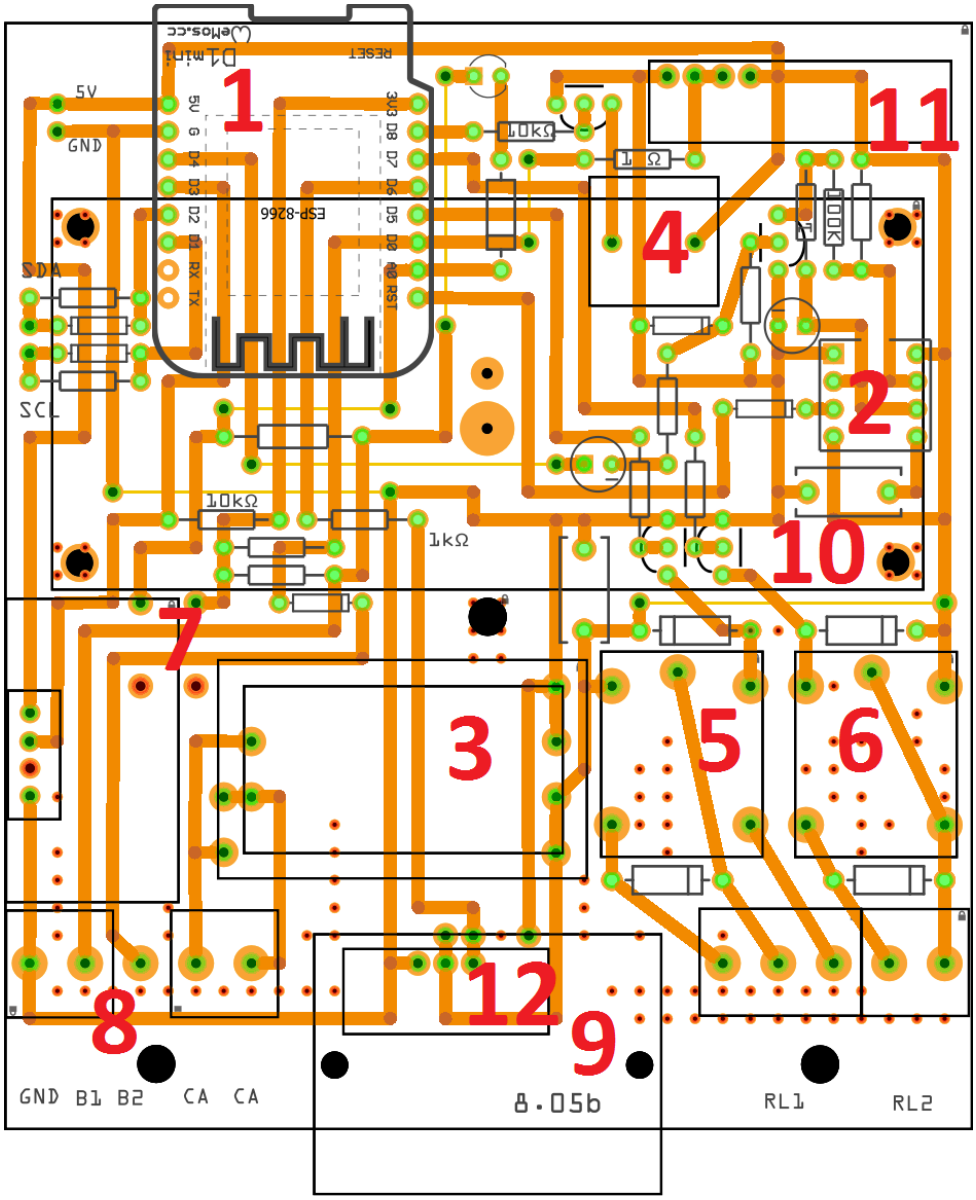


4.3.1.5 Diagrama

Abaixo está o diagrama do circuito impresso (PCB) feito em conjunto com o Projeto Katz-House[3]. Funcionalidade, componentes e arquitetura foram responsabilidade do Projeto Hedwig, enquanto a disposição física, se atentando para problemas de interferência e mantendo o módulo o menor possível foi responsabilidade do Projeto Katz-House.

[3]Katz-House, Fabio Hayashi. Projeto Pessoal, 2017.

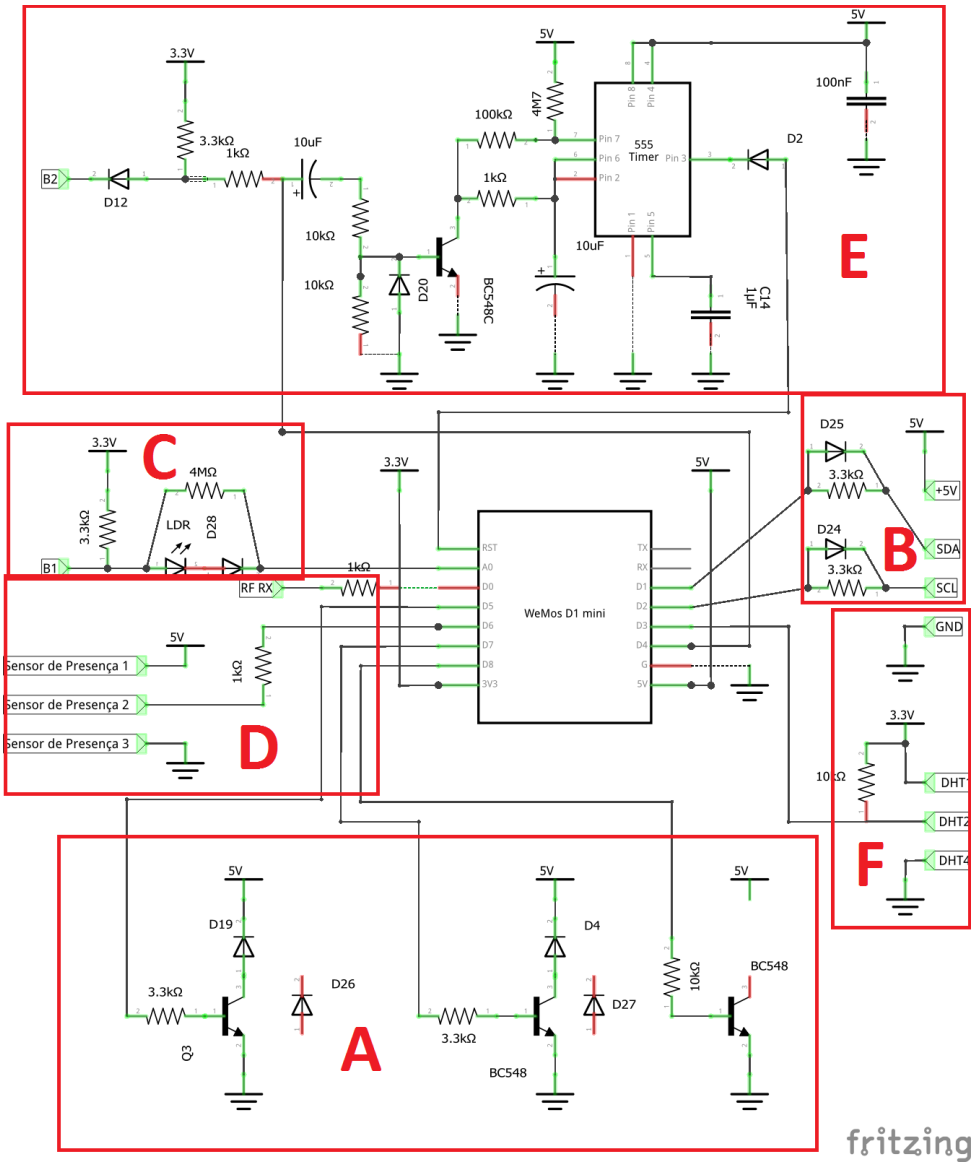
Figura 9: Diagrama PCB do Módulo Base



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 10: Diagrama PCB do Módulo Base

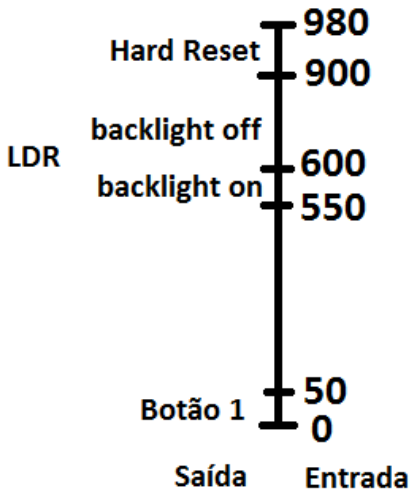


A - Saídas Circuitos simples de transistor para acionamento de relés (para lâmpadas) e buzzer.

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

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

Figura 11: Entradas Em A0



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

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

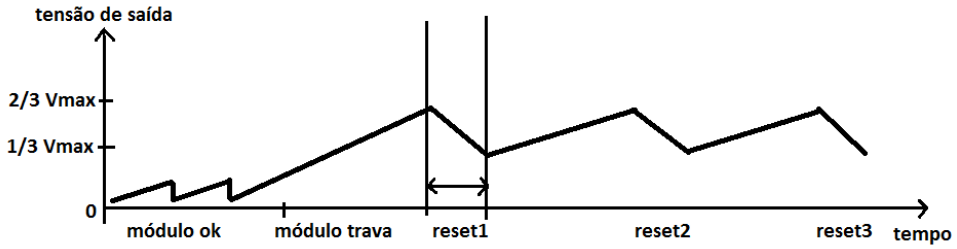
O primeiro capacitor tem como objetivo desacoplamento DC, de forma que a entrada do circuito envolvendo o astável 555 seja somente AC. Assim, travamentos em 0 ou 1 indicam travamento.

Enquanto o LED pisca em intervalos esperados (regularmente), o transistor conduz e mantém uma saída dente de serra muito próxima de 0. Quando o módulo trava, o transistor não conduz mais, e a saída passa a oscilar entre 1/3 e 2/3 da tensão total (observe que o carregamento é feito pelo resistor de 4M7, muito maior que o resistor de 100k, fazendo com que o tempo de carga seja muito maior que o tempo de descarga, uma vez que esses tempos são diretamente proporcionais à constante de tempo dos circuitos RC, que é dada pelo produto do R*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 12: Funcionamento do Circuito de Antitravamento



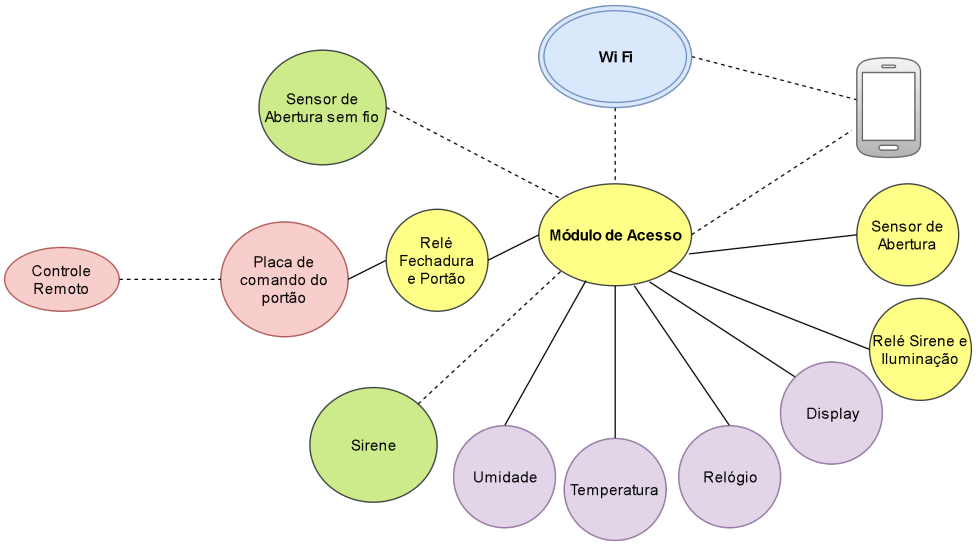
DHT11 Entrada D3 é ligada a uma montagem básica para leitura de umidade e temperatura através do periférico DHT11.

4.3.2 Módulo de Interface com Sistema de Alarmes

4.3.3 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, que é acionada por meio de controle remoto, que utiliza ondas de rádio. Assim, mesmo com falha total do sistema, o usuário pode abrir o portão diretamente, sem a necessidade de acesso à Internet.

Figura 13: Diagrama ilustrativo do módulo de Acesso ao Portão



O diagrama ilustra, em vermelho, os sistemas já existentes. Sensor e sirene sem fio adicionais são mostrados em verde (dispositivos externos ao módulo, que se comunicam por rádio); o próprio módulo de acesso, com um buzzer embutido, e sua conexão com a rede local Wi-Fi ou sua conexão direta com o celular (quando o módulo opera como um ponto de acesso de rede), em amarelo; além de funcionalidades adicionais, 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, a funcionalidade é apenas local (o usuário deve estar com o celular conectado à rede da casa para acessar a página local), e um algoritmo de rotação de teclas é utilizado, para evitar que pessoas mal intencionadas possam (1) olhar e copiar a senha que o usuário digita em seu celular e (2) copiar os dados de abertura e usá-los mais tarde (*middle man*). Na última alternativa, a cada acesso de um usuário, um novo mapeamento de teclas é gerado e enviado ao usuário. Mesmo que haja cópia, 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.

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

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

Por meio dos dados de login dos usuários no sistema, é possível saber qual dos usuários que solicitou a abertura do portão. A persistência destes acessos pode ser analisada, e,

utilizando-se técnicas de Machine Learning, perfis de acesso podem ser determinados, e evoluir até o sistema saber quando houver um acesso em horário inesperado e notificar o usuário remotamente. O aprendizado de máquina é fundamental aqui para descobrir comportamentos que podem ser entendidos como suspeitos. Para um usuário que costuma chegar em um horário aproximado todos os dias, e acionar funções semelhantes da casa, uma tentativa de acesso que não se enquadre em tais padrões pode ser produto de atividade criminosa, a qual pode ser informada pela casa para uma central, que acionaria a polícia caso não seja um falso positivo.

4.4 Controlador Local

Para a intercomunicação entre os módulos e a nuvem, há a presença do servidor local Morpheus, responsável por introduzir mais uma camada de segurança na troca de mensagens. Para isso, foi desenvolvida uma plataforma com a utilização de mensageria e foi definido um protocolo de comunicação entre os serviços de nuvem e os módulos. Assim, quando um usuário desejar realizar 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.

4.4.1 Raspberry Pi

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

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

Figura 14: Raspberry Pi 3 Modelo B

Fonte: (RASPBERRY PI, 2017)

4.5 Servidor na nuvem

O servidor na nuvem tem a responsabilidade de realizar a comunicação entre as aplicações cliente disponíveis para o usuário final e as casas inteligentes, de armazenar os dados coletados pelos sensores e de realizar processamentos que sejam muito onerosos para a capacidade de processamento dos dispositivos físicos. 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* 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 precificação flexível e a facilidade em escalar, é possível aproveitar melhor a capacidade de processamento disponível e evitar desperdício com recursos ociosos.

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

4.5.2 Banco de dados não-relacional

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

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

4.5.3 Banco de dados em memória

Bancos de dados em memória usam a memória principal do computador para realizar o armazenamento de dados ao invés de dispositivos de armazenamento em disco (RAIMA, 2013). Como o tempo de acesso em disco é muito maior, esse tipo de banco de dados

é capaz de atingir altos níveis de performance, proporcionando latências menores e mais previsíveis.

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

4.5.4 WebSockets

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

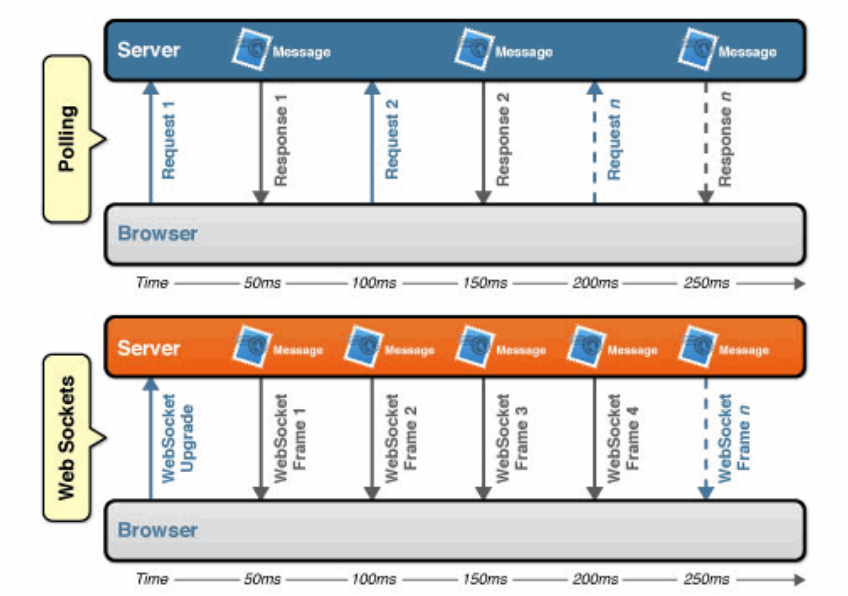
É um protocolo da camada de aplicação compatível com HTTP: a transição entre esses dois protocolos é feita por meio de um handshake que usa o cabeçalho de HTTP Upgrade, como observado na figura a seguir.

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

4.5.5 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.

Figura 15: Comparação entre WebSockets e *polling*



Fonte: (LUBBERS; GRECO, 2016)

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

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

4.5.6 Arquitetura de Microsserviços

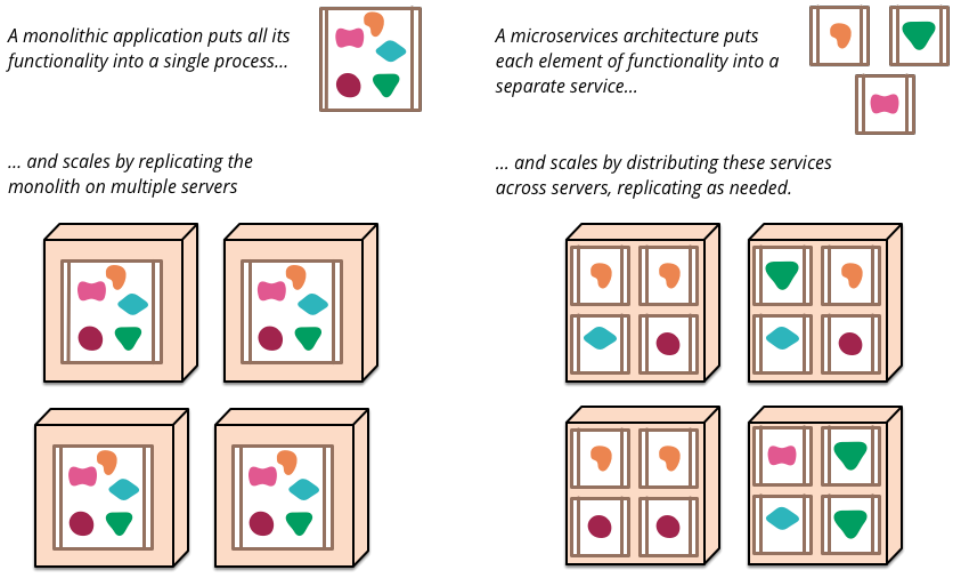
4.5.6.1 Características

A arquitetura de microsserviços é um estilo que compreende 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, podemos compará-la à arquitetura monolítica. Uma aplicação monolítica está contida em uma única unidade, que geralmente é dividida em camadas de funcionalidade tecnológica como interface web, camada de

negócios server-side e camada de persistência de dados. A escalabilidade desse modelo é dada por meio do aumento do número de servidores, máquinas virtuais ou contêineres juntamente a um load balancer - é a chamada escalabilidade horizontal. Uma alteração em uma pequena parte da aplicação significa que toda a aplicação deverá passar por um processo de *build* e *deploy*. Já a arquitetura de microsserviços divide as funcionalidades em serviços autônomos, muitas vezes usando as regras de negócios para realizar essa divisão. Cada serviço tem seu próprio ciclo de desenvolvimento e pode ser atualizado independentemente. A escalabilidade também é tratada serviço a serviço.

Figura 16: Comparação entre uma aplicação monolítica (esquerda) e com microsserviços (direita)



Fonte: (LEWIS; FOWLER, 2014)

É difícil delimitar uma definição formal para arquitetura de microsserviços, pois não existe consenso a respeito de sua definição formal. Contudo, existe uma série de características que projetos usando essa arquitetura compartilham. Detalhamos 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 capabilidades 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.**
- **Cada serviço tem um ciclo de vida independente.**Isso inclui o desenvolvimento e os processos de *deploy*. Um microsserviço pode ser implementado e atualizado independentemente dos outros.

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

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

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

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

4.6 Cliente Web

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

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

4.6.1 *Progressive Web Apps*

4.6.1.1 Contexto

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

Desde então, ocorreram grandes avanços no desenvolvimento web com o amadure-

cimento do HTML5, CSS3 e JavaScript, a criação de novas bibliotecas e ferramentas, surgimento de mais metodologias para design responsivo e a evolução dos navegadores para cumprir os padrões e especificações mais recentes.

Assim, houve o crescimento do número de aplicativos híbridos, que combinam as técnicas de desenvolvimento web com benefícios dos aplicativos nativos como o suporte para usar funções do hardware. Pode-se dividir os aplicativos híbridos em dois grandes grupos (RUDOLPH, 2014): aplicativos que usam WebView, uma espécie de navegador interno que é envolvido por uma aplicação nativa, permitindo que algumas APIs nativas sejam acessíveis por JavaScript, e aplicativos híbridos compilados, que são escritos em uma linguagem não nativa e então compilados para várias plataformas de dispositivos móveis. Isso permite obter versões para várias plataformas com o mesmo código, mas para obter tal resultado há limitações durante o desenvolvimento.

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

4.6.1.2 Conceito

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

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

- **Confiáveis** - devem carregar de forma instantânea, independentemente das condições de conectividade, sem prejudicar a experiência com erros e falhas na aplicação
- **Rápidos** - devem responder rapidamente às interações do usuário, com animações e renderizações suaves
- **Envolventes** - devem oferecer uma experiência imersiva, que se assemelhe à de um aplicativo nativo

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

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

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

4.6.1.3 Tecnologias e técnicas

Manifesto

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

Service Workers

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

4.6.1.4 Aplicações

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

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

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

4.6.2 JSON *Web Tokens*

4.6.2.1 Definição

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

Uma token é uma string composta por três partes separadas por pontos. As três partes são: Cabeçalho, Corpo e Assinatura. O Cabeçalho típico contém o tipo da token

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

4.6.2.2 Autenticação

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

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

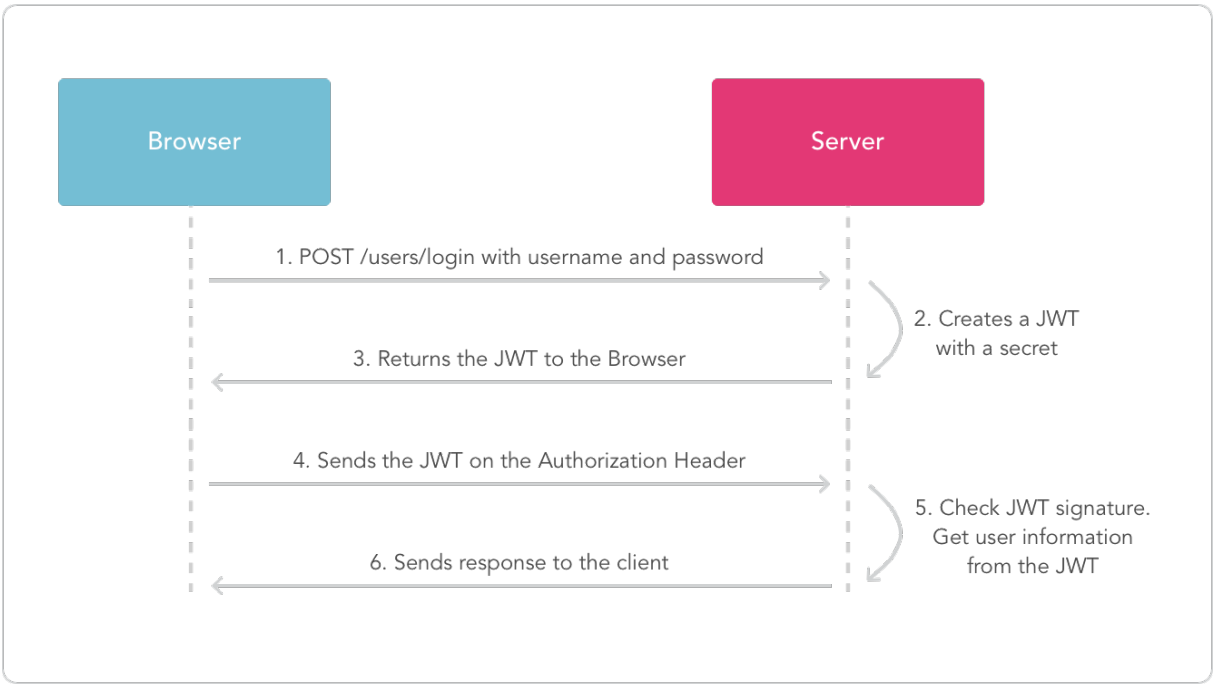
```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    ↪ eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
    ↪ .TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

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

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

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

Figura 17: Diagrama de interação na autenticação por JWT



Fonte: (JWT, 2016)

como é o caso da arquitetura de microsserviços.

4.7 Comunicação

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

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

A infraestrutura de comunicação entre a nuvem e o servidor local, e o servidor local e os sensores e atuadores utiliza o protocolo de aplicação $MQTT$, referência em aplicações IoT no mundo. O protocolo $MQTT$ é estabelecido em cima dos protocolos TCP/IP (nas

camadas inferiores) e é orientado à sessão, diferentemente do protocolo HTTP, de mesma camada.

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

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

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

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

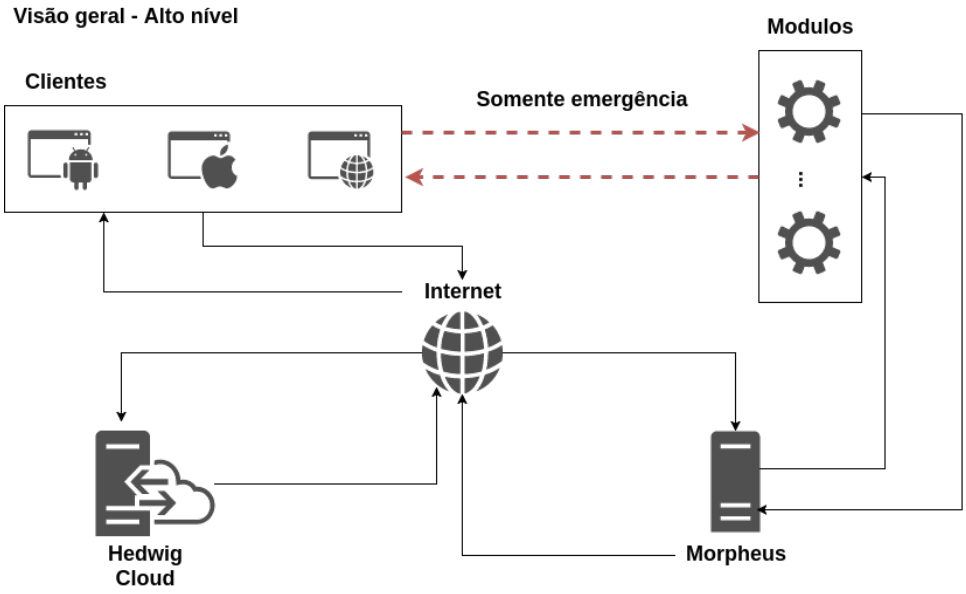
As mensagens são transmitidas em texto puro, e é necessário estabelecer um protocolo para a sua utilização. Utilizaremos aqui o protocolo que define a configurações das mensagens, desenvolvido no projeto HomeSky.

O *broker* Mosquitto[4] será utilizado, e foi escolhido por ser amplamente adotado em projetos de *IoT*, além de ser open source e com licença abrangente (MIT). Entretanto, há diversas possibilidades, como o HiveMQ, adotado no projeto HomeSky, e com grande uso em aplicações enterprise.

A arquitetura de comunicação é representada pelo diagrama abaixo, com um alto nível de abstração, cujos detalhes serão vistos no momento oportuno, com granularidade menor.

[4]https://mosquitto.org/

Figura 18: Visão alto nível da comunicação no Hedwig



4.7.1 Entre módulos e controlador local

4.7.2 Entre controlador local e nuvem

4.7.3 Entre cliente web e nuvem

4.7.4 Entre app backup e módulos

5 Metodologia

5.1 Gerência do projeto

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

Para gerenciar as tarefas, estudos e pesquisas necessárias para a realização do projeto, foi utilizado o Trello[1] - sistema online para organização de ideias e projetos, que permite listagem e acompanhamento de tarefas a serem realizadas, com deadlines, responsáveis e categorização em diversos tipos de tarefas.

5.1.1 Gerência de Escopo Tempo

5.1.2 Gerência de Partes Interessadas Aquisição

5.1.3 Gerência de Processos de Software

Para gerenciar o código-fonte e permitir o trabalho da equipe em múltiplas partes do projeto ao mesmo tempo, foi utilizado o Git, um sistema de controle de versão distribuído. Para publicação do código, foi escolhido o GitHub, onde está a organização do projeto Hedwig[2] e os repositórios de código dos módulos associados ao sistema. A preferência pelo GitHub se deu pelas suas funcionalidades de gerenciamento e colaboração como a notificação de bugs, acompanhamento do progresso de tarefas e criação de wikis, além de ser uma plataforma conhecida por abrigar grandes projetos open-source que chegam a ter centenas ou milhares de contribuidores (GITHUB, 2016).

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

[1] Pode ser acessado gratuitamente em https://trello.com/
[2] https://github.com/hedwig-project

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

5.1.4 Gerência de Partes Interessadas

5.1.5 Gerência de Comunicação

5.1.6 Gerência de Escopo

5.1.7 Gerência de Riscos

5.2 Pesquisa bibliográfica

O estudo dos tópicos relacionados a aprendizagem de máquina foi realizado com auxílio do curso Aprendizagem Automática do Professor Andrew Ng[3], 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 Hopkins[4], também disponíveis no Coursera, foram usados como referência e treinamento para realizar a coleta de dados de maneira metódica. Por esse motivo, foi dada maior atenção ao curso *Getting and Cleaning Data*. Contudo, também foi aproveitado conteúdo do curso *Practical Machine Learning*.

5.3 Ferramentas e tecnologias

Para aprender a utilizar a biblioteca React para o desenvolvimento do front-end, foi usada como referência a documentação oficial[5] oferecida pelo Facebook e o curso *React for Beginners* de Wes Bos[6]. O aprendizado de Redux foi auxiliado pelo curso *Learn Redux*[7], do mesmo autor.

[3]https://www.coursera.org/learn/machine-learning
[4]https://www.coursera.org/specializations/jhu-data-science
[5]https://facebook.github.io/react/docs/hello-world.html
[6]https://reactforbeginners.com/
[7]https://learnredux.com

6 Implementação

6.1 Morpheus

6.1.1 Descrição

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

6.1.2 Plataforma

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

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

possível verificar a arquitetura desenvolvida.

6.1.3 Tecnologias utilizadas

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

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

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

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

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

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

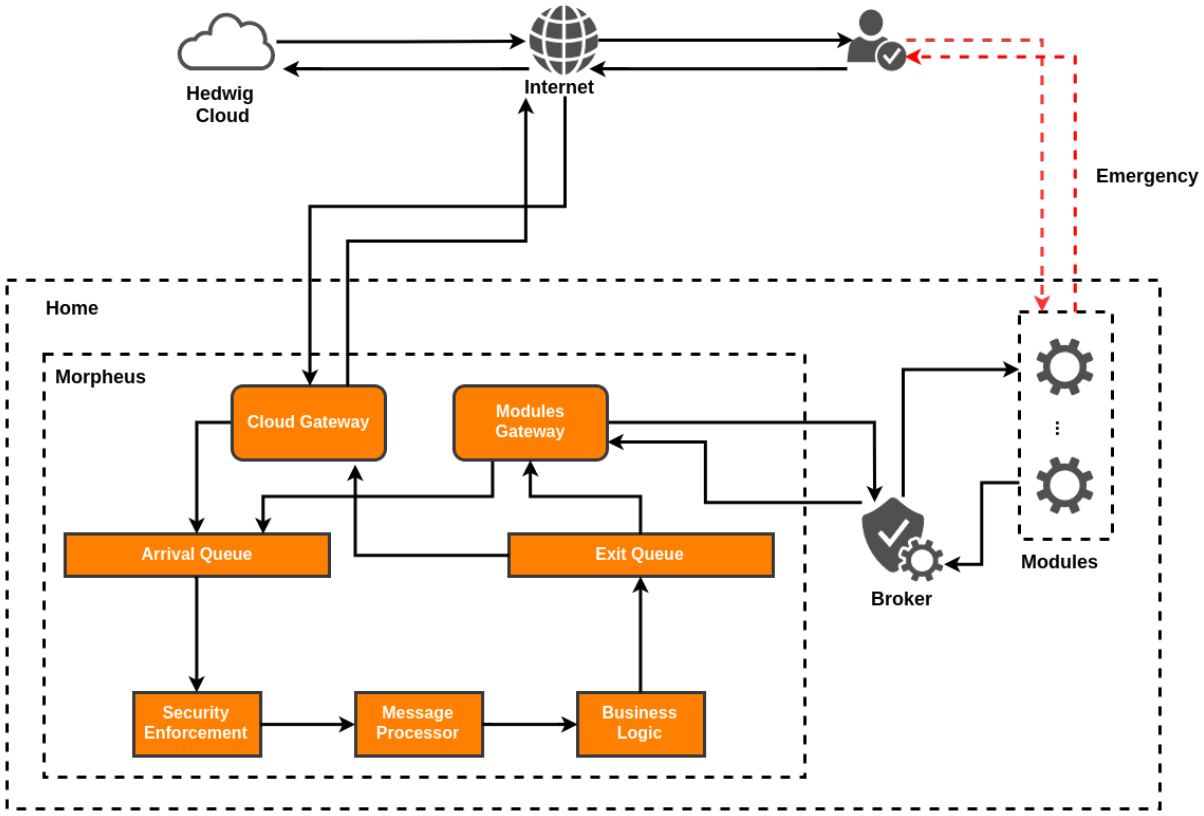
A arquitetura oferecida pelo Java mostra ser efetiva para as necessidades levantadas acima. Com a utilização de uma IDE avançada, inúmeros recursos estão disponíveis para limpeza, refatoração, organização do código, etc. O *framework* Spring Boot[1] foi utilizado para o desenvolvimento, por oferecer diversos recursos que facilitam o desenvolvimento, com configurações de ambiente e o oferecimento de um *container* para inversão de controle e injeção de dependência[2]. Além disso, o gerenciador de dependências Gradle[3] foi também utilizado, por oferecer um poderoso ambiente para configurar, construir e distribuir aplicações. Gradle faz uso de Groovy[4], tecnologia que também roda na *Java Virtual Machine* (JVM). Por outro lado, o uso de tais ferramentas e plataformas necessita de hardware mais robusto, para que funcione, sendo uma desvantagem. Entretanto, frente aos benefícios, ainda é vantajoso a utilização de Java, neste caso.

Java é utilizado em vasta gama de aplicações, deste complexos softwares comerciais como a IDE Eclipse, até software embutidos, como controladores de BlueRay[5]. O padrão de inversão de controle, com injeção de dependências, foi utilizado para o desenvolvimento, de modo a diminuir o acoplamento entre as partes, e facilitar futuras modificações. Assim, cada módulo recebe, em seu construtor, todas as dependências que serão utilizadas. A responsabilidade da construção de tais dependências passa, então, a ser responsabilidade do gerenciador de contexto, e não mais do módulo. A escolha do Java 8 foi decidida para que o desenvolvimento possa utilizar certos recursos de paradigmas funcionais, como o conceito de Streams de dados e Funções Lambdas.

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

[1]https://projects.spring.io/spring-boot/
[2]https://martinfowler.com/articles/injection.html
[3]https://gradle.org/
[4]http://groovy-lang.org/
[5]http://www.oracle.com/technetwork/articles/javame/bluray-142687.html

Figura 19: Arquitetura do servidor local



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

6.1.4 Requisitos

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

6.1.4.1 Requisitos Funcionais

Configuração dos módulos físicos

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

Conexão de emergência

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

Envio de dados para a nuvem

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

Persistência de dados

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

Tentativas de reenvio

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

Envio de e-mail

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

Verificação do time-stamp

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

Tomadas de ação

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

Configuração em arquivo

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

Listeners para diferentes tipos de mensagens

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

6.1.4.2 Requisitos Não Funcionais

Processamento concorrente

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

Utilização de criptografia na troca de mensagens com a nuvem

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

Conversão de mensagens

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

Serialização das configurações

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

Destruição de pools de threads

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

6.1.5 Especificações

6.1.5.1 Tópicos

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

1. Somente o Morpheus conseguirá publicar em qualquer tópico, ou ser um *subscriber* de qualquer tópico.

2. Cada módulo somente consiga publicar no tópico determinado para ele, que será garantido com as credenciais (usuário e senha) fornecidos pelo tópico.

3. Caso um módulo malicioso seja implantado, com o roubo das credenciais de um módulo legítimo, o impacto será unicamente concentrado naquele tópico, não atingindo outros módulos.

Temos as seguintes regras:

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

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

6.1.5.2 Regras de negócio

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

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

2. Cada módulo enviará mensagens para o servidor local com o seus dados, por meio do $MQTT$.

3. Para a troca de senha do *WiFi*, o usuário cadastra no site a nova senha. O servidor na nuvem faz a requisição para o servidor local, o qual enviará um arquivo de configuração com a nova senha para cada um dos módulos registrados. Após a configuração de todos os módulos, o servidor local envia resposta de sucesso para a nuvem, a qual indica ao usuário que a troca de senha já pode ser feita com sucesso.

4. Todo módulo sai de fábrica configurado com o tópico que deve se inscrever e publicar, com base no seu ID, o qual será o seu usuário, e também haverá a senha para se autenticar junto ao *broker* $MQTT$.

6.1.5.3 Definição de interfaces

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

6.1.5.4 Definição das mensagens

Configuração (configuration)

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

Configuração de hora:

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

Configuração de nome:

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

Configuração de comunicação:

```
#configuration
$ts: <timestamp>
$ty: comunication_config
@
new_ssid: <novo_SSID>
new_password: <nova_senha>
ip_local: <novo_IP_local>
ap_mod: <"sempre ativo" | "automatico">
ap_name: <nome_AP_acesso_direto>
ap_password: <senha_AP_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>
@
```

Requisição de ação (action_request)

- Sentido: Servidor para módulo

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

Requisição de acionamento:

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

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

Requisição de SW Restart:

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

Requisição de Teste de Auto Reset:

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

Confirmação (confirmation)

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

Confirmação de hora

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

Confirmação de nome

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

```
#confirmation
$ts: <timestamp>
$ty: rf_confirm64
@
*
<nome_do_[sensor | controle | funcao>: <valor_gravado>
*
@
```

Configuração de Display:

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

Confirmação de SW Restart:

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

Confirmação de Teste de Auto Reset:

```
#confirmation
$ts: <timestamp>
$ty: autoreset_test_confirm
@
autoreset: <0 | 1>
@
```

Transmissão de dados (data_transmission)

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

Transmissão de Umidade, Temperatura, Presença e Reles

```
#data_transmission
$ts: <timestamp>
$ty: temp_umi_pres
@
s1: umidade
vl1: <valor>
s2: temperatura
vl2: <valor>
s3: presenca
vl3: <valor>
s4: rl1
vl4: <valor>
s5: rl2
vl5: <valor>
@
```

Requisição de dados (data_request)

- Sentido: Do módulo para o servidor
- Protocolo: *MQTT*
- Uso: Requisição de alguma informação do servidor. Ex.: Atualização de hora

Requisição de atualização da hora

```
#data_request
$ts: <timestamp>
$ty: time_update
@
@
```

6.1.5.5 Testes realizados da comunicação Morpheus e módulos:

Para que fosse simulado o envio de mensagens, o aplicativo *MQTT* Fx[6] foi utilizado. Com o uso deste software, é possível se inscrever em determinado tópico (enviando as credenciais para o *broker*, tanto em forma de usuário e senha, quando em forma de certificados), bem como publicar no tópico desejado.

Requisição de acionamento 1:

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

Esperado: 0 no serial do Arduino, indicando recebimento

Resultado: De acordo

Requisição de acionamento 2:

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

Esperado: 1 no serial do Arduino, indicando recebimento

Resultado: De acordo

Requisição e confirmação de SW Restart:

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

[6]http://mqttfx.jensd.de/

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

Esperado: Confirmação no tópico MQTT *m2s*

Resultado: De acordo

Configuração e confirmação de RF:

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

Esperado: Confirmação no tópico MQTT *m2s*

Resultado: De acordo

Configuração e confirmação de display:

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

Esperado: Confirmação no tópico MQTT *m2s*

Resultado: De acordo

Transmissão de Umidade Temperatura e Presença e Reles

```
messageToSend = UmiTempPresReles(0,80,25,1,1,0);
//UmiTempPresReles(unsigned long ts, int umidade, float temp, bool pres,
    ↪ bool rele1, bool rele2)
```

Esperado: Mensagem no Tópico MQTT *m2s Resultado: De acordo*

6.1.6 Comunicação entre Morpheus e Nuvem

Inicialmente, foi proposto um modelo arquitetural onde, para a comunicação com a nuvem, haveriam *endpoints* para requisições HTTP tanto do lado da casa quanto do lado da nuvem. Assim, quando o Morpheus precisasse enviar uma mensagem, seria necessário que fosse realizada uma chamada ao *endpoint* correspondente. Neste sentido (Morpheus para nuvem), não há nenhum problema, pois é possível garantir configurações avançadas de segurança, bem como a utilização de load balancers e servidores terceiros (como Akamai[7]), para lidar com ataques do tipo DoS (*Denial of Service*).

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

Foi discutida, então, uma mudança arquitetural na forma de comunicação entre a nuvem e a casa. A solução para o problema se encontra no uso de WebSockets. Assim, o

[7]https://www.akamai.com/

Morpheus se comporta como um cliente em relação à nuvem, e é sempre ele que abre uma conexão. Assim, já não há mais a vulnerabilidade local, de estar exposto às negativas de serviço. Além disso, a conexão se mantém aberta, e forma um caminho *full duplex*, de modo que é possível receber as mensagens da nuvem a qualquer momento também. Com essa arquitetura, os desafios relativos à segurança recaem aos servidores, e não mais à casa, de modo que é possível gerenciar esses riscos, como o fazem grandes empresas, de forma transparente ao cliente final.

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

6.1.7 WebSocket

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

6.1.7.1 Morpheus

O Morpheus ouvirá os seguintes eventos, vindos da nuvem.

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

6.1.7.2 Nuvem

A nuvem ouvirá os seguintes eventos, vindos do Morpheus.

- confirmation
- configuration
- data

Definição de mensagens entre Nuvem e Morpheus

```
configuration =
{
    "configurationId": <configurationId> ,
    "timestamp": <timestamp> ,
    "morpheusConfiguration": <morpheusConfiguration> ,
    "modulesConfiguration": <modulesConfiguration>
}

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

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

Requisitos:

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

Ex: Requisição de mensagens persistidas e configuração do Morpheus

Configuração de módulo

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

```
{
    "moduleId": <moduleId> ,
    "moduleName": <moduleName> ,
    "moduleTopic": <moduleTopic> ,
```

```
    "unregister": <true|false>,
    "messages": [<message> ]
}

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

Ex.: Unregister a module and configure another

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

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

6.1.8 Configurações

6.1.8.1 Configuração do *MQTT* Mosquitto *broker*

Em situações reais, cada casa terá uma instância do *Mosquitto Broker* rodando, independente de todas as outras, e aceitando conexões locais, somente. Entretanto, para que fossem realizados testes e simulações, a instalação e execução de uma instância em cada máquina diferente, localmente, seria inviável. Para tanto, foi configurada uma instalação em uma máquina remota - em servidor da Digital Ocean[8] -, mas com configurações diferentes, uma para cada residência simulada. São executadas instâncias como processos *Daemon*, e vinculadas à portas diferentes - a partir da porta 8883 (conexão com criptografia), para Morpheus, e 1883 para módulos. Também foi criado um script em *bash* para iniciar o processo e ativar as portas no *firewall*. São necessárias as seguintes configurações, para a máquina remota.

[8]Digital Ocean: https://www.digitalocean.com/

1. Habilitar a restrição de tópicos na instância[9]. 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). Essa porta não exige criptografia, mas deve exigir somente usuário e senha (que estarão vulneráveis).

5. O Morpheus será obrigado a se conectar a partir da porta 8883 (e.g. primeira casa → 8884; segunda casa → 8884), passando suas credenciais encriptadas.

6.1.8.2 Guia de instalação (Testado com Ubuntu 16.10 x64)

1. Utilizar o terminal para fornecer os seguintes comandos.

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get install mosquitto mosquitto-clients
   ```

   ```
   sudo systemctl enable mosquitto
   ```

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

   acl_list

   ```
   # General section


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

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

```
# 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 dhould bind to localhost
port 1883
require_certificate false
use_username_as_clientid true

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

passwd

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

```
asd561asd5asd984faee:852456987
```

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

   ```
   . start.sh
   ```

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

```
#!/bin/bash

NUMBER_OF_HOMES=4
BASE_PORT_MORPHEUS=8882
BASE_PORT_MODULES=1882

echo "Oi, $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 "Instancia 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!’
```

6.1.8.3 Criação dos certificados

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

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

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

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

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

3. Morpheus Key e Certificado. Common Name será localhost

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

6.1.8.4 Senhas

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

```
sudo mosquitto_passwd -U passwd
```

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

1. Conectar na porta 1883 sem usuário e senha.

   Esperado: Falha de conexão

   Resultado: Bem sucedido.

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

   Esperado: Permissão de conexão

   Resultado: Bem sucedido.

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

   Esperado: Não publicação

Resultado: Bem sucedido.

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

   Esperado: Publicação

   Resultado: Bem sucedido.

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

   Esperado: Não receber dados

   Resultado: Bem sucedido.

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

   Esperado: Receber dados

   Resultado: Bem sucedido.

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

   Esperado: Publicação ou subscrição com sucesso

   Resultado: Bem sucedido.

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

6.2 Servidor na nuvem

6.2.1 Descrição

O servidor na nuvem é composto por um servidor WebSocket para comunicação entre clientes e casas, um banco de dados em memória para armazenar informações sobre conexões WebSocket ativas, uma API REST para acesso aos dados persistidos, um banco de

[10]Topic Restriction:
⟨http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/⟩
Security Mechanisms:
⟨http://www.steves-internet-guide.com/mqtt-security-mechanisms/⟩
Pub/Sub Client:
⟨https://pubsubclient.knolleary.net/index.html⟩

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.

6.2.2 Requisitos

6.2.2.1 Requisitos funcionais

Comunicação

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

Persistência de dados

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

Gerenciamento de dados

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

6.2.2.2 Requisitos não-funcionais

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

6.2.3 Tecnologias usadas

O servidor foi desenvolvido usando Node.js [11], 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 MongoDB [12], banco de dados não-relacional baseado em documentos. A facilidade de integração com JavaScript e Node.js, a similaridade dos documentos com objetos JSON e a natureza dos dados de sensores foram as motivações para sua escolha como banco de dados principal.

Contudo, não são apenas informações sobre usuários e dispositivos e dados coletados pelos sensores que precisam ser armazenados. Para gerenciar quais Morpheus estão conectados à nuvem e e a quais aplicativos suas informações em tempo real devem ser enviadas, é usado o Redis [13], 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.io [14], 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

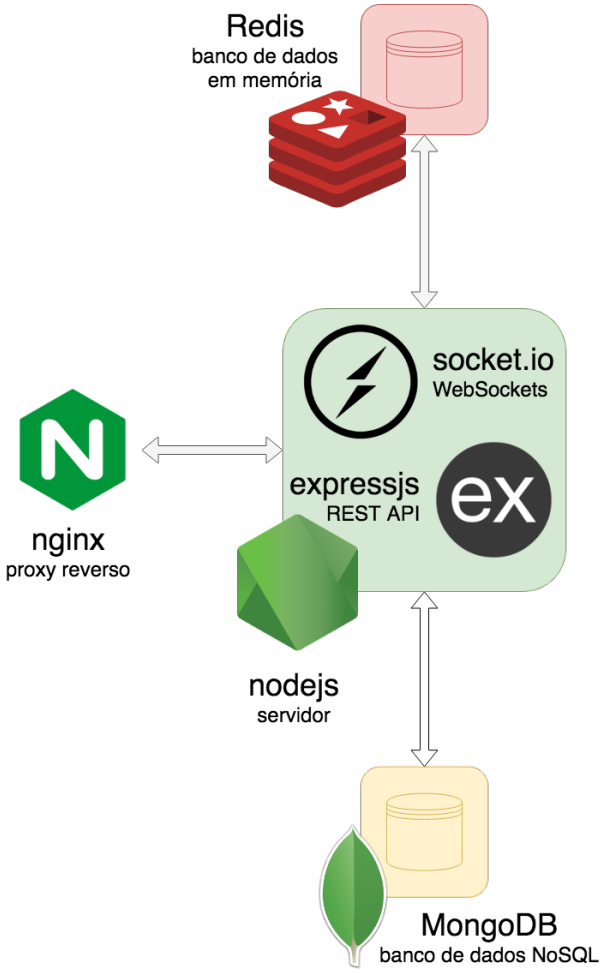
[11] https://nodejs.org
[12] https://www.mongodb.com/
[13] https://redis.io/
[14] https://socket.io/

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 nginx [15], popularmente utilizado como servidor HTTP e proxy genérico para TCP e UDP. Ele permite a configuração de conexões seguras via HTTPS e dispensa a necessidade de delegar privilégios para acessar as portas reservadas 80 e 443 ao processo que roda o servidor Node.js.

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

Figura 20: Componentes e implementação na nuvem



[15] https://nginx.org/

6.2.4 Infraestrutura

Para hospedar o servidor do Hedwig, foi utilizado o serviço de computação na nuvem Digital Ocean [16]. Com ele, foi possível implantar o servidor em uma instância que roda Ubuntu 16.04.3 x64, com 1 CPU, 512MB de memória, 20GB de armazenamento de disco SSD e 1000GB de cota disponível para transferência de dados. O data center que hospeda essa instância fica em Nova Iorque.

6.3 Aplicativo de dashboard

6.3.1 Descrição

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

6.3.2 Requisitos

6.3.2.1 Requisitos funcionais

Realizar cadastro, login e gerenciamento de informações pessoais

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

Gerenciar Morpheus

- O usuário pode adicionar e remover controladores locais - Morpheus, sendo que para o cadastro basta adicionar o número de série do dispositivo.
- O usuário pode configurar o Morpheus, especificando se deseja que mensagens que não puderam ser enviadas por problemas de conectividade sejam mandadas assim que a conexão for reestabelecida.

[16] https://www.digitalocean.com/

Gerenciar módulos

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

Receber dados e enviar ações em tempo real

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

Alterar configurações avançadas dos módulos

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

Visualizar estado das conexões

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

6.3.2.2 Requisitos não-funcionais

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

6.3.3 Tecnologias utilizadas

Para criar a aplicação web que demonstra a funcionamento do Hedwig, foi escolhido o React[17], 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.

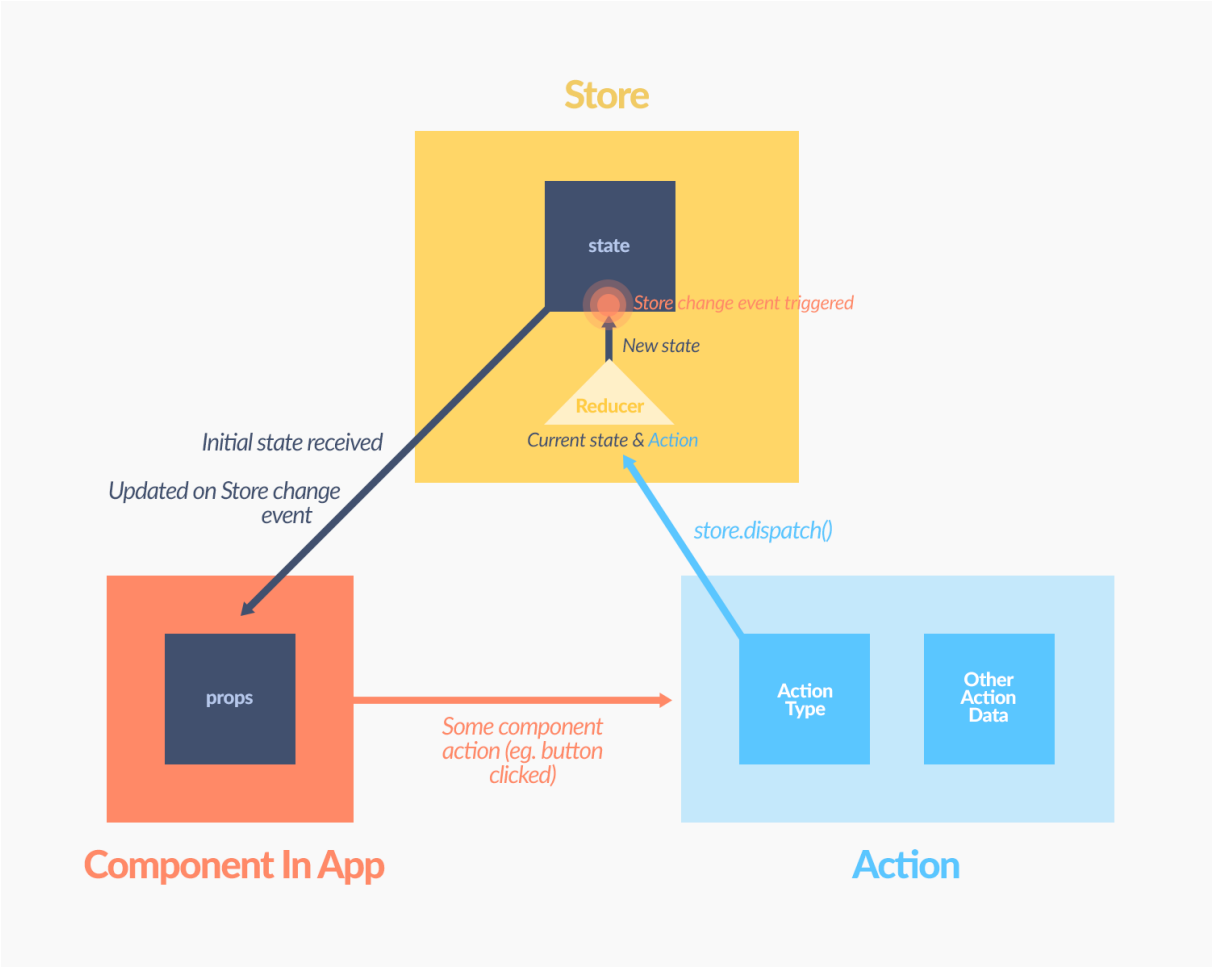
Outro ponto interessante para a utilização do React é que, com a biblioteca React Native[18] - uma extensão do React - é possível a geração de aplicativos nativos para iOS e Android. Assim, caso surja a necessidade de implementar uma nova funcionalidade que possua algum requisito que não pode ser contemplado por um *Progressive Web App*, mas pode ser atendido por um aplicativo nativo, pode-se usar o React Native. Isso diminui a necessidade de retrabalho e dispensa a necessidade de estudo aprofundado das linguagens e ambientes de desenvolvimento tradicionais de projeto de aplicativos nativos.

A fim de facilitar a implementação das interações com o usuário, usamos juntamente ao React a biblioteca Redux[19], um gerenciador de estado global. Dessa forma, pretendemos facilitar o compartilhamento de informações entres diferentes componentes. A motivação por trás do Redux é facilitar a leitura e atualização do estado da aplicação, que, em sites *single-page* modernos, pode armazenar respostas do servidor, cache de dados e dados criados localmente que ainda não foram persistidos no servidor. O Redux é baseado em três princípios (REDUX, 2017):

- O estado é o ponto único de verdade.
- 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.

[17] https://reactjs.org/
[18] https://facebook.github.io/react-native/
[19] https://redux.js.org/

Figura 21: Diagrama de funcionamento do Redux



Fonte: (FACEBOOK, 2016)

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

Uma das vantagens em usar JavaScript para criar a view da aplicação é sua versatilidade, visto que é uma linguagem multi-paradigmas que suporta programação imperativa, declarativa e orientada a objetos (MDN, 2016). Outro ponto é que ela é usada tanto para desenvolvimento front-end como back-end, permitindo que o conhecimento acumulado na execução de um projeto ajude no outro.

Para aproveitar as funcionalidades e facilidades sintáticas das especificações mais recentes de JavaScript, usamos Babel[20], 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 versãp mais recente de JavaScript - sem se preocupar com a compatibilidade do aplicativo com os navegadores que ainda não implementaram essa especificação completamente.

Para agilizar o desenvolvimento e prevenir falhas, usamos o *linter* dedicado a JavaScript ESLint[21]. *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 Webpack[22] 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.

[20] https://babeljs.io/
[21] https://eslint.org/
[22] https://webpack.js.org

6.3.4 Interface

6.3.4.1 Identidade visual

Para facilitar o desenvolvimento, foram usados os componentes do Material Design[23] da *Google*, que satisfazem várias necessidades básicas e casos de uso da criação de interfaces modernas. Para distinguir cada tipo de módulo, foram usados ícones e paletas de cores distintas.

6.3.5 Interações

6.3.5.1 Dados

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

6.3.5.2 Ações

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

6.3.5.3 Conectividade

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

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

[23] https://material.io/

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

6.3.5.4 Aplicativo na tela inicial do dispositivo móvel

Usando um arquivo de manifesto, foi possível configurar como o aplicativo aparece na tela inicial de um dispositivo móvel como um smartphone. Diminuindo o número de passos para o usuário acessar o aplicativo o encoraja a usá-lo mais vezes, além de criar uma sensação parecida com a de um aplicativo nativo.

6.3.6 Publicação

O aplicativo foi publicado com o serviço Surge[24], 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⟩.

6.3.7 Segurança

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

6.4 App Backup

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

[24]https://surge.sh/

6.4.1 Navegação

A partir da tela inicial, um cliente pode verificar todos os módulos presentes em sua casa, visualizar temperatura, umidade, luminosidade, sensores de presença ou abertura e apagar ou acender luzes (para atuadores protegidos com senha, o usuário deve acessar a página do respectivo módulo), numa interface configurável (o usuário pode configurar quais parâmetros observar nesse menu principal). A exibição da Figura 22 é própria para celulares, enquanto a exibição da Figura 23, com posicionamento configurável, simula a planta de uma casa (num cenário real, poderia ser a própria planta da casa do usuário), e é própria para desktops.

Figura 22: Telas principais do aplicativo backup

Figura 23: Visão geral de uma planta com o aplicativo backup

A partir do menu principal, podemos acessar o menu do módulo desejado (vide Figura 24). Nele, encontram-se informações de luminosidade, horário, data, temperatura, umidade, sensor de presença (sensor 1) e sensor de abertura (sensor 2), além de controle de portão,lâmpada ou eletrodoméstico. Também exibe o nível de WiFi do módulo e a versão do firmware.

Figura 24: Menu Principal

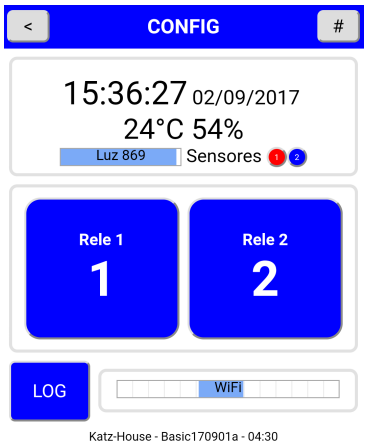


Figura 25: Teclado para digitação da senha

No caso de proteção de controle por senha, é exibido o painel numérico como na Figura 25. A cada requisição da página, o modulo manda um mapeamento das teclas diferente (por exemplo, de teclas A,B,C,... para (3,1), (9,2), ...). Após o usuário entrar com a senha (sequência de teclas do tipo A, B, C, ...), o módulo valida a sequência e autoriza o acionamento. Dessa forma, pessoas não conseguem copiar a senha ao visualizar a sequência de teclas do usuário, tampouco um invasor poderia copiar a sequência e utilizá-la para abertura logo em seguida, pois o mapeamento seria outro.

6.4.2 Configurações

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

Ainda do menu de configurações, pode-se configurar alertas e alarmes (sonoros a partir dos módulos, e pela internet através do provedor gratuito Blynk e e-mail), relés (possibilidade de auto ligar a partir de um sensor específico, ou de ser acionado a partir de um sinal de rádio - usualmente um controle remoto ou até um sensor de abertura adicional, ou mais de um), e o log (quais parâmetros serão persistidos e mandados para a nuvem). Também pode-se acessar o menu de Ferramentas, onde é possível realizar testes de auto reset (para verificação do funcionamento do circuito antitravamento, que age em cerca de 30 segundos), reiniciar o módulo, desconectá-lo, voltar à versão de fábrica (versão implementada em software, enquanto a versão em hardware é realizada por meio de botão oculto), e atualizar o firmware (apenas disponível para administradores). Ao acessar a atualização de firmware, escolhe-se a opção, que mostra a versão atual e, após escolha do arquivo, a versão a ser inserida.

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

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

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

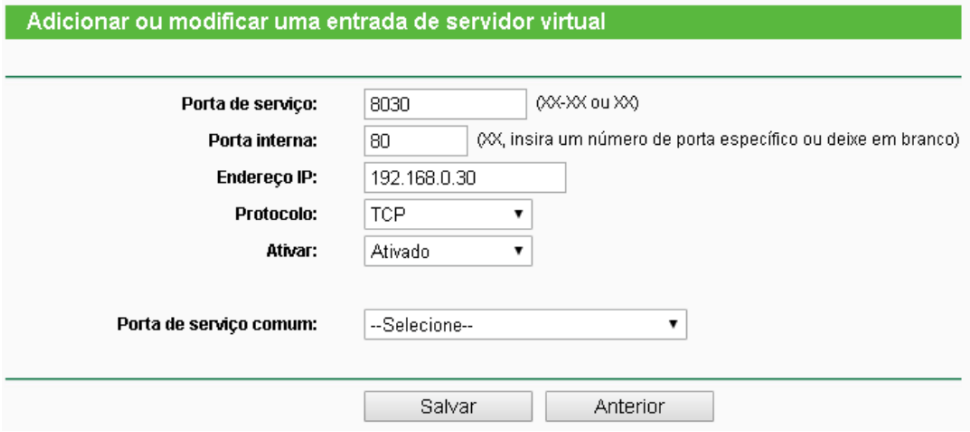
Demais ilustrações, referentes às configurações, estão disponíveis no Anexo C.

6.4.3 Abertura de porta do roteador

Para acessar o módulo/menu remotamente, podemos usar a abertura de porta (NAT/-virtual servers), configurável nas páginas de configuração dos roteadores. Maiores informações para essa configuração podem ser encontradas nos manuais dos roteadores.

(Observe que há uma segurança menor envolvida com essa configuração).

Figura 26: Página de configuração TP Link



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

6.4.4 Controle remoto

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

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

(b) Aperte + (ao lado do rele 1 ou rele 2), espere até a página indicar “Aguardando”, e abra o sensor de abertura. (Podemos incluir diversos sinais para o controle do mesmo relé, abertura ou fechamento de sensor).

(c) Aperte ”OK”no canto superior direito da página, para salvar suas configurações.

6.4.5 Notificações

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

1. Para que o suporte consiga acessar remotamente o módulo e realize a coleta de dados, acesse # → Configurações Avançadas.→ Blynk
2. Insira o Auth Token (e.g. aa7a6dc1170640f08e951ed8cd2198a1).
3. Selecione Notificações ao iniciar: Sim, e WiFi: Sim.
4. Aperte em OK, no canto superior direito, para salvar suas alterações.

6.4.6 Offset Temperatura e Umidade

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

7 Aprendizado de Máquina e Análise de Dados

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

7.1 Coleta e Análise de Dados

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

7.1.1 Conexões

Para melhor diagnóstico do estado de disponibilidade de conexão dos módulos, o seguinte vetor de parâmetros é monitorado ao longo do tempo, para cada módulo instalado:

bit 0: 1 se houve reinicialização do módulo, 0 se não;

bit 1: 1 se houve reconexão de $WiFi$, 0 se não;

bit 2: 1 em caso de reconexão ao broker $MQTT$, 0 se não;

bit 3: 1 em caso de reconexão a servidor para persistência de dados, 0 se não;

Desta forma, podemos acompanhar a relação entre problemas de conexão para posterior análise e tratamento.

7.1.2 Uso

Para monitorar o uso das funcionalidades dos produtos, há o seu monitoramento. Dessa forma, funcionalidades mais usadas podem ser melhoradas e funcionalidades não utilizadas podem ser excluídas, gerando um melhor retorno aos usuários.

Por exemplo, para um uso de automação da iluminação, temos:

bit 0: acionamento manual por botão físico no módulo;

bit 1: acionamento manual por aplicativo de celular;

bit 2: acionamento manual por página web;

bit 3: acionamento automático.

7.1.3 Coleta

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

1. Aquário
2. Corredor
3. Lavanderia
4. Sala/Cozinha
5. Entrada
6. Caixa d'água
7. Victor
8. Jarinu
9. Daniela
10. Hugo
11. Gabriela

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

Para os módulos em Santo André, foi utilizado um dispositivo com cartão SD para persistência de dados. Sua análise tem como objetivo principal acompanhar parâmetros relacionados à disponibilidade, e sua coleta é local. O módulo de Jarinú possui uma interface com o sistema de alarmes (já instalado no local) e tem como objetivo obter dados de presença e abertura de portas (teste de conceito para validar possível integração futura com parceiros estratégicos).

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

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).

Tabela 3: Validação e Análise Final - de 10/09/2017 a 13/11/2017 - Aquário

Descrição	**Valor**
Data Início	1

8 Conclusões

REFERÊNCIAS

9 TO 5 MAC. *Jobs' original vision for the iPhone: No third-party native apps*. 2011. ⟨https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/⟩. Accessed: 2017-11-07.

ATLASSIAN. *Comparing Workflows*. 2017. ⟨https://www.atlassian.com/git/tutorials/comparing-workflows⟩. Accessed: 2017-04-17.

BENSON, M. *An end or an error: App downloads are on a dangerous decline in the USA*. 2016. ⟨https://www.androidauthority.com/end-era-app-downloads-decline-usa-698555/⟩. Accessed: 2017-11-10.

BIBLIOTECA VIRTUAL. *São Paulo: população do estado*. 2017. ⟨http://www.bibliotecavirtual.sp.gov.br/temas/sao-paulo/sao-paulo-populacao-do-estado.php⟩. Accessed: 2017-02-20.

BROWSER COOKIE LIMITS. *Browser Cookie Limits*. 2016. ⟨http://browsercookielimits.squawky.net/⟩. Accessed: 2017-11-08.

BUSINESS WIRE. *Internet of Things Spending Forecast to Grow 17.92016 Led by Manufacturing, Transportation, and Utilities Investments, According to New IDC Spending Guide*. 2017. ⟨http://www.businesswire.com/news/home/20170104005270/en/Internet-Spending-Forecast-Grow-17.9-2016-Led⟩. Accessed: 2017-11-17.

CHAMPEON, S. *Progressive Enhancement and the Future of Web Design*. 2003. ⟨http://hesketh.com/publications/progressive_enhancement_and_the_future_of_web_design.html⟩. Accessed: 2017-11-11.

CHEN, A. *New data shows losing 80the best apps do better*. 2015. ⟨http://andrewchen.co/new-data-shows-why-losing-80-of-your-mobile-users-is-normal-and-that-the-best-apps-do-much-bet⟩. Accessed: 2017-11-10.

CISCO SYSTEMS. *The Zettabyte Era: Trends and Analysis*. 2017. ⟨https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf⟩. Accessed: 2017-11-09.

ELECTRONIC FRONTIER FOUNDATION MEDIA RELEASE. *Movie Legend Hedy Lamarr to be Given Special Award at EFF's Sixth Annual Pioneer Awards*. 1997. ⟨https://w2.eff.org/awards/pioneer/1997.php⟩. Accessed: 2017-05-15.

ESPRESSIF SYSTEMS. *ESP8266EX Datasheet*. 2015. ⟨http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf⟩. Accessed: 2017-05-22.

FACEBOOK OPEN SOURCE. *Building the F8 2016 App: Integrating Data with React Native*. 2016. ⟨http://makeitopen.com/docs/en/1-3-data.html⟩. Accessed: 2017-11-11.

FACEBOOK OPEN SOURCE. *Reconciliation*. 2017. ⟨https://reactjs.org/docs/reconciliation.html⟩. Accessed: 2017-11-09.

G1 SãO PAULO. *Grande SP atinge 83Sabesp*. 2015. ⟨http://glo.bo/1K5JsQh⟩. Accessed: 2017-02-20.

GITHUB. *The state of the Octoverse 2016*. 2016. ⟨https://octoverse.github.com/⟩. Accessed: 2017-05-17.

GOOGLE CHROME. *Managing HTML5 Offline Storage*. 2017. ⟨https://developer.chrome.com/apps/offline_storage⟩. Accessed: 2017-11-08.

GOOGLE DEVELOPERS. *AliExpress*. 2016. ⟨https://developers.google.com/web/showcase/2016/aliexpress⟩. Accessed: 2017-11-08.

GOOGLE DEVELOPERS. *eXtra Electronics*. 2016. ⟨https://developers.google.com/web/showcase/2016/extra⟩. Accessed: 2017-11-08.

GOOGLE DEVELOPERS. *Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage*. 2016. ⟨https://developers.google.com/web/showcase/2017/twitter⟩. Accessed: 2017-11-08.

GOOGLE DEVELOPERS. *Progressive Web App Checklist*. 2017. ⟨https://developers.google.com/web/progressive-web-apps/checklist⟩. Accessed: 2017-11-07.

GOOGLE DEVELOPERS. *Progressive Web Apps*. 2017. ⟨https://developers.google.com/web/progressive-web-apps/⟩. Accessed: 2017-11-07.

I-SCOOP. *IoT in smart home automation: the fast growing list of use cases*. 2017. ⟨https://www.i-scoop.eu/smart-home-home-automation/iot-home-automation-applications/⟩. Accessed: 2017-11-17.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATíSTICA. *Censo Demográfico de 2010. Fundação Instituto Brasileiro de Geografia e Estatística, dados referentes ao município de São Paulo*. 2010. ⟨http://cod.ibge.gov.br/6QV⟩. Accessed: 2017-02-20.

INTERNET ENGINEERING TASK FORCE. *The WebSocket Protocol*. 2011. ⟨https://tools.ietf.org/html/rfc6455⟩. Accessed: 2017-11-19.

INTERNET ENGINEERING TASK FORCE. *JSON Web Token (JWT)*. 2015. ⟨https://tools.ietf.org/html/rfc7519⟩. Accessed: 2017-11-08.

ISO/IEC. Iso/iec is 25010: Software engineering: Software product quality requirements and evaluation (square) — quality model. In: . [S.l.], 2011.

ISO/IEC-IEEE. *ISO/IEC 12207: Systems and software engineering — Software life cycle processes, 2ª ed.* 2. ed. [S.l.]: ISO/IEC-IEEE, 2008.

JAMES, G. e. a. *An Introduction to Statistical Learning with Applications in R*. [S.l.]: [S.L.]: Springer, 2013.

JWT. *Introduction to JSON Web Tokens*. 2016. ⟨https://jwt.io/introduction/⟩. Accessed: 2017-11-08.

KENNEMER, Q. *Google Home gets a $5 million ad spot in the Superbowl*. 2017. ⟨http://phandroid.com/2017/02/01/google-home-gets-a-5-million-ad-spot-in-the-super-bowl⟩. Accessed: 2017-05-15.

LEWIS, J.; FOWLER, M. *Microservices: a definition of this new architectural term*. 2014. ⟨https://martinfowler.com/articles/microservices.html⟩. Accessed: 2017-05-22.

LUBBERS, P.; GRECO, F. *HTML5 WebSocket: A Quantum Leap in Scalability for the Web*. 2016. ⟨https://websocket.org/quantum.html⟩. Accessed: 2017-11-19.

MCKINSEY & COMPANY. *There's No Place Like A Connected Home*. 2016. ⟨http://www.mckinsey.com/spContent/connected_homes/index.html⟩. Accessed: 2017-04-24.

MOZILLA DEVELOPER NETWORK. *JavaScript*. 2016. ⟨https://developer.mozilla.org/bm/docs/Web/JavaScript⟩. Accessed: 2017-11-11.

MULLINS, C. S. *What is an In-Memory Database System?* 2017. ⟨http://www.dbta.com/Columns/DBA-Corner/What-is-an-In-Memory-Database-System-119241.aspx⟩. Accessed: 2017-11-19.

MURAMATSU, F. T.; RODRIGUES, H.; GALLEGO, R. B. Projeto homesky. trabalho de conclusão de curso (graduação em engenharia de computação) - escola politecnica, universidade de são paulo, são paulo. 2016.

MYSLEWSKI, R. *iPhone App Store breezes past 500 million downloads*. 2009. ⟨https://www.theregister.co.uk/2009/01/16/half_billion_iphone_apps/⟩. Accessed: 2017-11-10.

NOSQL DATABASES. *NOSQL Databases*. 2009. ⟨http://nosql-database.org/⟩. Accessed: 2017-11-17.

PEREIRA, T. *Quando utilizar RDBMS ou NoSQL?* 2016. ⟨http://datascienceacademy.com.br/blog/quando-utilizar-rdbms-ou-nosql/⟩. Accessed: 2017-11-19.

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

RAIMA. *In-Memory Advantages From RDM*. 2013. ⟨https://raima.com/in-memory-database/⟩. Accessed: 2017-11-19.

RASPBERRY PI FOUNDATION. *Model 3*. 2017. ⟨https://www.raspberrypi.org/products/raspberry-pi-3-model-b⟩. Accessed: 2017-06-12.

REDUX. *Introduction*. 2017. ⟨https://redux.js.org/docs/introduction/⟩. Accessed: 2017-11-09.

RICKER, T. *Jobs: App Store launching with 500 iPhone applications, 25free*. 2008. ⟨https://www.engadget.com/2008/07/10/jobs-app-store-launching-with-500-iphone-applications-25-free/⟩. Accessed: 2017-11-10.

ROTEM-GAL-OZ, A. *Services, Microservices, Nanoservices – oh my!* 2014. ⟨http://arnon.me/2014/03/services-microservices-nanoservices/⟩. Accessed: 2017-05-23.

RUDOLPH, P. *Hybrid Mobile Apps: Providing A Native Experience With Web Technologies*. 2014. ⟨https://www.smashingmagazine.com/2014/10/providing-a-native-experience-with-web-technologies/⟩. Accessed: 2017-11-08.

RUSELL, A. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*. 2015. ⟨https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/⟩. Accessed: 2017-11-07.

SHEARER, S. M. *Beautiful: The Life of Hedy Lamarr*. [S.l.]: Thomas Dunne Books, 2010. ISBN 978-0-312-55098-1.

SUBLIMELINTER. *About SublimeLinter*. 2016. ⟨http://www.sublimelinter.com/en/latest/about.html⟩. Accessed: 2017-11-09.

THOMSEN, A. *Como programar o NodeMCU com IDE Arduino.* 2016. ⟨https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/⟩. Accessed: 2017-05-22.

VISWANATHAN, P. *Cloud Computing and Is it Really All That Beneficial?* 2017. ⟨https://www.lifewire.com/cloud-computing-explained-2373125⟩. Accessed: 2017-11-18.

ANEXO A – CÓDIGOS DAS APLICAÇÕES DESENVOLVIDAS

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

ANEXO B – LISTA DE MATERIAIS PARA MONTAGEM DOS MÓDULOS

Tabela 4: Lista de materiais

Item	**Quantidade**
Raspberry Pi 3 Modelo B	1
Wemos D1 Mini	7
Cabo fêmea-fêmea 20cm	1
RF 433	4
DHT 11	4
Módulo I2C	4
Display LCD 16x2	4
Fonte 5V 3W	4
Sensor de presença embutido	5
Sensor de abertura sem-fio	7
Controle remoto	5
Chave push button R13-507 sem trava preta	15
Chave táctil 6x6x5mm 4 terminais	10
Rolo de solda Best azul 189 MSX10 60x40 1/2 kg fio 1mm	1
Placa de circuito impresso padrão 10x20cm tipo ilha	2
Cabo de força	6
Caixa patola PB-114/2 36x97x147	4
Relê T73 5V 1 pólo 2 posições 5 terminais 125V 10A	8
Circuito integrado LM555 (NE555/NE555P)	4
Buzzer 12mm com oscilador interno	4
Borne KF-3000 2 terminais	10
Borne KF-3000 3 terminais	10
Capacitor de tântalo 10µF	4
LDR	4
Capacitor poliéster 100nF	8
Chave gangorra KCD1-102 preta 3 terminais	5
Resistor 1k	100
Resistor 3k	100
Resistor 3M3	100

ANEXO C – IMAGENS DE CONFIGURAÇÃO PARA O APLICATIVO BACKUP

Images will be placed here