

**LETÍCIA DA SILVA MOTA**

**UMA PROPOSTA DE SISTEMA DE CONTROLE DE ACESSO A  
MÚLTIPLOS AMBIENTES BASEADA EM INTERNET DAS COISAS**

Trabalho de Conclusão de Curso apresentado  
à banca avaliadora do Curso de Engenharia  
de Computação, da Escola Superior de  
Tecnologia, da Universidade do Estado do  
Amazonas, como pré-requisito para obtenção  
do título de Engenheira de Computação.

Orientador(a): Profa. Dra. Elloá Barreto Guedes da Costa

Manaus – Junho – 2019

## **Ficha Catalográfica**

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).  
**Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.**

M917p Mota, Letícia da Silva

Uma proposta de sistema de controle de acesso a múltiplos ambientes baseada em Internet das Coisas / Letícia da Silva Mota. Manaus : [s.n], 2019.

116 f.: color.; 30 cm.

TCC - Graduação em Engenharia de Computação - Universidade do Estado do Amazonas, Manaus, 2019.

Inclui bibliografia

Orientador: Costa, Elloá Barreto Guedes da

1. Internet das Coisas. 2. Controle de Acesso. 3. RFID. 4. MQTT. I. Costa, Elloá Barreto Guedes da (Orient.). II. Universidade do Estado do Amazonas. III. Uma proposta de sistema de controle de acesso a múltiplos ambientes baseada em Internet das Coisas

**Elaborado por Jeane Macelino Galves - CRB-11/463**

**LETÍCIA DA SILVA MOTA**

**UMA PROPOSTA DE SISTEMA DE CONTROLE DE ACESSO A  
MÚLTIPLOS AMBIENTES BASEADA EM INTERNET DAS COISAS**

Trabalho de Conclusão de Curso apresentado  
à banca avaliadora do Curso de Engenharia  
de Computação, da Escola Superior de  
Tecnologia, da Universidade do Estado do  
Amazonas, como pré-requisito para obtenção  
do título de Engenheira de Computação.

**Aprovado em: 17/06/2019**

BANCA EXAMINADORA

Elloá Barreto Guedes da Costa

**Profa. Dra. Elloá Barreto Guedes da Costa**  
*UNIVERSIDADE DO ESTADO DO AMAZONAS*

Almir de Oliveira Costa Junior

**Prof. Almir de Oliveira Costa Junior, M.Sc.**  
*UNIVERSIDADE DO ESTADO DO AMAZONAS*

Carlos Maurício Serodio Figueiredo

**Prof. Carlos Maurício Serodio Figueiredo, D.Sc.**  
*UNIVERSIDADE DO ESTADO DO AMAZONAS*

# Resumo

Este trabalho de conclusão de curso contempla uma proposta de sistema de controle de acesso a múltiplos ambientes. O sistema proposto é baseado em Internet das Coisas, considera o projeto de um objeto inteligente de baixo custo, contempla agendamentos e registros de pontualidade dos usuários e faz uso de uma padronização de tópicos para troca de mensagens. O projeto desta solução integra as regras de negócio para normatização do funcionamento deste sistema, a especificação completa em termos de *hardware* e *software* do objeto inteligente, de um painel administrativo sob a forma de uma aplicação web e de um orquestrador de dados para comunicação entre estas partes. As tecnologias envolvidas na elaboração da solução proposta contemplam: protocolo MQTT, para troca de mensagens leves; tecnologia RFID, para interação do usuário com o sistema; NodeMCU, microcontrolador munido de *Wi-Fi* para conexão dos objetos à internet; e, *framework* Web2py, para desenvolvimento da aplicação web. Quando analisada de maneira comparativa com soluções análogas na literatura, a solução proposta mostra-se escalável para vários objetos e disponibiliza código aberto para livre reprodução.

**Palavras-Chave.** Internet das Coisas; Controle de Acesso; RFID; MQTT.

# Abstract

*This work aims at proposing a multi-location access control system. The proposed solution is based on Internet of Things, considers the design of a low-cost intelligent object, contemplates users' schedules and records of punctuality and uses standard topics on the message exchange process. The design of this solution integrates business requirements for system processes standardization, the entire specification of the intelligent object in terms of hardware and software, an administrative panel comprised of a web application and a data orchestrator for communication between these modules. The technologies involved in the elaboration of the proposed solution include: MQTT protocol, for lightweight message exchange; RFID technology, for the interaction between users and the system; NodeMCU, microcontroller equipped with Wi-Fi, to connect objects to the Internet; and Web2py framework, for web application development. When compared with similar solutions in the literature, the proposed solution is scalable for multiple objects and provides open source code for free reproduction.*

**Keywords.** *Internet of Things; Access Control; RFID; MQTT.*

# Agradecimentos

Agradeço primeiramente a Deus por me proporcionar ter chegado até aqui e por ter colocado em meu caminho pessoas especiais que me deram o suporte e apoio necessários para manter-me firme nesta caminhada, que hoje tem mais um de seus trechos concluído. Em especial, dedico esta etapa vencida a minha avó Terezinha, que é para mim a maior representação de fé e amor. Agradeço a minha tia, prima, meu pai e meu namorado por todo carinho, amor e incentivo que recebi de vocês.

Aos amigos, por todas as experiências vividas juntos e a todos os professores que contribuíram com a minha trajetória acadêmica, especialmente à minha orientadora por esclarecer tantas dúvidas, por todas as conversas, conselhos e confiança depositada no desenvolvimento deste trabalho.

Agradeço também os recursos materiais e financeiros providos pela Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por meio do Projeto FAPEAM/CNPQ PPP 04/2017.

## **Epígrafe**

*Seja corajoso, seja determinado, supere as probabilidades.*

Stephen Hawking

# Sumário

<b>Lista de Tabelas</b>	<b>ix</b>
<b>Lista de Figuras</b>	<b>xi</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Objetivos . . . . .	3
1.2 Justificativa . . . . .	3
1.3 Metodologia . . . . .	4
1.4 Organização do Documento . . . . .	5
<b>2 Fundamentação Teórica</b>	<b>6</b>
2.1 Internet das Coisas . . . . .	6
2.2 Família ESP8266 e o NodeMCU . . . . .	7
2.3 Identificação de Rádio Frequência . . . . .	10
2.4 MQTT . . . . .	13
2.5 <i>Framework</i> Web2py . . . . .	16
2.6 Trabalhos Relacionados . . . . .	18
<b>3 Solução Proposta</b>	<b>22</b>
3.1 Visão Geral da Solução Proposta . . . . .	22
3.2 Diagramas de Caso de Uso . . . . .	25
3.3 Projeto do Hardware do Objeto Inteligente . . . . .	29
3.4 Projeto de Software . . . . .	32

<b>4 Resultados e Discussão</b>	<b>40</b>
4.1 Padronização dos Tópicos de Comunicação . . . . .	40
4.2 Implementação do Projeto de Hardware . . . . .	42
4.3 Implementação do Software Embocado . . . . .	45
4.4 Implementação da Aplicação Web . . . . .	48
4.5 Implementação do Orquestrador . . . . .	55
4.6 Análise Comparativa . . . . .	56
<b>5 Considerações Finais</b>	<b>58</b>
<b>A Detalhamento dos Casos de Uso</b>	<b>63</b>
<b>B Diagramas de Sequência</b>	<b>82</b>
<b>C Código-Fonte do Orquestrador Publisher/Subscriber</b>	<b>88</b>
<b>D Código-Fonte do Software Embocado do Objeto Inteligente</b>	<b>96</b>

# **Lista de Tabelas**

4.1 Custo total dos componentes para implementação de um protótipo do objeto inteligente. Estes preços foram praticados em Manaus no mês de Setembro de 2018. . . . .	42
4.2 Especificação da conexão de pinos entre NodeMCU e módulo serial I2C. . . . .	43
4.3 Especificação da conexão de pinos entre NodeMCU e módulo RFID. . . . .	43
4.4 Especificação da conexão de pinos entre NodeMCU e módulo relé. . . . .	44
4.5 Tabela comparativa da solução proposta em relação a outras alternativas existentes.	57

# **Lista de Figuras**

2.1	Plataforma de desenvolvimento <i>NodeMCU</i> . Fonte: (TECHNOLOGY, 2017). . . . .	9
2.2	Esquema de pinos na plataforma <i>NodeMCU</i> . Fonte: (ESPRESSIF, 2018). . . . .	9
2.3	Transmissão de dados usando RFID. Adaptada de: (PANDIAN, 2010). . . . .	10
2.4	Exemplo de comunicação M2M com o protocolo MQTT intermediada por um <i>broker</i> . Fonte: < <a href="http://goo.gl/s1mBTk">http://goo.gl/s1mBTk</a> >. . . . .	13
2.5	Exemplo detalhado do padrão <i>publish-subscribe</i> no MQTT. Fonte: < <a href="https://goo.gl/iavnm4">https://goo.gl/iavnm4</a> >. . . . .	15
2.6	Especificação do modelo MVC. Fonte: < <a href="https://bit.ly/2ptI7wz">https://bit.ly/2ptI7wz</a> >. . . . .	17
3.1	Ideia geral da solução proposta. Fonte: Própria (2018) . . . . .	23
3.2	Esquema de fluxo de informação e componentes da solução proposta. . . . .	24
3.3	Caso de Uso: Módulo de Gerenciamento de Salas . . . . .	26
3.4	Caso de Uso: Módulo de Gerenciamento de Usuários . . . . .	26
3.5	Caso de Uso: Módulo de Gerenciamento de <i>Tags</i> . . . . .	27
3.6	Caso de Uso: Módulo de Gerenciamento de Agendamentos . . . . .	28
3.7	Caso de Uso: Módulo de Gerenciamento de Relatórios . . . . .	28
3.8	Diagrama esquemático do projeto de <i>hardware</i> do protótipo do objeto inteli-gente. Os símbolos + e – denotam os polos do circuito elétrico que alimentam a fechadura e o dispositivo interno. . . . .	31
3.9	Esquema elétrico do projeto de <i>hardware</i> . . . . .	31
3.10	Diagrama de pacotes da aplicação web do sistema de controle de acesso. . . . .	34

3.11	Diagrama de entidade-relacionamento da aplicação web do sistema de controle de acesso.	35
3.12	Diagrama de sequência para a Associação de <i>Tag-Usuário</i> .	36
3.13	Diagrama de sequência para a Associação de <i>Tag-Master</i> .	37
4.1	Protótipo do <i>hardware</i> do objeto inteligente.	44
4.2	Fluxograma que identifica etapas gerais do processo de comunicação entre módulos do projeto.	46
4.3	Fluxograma que identifica etapas de cadastramento de <i>Tag Master</i> .	47
4.4	Fluxograma que identifica etapas de cadastramento de <i>Tag User</i> .	47
4.5	Fluxograma que identifica etapas de verificação de registros de Agendamento a partir da <i>Tag</i> de usuário.	47
4.6	Página de login da aplicação web.	49
4.7	Página principal da aplicação web após autenticação do usuário.	49
4.8	Página de cadastro de usuários	50
4.9	Página de associação de <i>tag</i> a perfil master.	50
4.10	Página de associação de <i>tag</i> a perfil usuário.	51
4.11	Diagrama detalhado da classe Publisher.	51
4.12	Listagem de usuários cadastrados.	52
4.13	Página de cadastro de salas.	52
4.14	Listagem de salas cadastradas.	53
4.15	Página de cadastro de agendamento.	53
4.16	Listagem de registros de agendamentos.	54
4.17	Listagem de registros de pontualidade e frequência.	54
4.18	Estatísticas de registros de pontualidade e frequência.	54
4.19	Diagrama de classes do módulo orquestrador.	55
B.1	Diagrama de sequência para o Cadastro de Sala.	82
B.2	Diagrama de sequência para a Listagem de Salas Cadastradas.	83

B.3	Diagrama de sequência para a Edição de Cadastro de Sala. . . . .	83
B.4	Diagrama de sequência para a Inativação de Cadastro de Sala. . . . .	83
B.5	Diagrama de sequência para o Cadastro de Usuário. . . . .	84
B.6	Diagrama de sequência para a Listagem de Usuários Cadastrados. . . . .	84
B.7	Diagrama de sequência para a Edição de Cadastro de Usuário. . . . .	85
B.8	Diagrama de sequência para a Inativação de Cadastro de Usuário. . . . .	85
B.9	Diagrama de sequência para o Cadastro de Agendamento. . . . .	86
B.10	Diagrama de sequência para a Listagem de Agendamentos Cadastrados. . . . .	86
B.11	Diagrama de sequência para a Edição de Cadastro de Agendamento. . . . .	86
B.12	Diagrama de sequência para a Exclusão de Cadastro de Agendamento. . . . .	87
B.13	Diagrama de sequência para Relatório de Pontualidade. . . . .	87
B.14	Diagrama de sequência para Relatório de Taxa de Frequência. . . . .	87
C.1	Classe do pacote orquestrador responsável por inscrever-se em tópicos de saída de objetos inteligentes. . . . .	89
C.2	Classe do pacote orquestrador responsável por manipular e analisar dados conforme solicitado por objetos inteligentes. . . . .	92
C.3	Classe do pacote orquestrador responsável por publicar mensagens a tópicos de entrada de objetos inteligentes para posterior exibição no visor LCD. . . . .	95
D.1	Código-Fonte do Software Embarcado . . . . .	97

# Capítulo 1

## Introdução

A *Internet das Coisas* (IoT, do inglês *Internet of Things*) está mudando a forma como as pessoas vivem e representa um dos maiores desafios a serem enfrentados pela indústria (SCHWARTZ, 2016a). Ela não diz respeito apenas a manter dispositivos conectados à Internet, mas de fazer objetos antes inertes tornarem-se “espertos”, dando-lhes habilidades para sentir, comunicar e responder (JAVED, 2016). Tamanha será a demanda por soluções desta natureza que analistas prevêem que no ano de 2020 haverá mais de 50 bilhões de dispositivos conectados e a receita total de IoT facilmente irá ultrapassar US\$ 1.5 trilhão (JAVED, 2016).

De maneira geral, as soluções atuais de IoT têm se concentrado em propor dispositivos de baixo custo que coletam grandes quantidades de dados, interagem uns com os outros e tiram proveito de serviços e de armazenamento em nuvem. Desenvolvedores de todo o mundo têm trabalhado em projetos que transformam objetos cotidianos em dispositivos inteligentes, com sensores e atuadores (SCHWARTZ, 2016a). Exemplos de soluções neste sentido que já permeiam o cotidiano são os *smartwatches* que, dentre outros, monitoram os sinais vitais, e monitores de segurança que geram alertas via mensagens mediante motivação não habitual.

Considerando o grande potencial de soluções a serem desenvolvidas neste domínio, é natural questionar-se sobre objetos do dia a dia que poderiam ser melhorados com IoT, diminuindo esforços, riscos e custos ao passo que geram ganhos de tempo e de informações, por exemplo. Neste sentido, foi observada uma problemática típica da instituição de ensino em que este trabalho de conclusão de curso se desenvolveu: diariamente, a cada tempo de aula, funcionários

do apoio administrativo movimentam-se por toda a instituição para abrir portas e conceder acesso para docentes e discentes às respectivas salas de aula onde desenvolverão atividades de ensino. Normalmente, há apenas um funcionário deste tipo por turno, o que gera sobrecarga e eventuais atrasos diante de horários de pico. Sabe-se também que as salas são alocadas de maneira fixa ao longo do semestre para as disciplinas, de modo que docentes e discentes sabem previamente onde as aulas ocorrerão. Além do que foi exposto, a abertura da sala só se dá mediante a presença do docente responsável e que está alocado para ela no respectivo horário, garantindo o uso apropriado do espaço e preservando os bens tipicamente ali contidos (ar condicionado, *datashow*, etc.).

Diante do problema constatado, observado há anos na Escola Superior de Tecnologia da Universidade do Estado do Amazonas, mas potencialmente típico de outras instituições de ensino similares, resolveu-se endereçá-lo, no escopo deste trabalho, por meio de uma solução de sistema de controle de acesso baseado em IoT. O sistema de controle de acesso a ambientes aqui proposto leva em conta a existência de múltiplos usuários (docentes), múltiplos ambientes (salas), restrições de horário (horários das aulas dos docentes) e provê um painel administrativo por meio de uma aplicação web. No acesso de cada sala, ao invés de uma fechadura comum, tem-se um objeto inteligente capaz de garantir o acesso de um docente em horário previamente alocado mediante validação com cartão individual de acesso e comunicação remota com um banco de dados. Além de validar este acesso, o objeto inteligente coleta o horário de entrada do docente, permitindo a geração de relatórios de pontualidade, e também pode ser utilizado para cadastro de novos cartões de acesso, dispensando o uso de cadastradoras. Neste objeto inteligente também há componentes capazes de habilitar a alimentação elétrica de um dispositivo interno da sala, a exemplo de um *datashow* ou ar-condicionado.

Levando em consideração o contexto descrito, este trabalho de conclusão de curso apresenta os resultados da elaboração de um projeto de *hardware*, *software* e comunicação dos elementos que integram um sistema de controle de acesso a múltiplos ambientes baseado em IoT. É importante ressaltar que todos estes elementos são *open source*, permitindo a reprodução gratuita em outros contextos, e facilitando eventuais adaptações que se fizerem necessárias. As seções a

seguir detalham os demais elementos característicos desta proposta de solução.

## 1.1 Objetivos

O objetivo geral deste trabalho consistiu em conceber um projeto aberto e de baixo custo de um sistema de controle de acesso a múltiplos ambientes baseado em Internet das Coisas. Para alcançar esta meta, alguns objetivos específicos precisaram ser contemplados, a citar:

1. Realizar a fundamentação teórica dos conceitos de Internet das Coisas;
2. Efetuar o levantamento de tecnologias para implementação dos projetos de *hardware* e *software*;
3. Conceber, implementar e documentar os projetos de *hardware* e *software*;
4. Avaliar a solução desenvolvida.

## 1.2 Justificativa

A solução proposta no escopo deste trabalho contempla o desenvolvimento de um projeto de *hardware* e *software* abertos para controle de acesso na perspectiva de IoT. Neste contexto, permite uma melhor gerência do acesso a espaços físicos, minimizando a intervenção humana presencial para viabilizar este acesso e também as eventuais falhas de segurança, colaborando para um uso mais eficiente.

Além de desenvolver este trabalho, considerou-se organizá-lo e documentá-lo, compartilhando todos os artefatos produzidos livremente em um repositório. Com isto, interessados em reproduzir o projeto podem obter seus elementos descritivos e códigos-fonte de maneira gratuita, minimizando os custos relativos à aquisição de soluções comerciais e permitindo a personalização para um contexto particular. Esta perspectiva foi considerada especialmente visando beneficiar instituições públicas de ensino, onde há uma demanda real por soluções desta natureza, mas com pouco investimento financeiro para tal.

Considerando a perspectiva e o potencial da Internet das Coisas, é essencial colaborar no desenvolvimento de soluções desta natureza, especialmente de código aberto, baixo custo e demanda prática. O trabalho em questão apresenta uma contribuição nesta perspectiva, integrando projeto de *hardware* e *software*, fazendo uso de tecnologias como RFID, MQTT, Python, dentre outras.

Do ponto de vista da discente que o desenvolveu, este trabalho se fez importante e relevante em sua formação, pois integrou conceitos adquiridos em diversas disciplinas do curso de Engenharia da Computação em paralelo às suas aplicações, unindo as vertentes da Computação em termos de *hardware* e *software*. Por fim, o trabalho desenvolvido está alinhado com os objetivos do *Laboratório de Sistemas Inteligentes* da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, especialmente no que diz respeito ao desenvolvimento de soluções de Internet das Coisas.

### 1.3 Metodologia

O processo de desenvolvimento deste projeto foi dividido em seis etapas, de fundamental importância, para balancear e unir teoria e prática a fim de constituir o protótipo a ser apresentado ao final do trabalho. As etapas realizadas encontram-se listadas a seguir:

1. Realizar a fundamentação teórica dos conceitos de Internet das Coisas;
2. Elicitar as funcionalidades a serem providas pelo sistema de controle de acesso;
3. Efetuar o levantamento de tecnologias para implementação do projeto de *hardware* e *software* do sistema de controle de acesso;
4. Conceber, implementar e documentar os projetos de *hardware* e de *software*;
5. Avaliar a solução desenvolvida;
6. Disponibilizar a solução desenvolvida.

## 1.4 Organização do Documento

Para apresentação deste trabalho de conclusão de curso, este documento está organizado como segue. No Capítulo 2 são contemplados os fundamentos teóricos necessários para a elaboração do projeto, incluindo conceitos de Internet das Coisas, Família ESP8266 e o *NodeMCU*, Identificação de Rádio Frequência (RFID), protocolo de comunicação MQTT e *Framework Web2py*. Neste capítulo também é realizada uma análise dos trabalhos relacionados, descrita na Seção 2.6. A concepção da solução proposta para o controle de acesso a múltiplos ambientes baseada em Internet das Coisas é detalhada no Capítulo 3. Em seguida, no Capítulo 4, são descritos os resultados obtidos da implementação desta solução, que incluem a prototipagem do *hardware* e passo a passo da implementação de todos os componentes de *software*. As considerações finais deste trabalho são discutidas no Capítulo 5. Alguns apêndices complementam os capítulos anteriores e serão referenciados ao longo do texto.

# Capítulo 2

## Fundamentação Teórica

Levando em consideração o inter-relacionamento entre os conceitos de tecnologias e suas aplicações práticas no projeto proposto, esta seção está dedicada à apresentação dos conceitos teóricos e tecnológicos que a fundamentam, a citar: Internet das Coisas, Família ESP8266 e hardware *NodeMCU*, Identificação de Rádio Frequência, Protocolo MQTT, além do *Framework* Web2py. Estes conceitos foram necessários para a compreensão do funcionamento de cada tecnologia, possibilitando assim um melhor entendimento do papel desempenhado por elas na solução proposta como um todo, como expresso nas subseções a seguir.

### 2.1 Internet das Coisas

Embora não haja uma definição única, entende-se *Internet das Coisas* (IoT, do inglês *Internet of Things*) como um domínio de aplicação que integra diferentes tecnologias e aspectos sociais. Por meio da interconexão de objetos inteligentes, esse sistema tecnológico visa enfatizar, além do monitoramento e controle, os processos de otimização e autonomia destes objetos (IEEE, 2015). IoT é oriunda da evolução da comunicação Máquina-a-Máquina (M2M) e, com a perspectiva de conectar objetos à Internet, ampliam-se as possibilidades de coleta de informações, interação destes objetos entre si e com as pessoas, as quais podem controlá-los remotamente com base nas informações recebidas, criando novos padrões no âmbito social e organizacional (PEREIRA; CARVALHO, 2017).

A definição das “coisas” ou objetos em IoT é muito ampla e inclui uma grande variedade de elementos físicos como *smartphones*, *tablets* e elementos dos ambientes do cotidiano que são equipados com *tags* RFID ou outros, conectados por um dispositivo de *gateway*. Assim, IoT acontece quando tais objetos passam a fazer parte da Internet, podendo ser endereçados unicamente e acessados livremente, realizando processamento, emitindo informações para outros dispositivos ou servidores, recebendo informações, atuando a partir das informações recebidas e sendo incrementados com serviços ou inteligência. Neste contexto, a conectividade considerada deve ser a qualquer hora, em qualquer lugar, para qualquer um e em qualquer coisa (LONGO, 2015).

Estima-se que 50 bilhões de dispositivos estejam operantes com IoT até o ano de 2020, permitindo assim a interação humano-máquina e também máquina-máquina de maneira inteligente (SCHWARTZ, 2016b). Com isso, IoT pode proporcionar grande impacto na economia e na produtividade com ganhos gerados pela otimização e automação de processos e pela diminuição da necessidade de intervenção humana. No Brasil, em particular, o impacto econômico da IoT é promissor, pois há a perspectiva da geração de 2 milhões de novos empregos criados a partir dessa nova tecnologia, com o provável aumento no Produto Interno Bruto brasileiro para mais de 122 bilhões de reais em 10 anos (PEREIRA; CARVALHO, 2017).

O potencial de domínios em que IoT pode ser aplicada é vasto e pode, por exemplo, revolucionar as cidades, os carros, as residências, a agricultura, a saúde e tantos outros contextos. Todavia, desafios relacionados à segurança, privacidade, grande volume de dados, interoperabilidade e escala ainda precisam ser endereçados para que tais soluções se tornem viáveis no cotidiano das pessoas (PEREIRA; CARVALHO, 2017).

## 2.2 Família ESP8266 e o NodeMCU

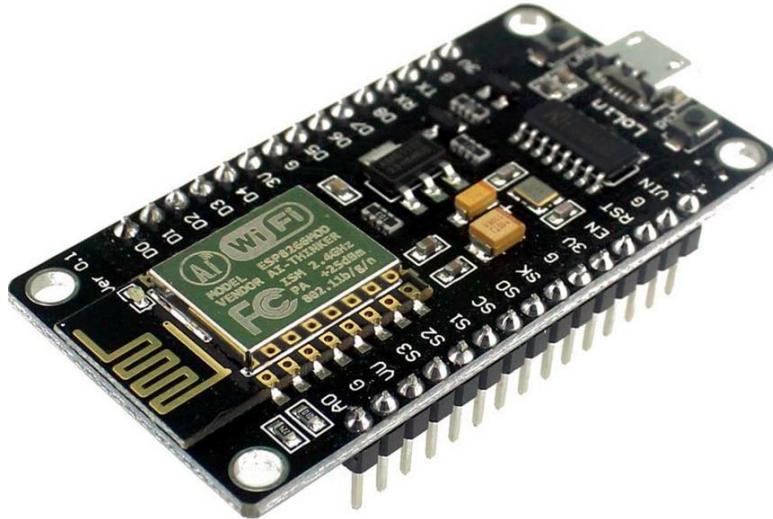
A família ESP8266 engloba um grupo de microcontroladores que possuem características de *hardware* e *software* necessárias para realizar a conexão à Internet. Por esse motivo, a utilização desses dispositivos está cada vez mais popular entre projetos de IoT (MORAIS, 2017). Esses microcontroladores, fabricados pela empresa chinesa *Espressif*, podem ser encontrados na forma

*standalone* ou inclusos em placas de desenvolvimento, como o *NodeMCU* ou *WeMos D1*, com diferentes configurações de pinagem, antenas *Wi-fi*, ou uma quantidade diferente de memória flash na placa, por exemplo (P, 2017). A escolha do módulo normalmente é feita levando em conta as necessidades do projeto em que será utilizado.

A placa *NodeMCU*, em particular, é ilustrada na Figura 2.1. Ela compõe uma plataforma *open-source* para desenvolvimento de projetos de IoT que combina o chip ESP8266, uma interface USB-Serial e um regulador de tensão 3,3 V. Algumas outras características desta placa, considerando a terceira versão, são elencadas a seguir:

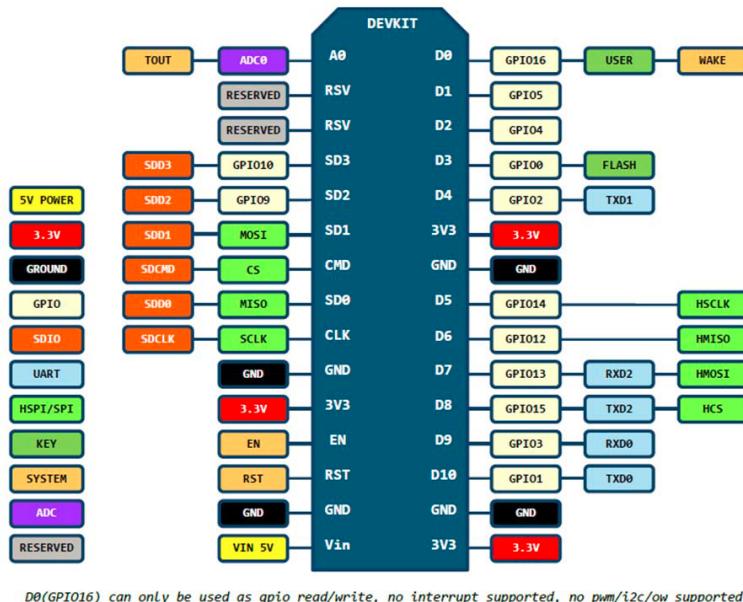
- Corrente de operação:  $270mA$ ;
- Processador: ARM 32 bits;
- Controlador: ESP8266 ESP-12E;
- Velocidade de processamento:  $80 \sim 160MHz$
- Conectividade: *Wi-fi* integrada à placa;
- Protocolos *Wi-fi* suportados: 802.11 b/g/n
- Características de protocolo: Pilha TCP/IP integrada
- Características de conexão: máximo de 5 conexões TCP simultâneas;
- Entrada e Saída: UART, I2C, PWM, GPIO, 1 ADC;
- GPIOs: 17 (multiplexadas com outras funções);
- Analógico para Digital: 1 entrada com resolução de 1024 passos (EINSTRONIC, 2017).

Figura 2.1: Plataforma de desenvolvimento *NodeMCU*. Fonte: (TECHNOLOGY, 2017).



A Figura 2.2 ilustra o esquema de pinos do *NodeMCU*, dos quais é possível identificar duas fileiras e um total de 30 pinos, representados por cores e legendas conforme suas funções e funcionalidades, os quais são detalhados no *datasheet* disponibilizado pelo fabricante, vide (ESPRESSIF, 2018). Conforme documentação, esta plataforma é especialmente indicada para aplicações de automação residencial, de controle inteligente de acionamento de interruptores e lâmpadas, redes de sensores, dispositivos *wearables*, controle industrial sem fio, monitoramento de bebês, dentre outros.

Figura 2.2: Esquema de pinos na plataforma *NodeMCU*. Fonte: (ESPRESSIF, 2018).



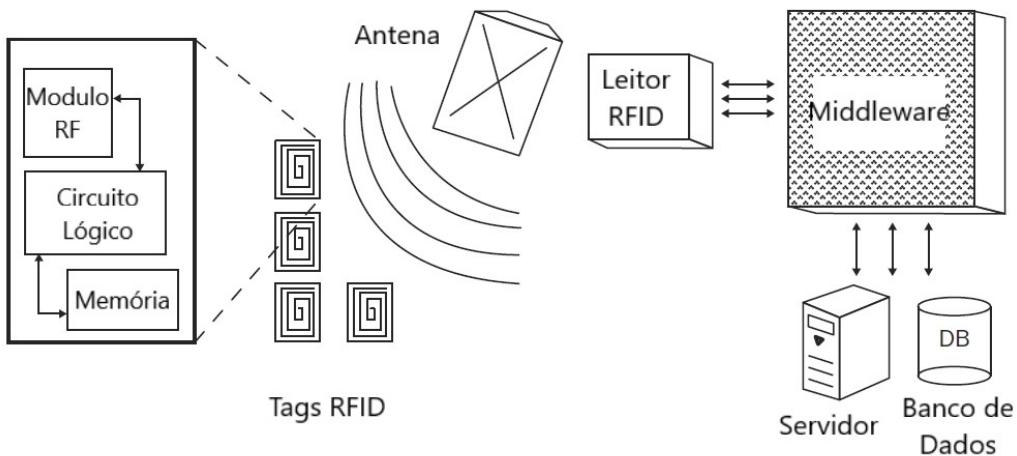
Além de especificações de *hardware*, é importante enfatizar os aspectos de *software*, especialmente relativos à programação da plataforma. O controle pode ser realizado via comandos AT, linguagem Lua ou até pela própria IDE do Arduino estabelecendo a comunicação via cabo micro-USB (TECHNOLOGY, 2017; THOMSEN, 2016).

## 2.3 Identificação de Rádio Frequência

A *Identificação de Rádio-Frequência* (RFID, do inglês, *Radio-Frequency Identification*) descreve qualquer sistema de identificação no qual um dispositivo eletrônico, de maneira *wireless*, seja por rádio-frequência ou variações de campo magnético, comunica-se com leitores, considerando uma certa distância (GLOVER; BHATT, 2006; CHEN; CHEN, 2016). Esta tecnologia tem sido de considerável relevância, pois emerge como substituta de outras tecnologias de identificação, a exemplo do código de barras (II, 2005).

Um sistema RFID é composto por três componentes básicos: (i) um transponder programável, chamado de *tag*; (ii) um leitor; e (iii) um servidor (II, 2005). Essa configuração permite que *tags* sejam identificadas, entretanto, para aplicações com manipulação de dados, filtragem, dentre outros processos, é necessário que o sistema RFID interaja também com um *middleware* e um banco de dados, por exemplo. A Figura 2.3 ilustra como esses componentes se relacionam em uma aplicação RFID mais completa.

Figura 2.3: Transmissão de dados usando RFID. Adaptada de: (PANDIAN, 2010).



As *tags* são o coração de um sistema RFID porque elas armazenam a informação que descreve o objeto a ser rastreado. Informações específicas dos objetos são armazenadas na memória das *tags* e acessadas por meio de sinais de rádio de leitores RFID. *Tags* que executam o transporte de dados são compostas por um circuito lógico com capacidades computacionais, memória para armazenamento de dados, e um módulo de rádio frequência, que consiste em uma antena para comunicação. Esses componentes são juntamente empacotados e podem ser anexados a objetos (PANDIAN, 2010).

Os leitores RFID, por sua vez, são compostos por um módulo de rádio-frequência, uma unidade de controle e um conjunto de antenas que interrogam as *tags* RFID por meio da comunicação de rádio-frequência, é importante mencionar que alguns leitores RFID disponíveis atualmente já possuem antenas embutidas a seu *hardware*. Eles são responsáveis por executar uma variedade de funções, como ativação de *tags* mediante envio de sinais de consulta de maneira a fornecer energia a *tags* passivas, codificação de sinais de dados a serem enviados à *tag*, e por fim, decodificação de dados recebidos de *tags*. Além da comunicação com a *tag*, o leitor também é responsável por comunicar-se a um servidor. Dessa maneira, as informações enviadas pela *tag* e recebidas pelo leitor devem ser demodularizadas e decodificadas para então serem enviadas ao servidor. Essa comunicação pode ser feita fazendo uso do protocolo TPC/IP, quando realizada por meio da Internet (LEHPAMER, 2012).

De maneira geral, o servidor é responsável pelo armazenamento de dados, processamento e coordenação de informações, possibilitando a execução de funções de alto desempenho. Para realizar essas funções adequadamente, os servidores fazem uso de *middlewares*, que conectam os dados enviados pelo leitor aos programas de processamento presentes nos servidores, esses dados podem ser armazenados em bancos de dados para análises posteriores. O *middleware* proporciona uma interface estável e coerente entre as operações de *hardware* RFID e o fluxo de elementos de dados como o código eletrônico do produto (EPC), em sistemas de banco de dados de estoque, vendas, compras, *marketing* e similares, distribuídos por toda a gerência de uma empresa (CHEN; CHEN, 2016; II, 2005).

Assim como as tecnologias de código de barra e biometria, a tecnologia RFID pode ser

empregada nos cenários denominados de “autoid”, ou seja, de identificação automática, capazes de descrever qualquer sistema automático que anexa uma identidade a um objeto, sendo utilizados, principalmente em Sistemas de Localização em Tempo-Real. Embora não exaustiva, pode-se mencionar cinco grandes categorias de possíveis usos do RFID:

1. **Controle de Acesso.** Considera as aplicações envolvendo sistema RFID que visam garantir acesso seletivo a certas áreas;
2. **Identificação e Envio.** Consiste nos sistemas mínimos de RFID que permitem que o usuário associe uma *tag* RFID a um objeto, anexando fisicamente a *tag* a ele, e então verifique se esse identificador opera adequadamente enquanto anexado;
3. **Identificação de Paletes e Caixas.** É uma das formas mais comuns de aplicação RFID, que consiste basicamente na identificação de unidades de remessas de produtos com utilização de placas de licenciamento compostas por um ou mais tipos de itens;
4. **Localização e Rastreamento.** É uma das primeiras formas de aplicações de sistemas RFID, onde há identificação de itens através das *tags*, de maneira que elas contenham informações específicas acerca desses objetos para que seja feita uma estrita monitoração dos mesmos;
5. **Prateleiras Inteligentes.** Consiste em um conjunto de prateleiras, ou alguma outra estrutura de armazenamento, que monitora constantemente os itens individuais que ela contém, de forma que se um item for removido ou adicionado, a prateleira atualiza imediatamente o inventário (GLOVER; BHATT, 2006).

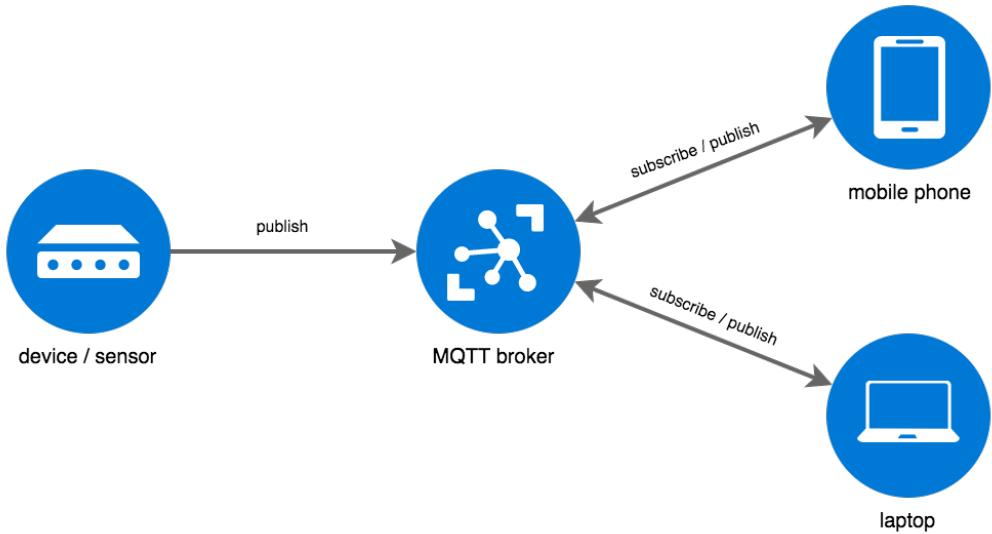
Diversas aplicações práticas para controle de acesso com RFID, em particular, já se encontram consolidadas. *Tags* podem, por exemplo, identificar automóveis permitindo o acesso a determinadas áreas de estacionamento (GARCIA, 2013); sob a forma de cartão, *tags* podem ser utilizadas por pessoas para acesso em determinadas áreas hospitalares (GODOY, 2011), dentre outras. Aplicações comerciais já consideram a utilização de RFID em áreas prediais ou aplicações de segurança também no contexto de controle de acesso (SOLUTIONS, 2011).

Apesar das vantagens mencionadas, assim como qualquer tecnologia, o sistema RFID possui limitações, e essas variam de acordo com a aplicação em uso. As limitações mais comuns no uso da tecnologia estão relacionadas a quatro principais características: (i) à tecnologia em si; (ii) à ausência de padrões para operação da tecnologia; (iii) a custos; e (iv) à integração (PRADO; PEREIRA; POLITANO, 2006).

## 2.4 MQTT

O MQTT (do inglês, *Message Queuing Telemetry Transport*) é um protocolo de conectividade máquina-a-máquina (M2M), muito utilizado em aplicações de Internet das Coisas, caracterizando-se pelo transporte extremamente leve de mensagens através do mecanismo *publish-subscribe* (em tradução literal, publicação-assinatura). Esse protocolo opera na camada de aplicação, sob os protocolos TCP e IP. (SAHOO, 2018). Um exemplo de arranjo entre dispositivos comunicantes com este protocolo é ilustrado na Figura 2.4.

Figura 2.4: Exemplo de comunicação M2M com o protocolo MQTT intermediada por um *broker*. Fonte: <<http://goo.gl/s1mBTk>>.



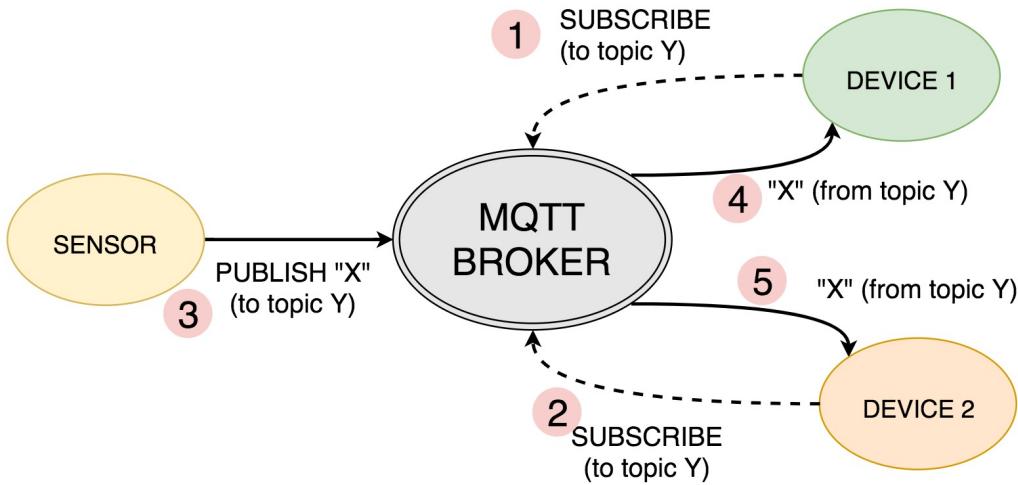
O MQTT é análogo ao HTTP, mas com algumas diferenças, tais como o tamanho do *payload* (mensagem enviada) e a maneira de aquisição de dados por parte do cliente, por exemplo. No HTTP, o *payload* é maior, o que inviabiliza o seu uso em conexões de baixa qualidade, enquanto

o protocolo MQTT funciona bem quando redes não confiáveis estão envolvidas e a conectividade é intermitente. Além disso, no MQTT os clientes não precisam realizar requisições sempre que desejarem receber informações, pois o *broker* as envia automaticamente, sempre que houverem atualizações de dados disponíveis (HILLAR, 2017; SAHOO, 2018).

Para melhor entender o funcionamento do protocolo *MQTT*, é necessário compreender a forma de comunicação estabelecida por ele, conforme o padrão *publish-subscribe*. Nesse padrão de comunicação, os agentes comunicantes são conhecidos como clientes e eles podem exercer o papel de publicadores ou de assinantes. O cenário padrão dessa comunicação é definido pela publicação de uma mensagem por um cliente *publisher*, que especifica o conteúdo da mensagem a ser enviada e o tópico em que ela será publicada; e um ou mais clientes *subscribers*, assinantes do tópico, irão receber a mensagem quando disponíveis. Um ponto importante dessa relação entre clientes *publishers* e *subscribers* é que eles são desacoplados uns dos outros, o que significa que o cliente *publisher*, por exemplo, não tem conhecimento da identidade dos *subscribers* que estão recebendo sua mensagem, assim como estes não sabem quem está provendo a informação recebida (HILLAR, 2017).

Essa transparência aos clientes, só é possível por conta do servidor intermediador utilizado nesse protocolo, o *broker*. Ele é o ponto central de comunicação no protocolo MQTT, pois está encarregado de receber e distribuir todas as mensagens pareadas entre os clientes publicadores e os devidos assinantes, conforme ilustrado na Figura 2.5. Cada mensagem publicada ao *broker* contém um tópico em seu escopo, e por meio dele, o *broker* saberá a rota de informação correta para que a mensagem seja devidamente entregue aos assinantes do tópico referenciado. Por meio desse processo de filtragem de mensagens, o *broker* assegura que clientes *subscribers* recebam apenas mensagens de seu interesse. As filtragens podem ser baseadas em alguns diferentes critérios dependendo da implementação do servidor MQTT. No cenário padrão, exemplificado anteriormente, em que utiliza-se o tópico para correta distribuição de mensagens, define-se a filtragem como baseada em tópico, ou, baseada em assunto como relata a literatura (HILLAR, 2017).

Figura 2.5: Exemplo detalhado do padrão *publish-subscribe* no MQTT. Fonte: <<https://goo.gl/ivnm4>>.



Um exemplo prático da utilização do protocolo MQTT pode ser visto em ambiente industriais, por exemplo, como o apresentado por Correa e outros (CORREA et al., 2016). Os autores fazem uso de sensores e autuadores como cliente-publicadores, de maneira a gerar dados e publicá-los ao *broker*. Cada sensor possuía seu tópico de envio definido, para que os dados pareados pudessem alcançar os clientes-*subscribers* corretamente. Na situação apresentada, um sistema supervisório é dedicado a receber as informações, inscrevendo-se no tópico correspondente à informação desejada a quem o *broker* distribuirá os dados recebidos (CORREA et al., 2016).

Como mencionado anteriormente, o cliente *subscriber* receberá a mensagem assim que disponível, o que significa dizer que a operação de publicação não está sincronizada com a operação de recebimento. Essa característica permite que o *publisher* continue a enviar mensagens de maneira não bloqueante, mesmo que os assinantes do tópico não tenham recebido. No entanto, é válido ressaltar a possibilidade de também realizar publicações ao *broker* de maneira síncrona e continuar a execução somente após a operação ter sido bem-sucedida. De maneira geral, a forma de comunicação, síncrona ou assíncrona, irá depender do tipo de solução que está sendo implementada e da necessidade abordada no problema (SAHOO, 2018).

Considerando o crescente desenvolvimento de soluções ligadas à Internet das Coisas, há uma quantidade razoável de bibliotecas e tecnologias para implementação de clientes MQTT

e para escolha do servidor *broker*. Algumas características como a plataforma, orçamento disponível (algumas versões são pagas) e linguagem de programação devem ser consideradas nesta escolha, juntamente com aspectos que visem o melhor provimento das funcionalidades necessárias projeto que se deseja desenvolver (HILLAR, 2017).

## 2.5 **Framework** Web2py

Web2py é um *framework* gratuito e de código aberto cujas características propiciam o desenvolvimento ágil de aplicações web seguras e baseadas em bancos de dados (PIERRO, 2010). Dentre as características relevantes deste *framework*, destacam-se: (i) a estrita relação com a linguagem Python; (ii) desenvolvimento *full-stack*; (iii) utilização do padrão de projeto *Model-View-Controller* (MVC).

A característica mais elementar deste *framework* é a linguagem de programação utilizada em sua escrita e desenvolvimento, o Python. Python é uma linguagem de programação de alto nível, interpretada e de script, funcionando muito bem para servidores web. Qualquer arquitetura computacional que suporte Python pode ser usada para o desenvolvimento com o Web2py, não necessitando qualquer instalação ou configuração além do download do próprio pacote do *framework* (PIERRO, 2010; DUARTE, 2017).

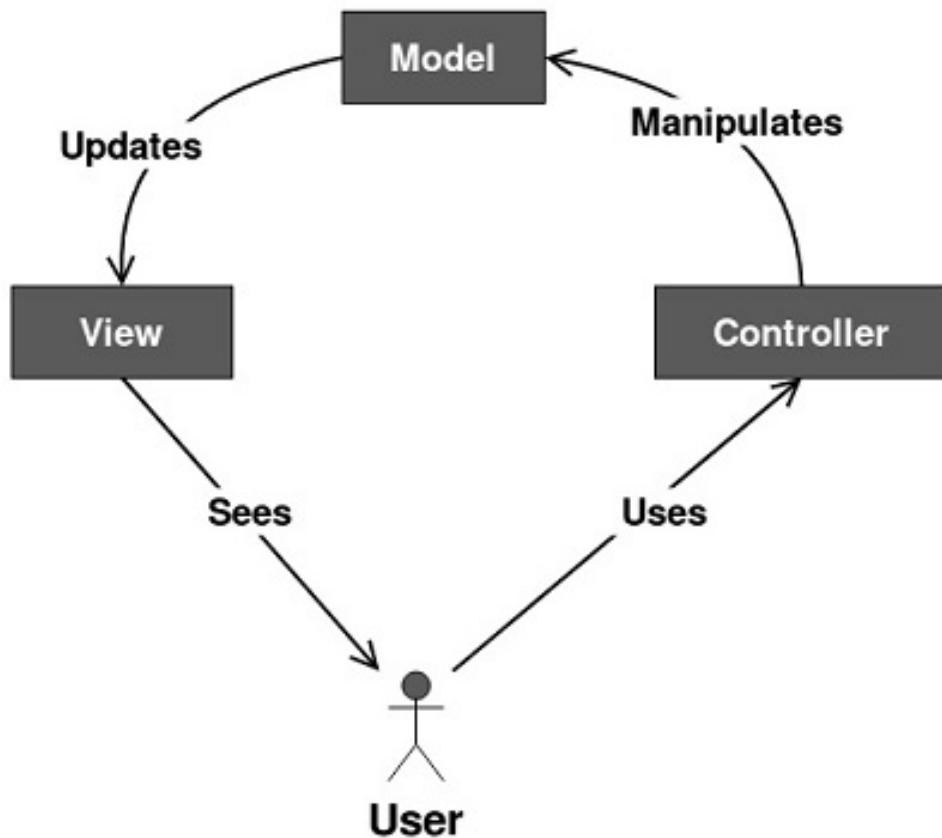
A característica *full-stack* desse *framework*, por sua vez, diz respeito à facilidade de desenvolver uma aplicação web por completo desde a estruturação do seu banco de dados (*back-end*) ao acesso e desenvolvimento visual da aplicação (*front-end*). O Web2py contém todos os componentes necessários para a construção de uma aplicação web funcional, pois, ao realizar o download do pacote Web2py, o usuário já possui um ambiente pronto para o desenvolvimento, com todos os módulos funcionando, e são eles: o Servidor web (*Rocket WSGI Web Server*), banco de dados (*SQLite*) e a IDE web (acessível pelo navegador) (PIERRO, 2010; RIBEIRO, 2017).

O desenvolvimento ágil desse *framework* está diretamente relacionado à percepção de comportamentos padrões no processo de codificação de aplicações web e a redução de esforços para novas aplicações através da modularização, reusabilidade, extensibilidade e inversão de controle

proporcionados, como por exemplo a construção de formulários, validação de dados, conexão com o banco de dados, entre outras ações de gerenciamento.

Dessa maneira, tem-se a utilização do padrão (ou modelo) MVC como característica relevante no desenvolvimento de projetos com Web2py, visto que esse padrão de projeto vem como uma forma de melhor organizar o código, cooperando também à agilidade do processo de desenvolvimento (DUARTE, 2017). O modelo MVC é um padrão de arquitetura de *software* que divide o desenvolvimento da aplicação em camadas, conforme sugere a sigla e como demonstra a Figura 2.6.

Figura 2.6: Especificação do modelo MVC. Fonte: <<https://bit.ly/2ptI7wz>>.



No MVC, a camada *Model* é responsável por tratar os dados que serão armazenados ou lidos do banco de dados e exibidos na tela, ou seja, qualquer comando (leitura, alteração, exclusão, inserção, criação, ou outros) que envolva dados armazenados, será escrito nessa camada. A segunda camada, *Controller*, é definida pela concentração da lógica de programação, tornando

mais limpa as demais camadas. Dessa maneira, tal camada é responsável pelas funcionalidades de manipulação e transferência de dados. A terceira camada, por sua vez, é aquela que formata os dados para o usuário e recebe comandos desse usuário, ou seja, a camada *View* é responsável pela interação com o usuário e, portanto, é nela onde ficarão os códigos HTML e CSS. É válido reiterar a utilização da linguagem Python em todas as camadas da aplicação, na camada *View*, especificamente, utilizada em conjunto com as linguagens comuns ao desenvolvimento *front-end* (DUARTE, 2017).

Além das características mencionadas anteriormente, é válido apontar ainda, a relevância que o *framework* provê à segurança dos dados manipulados por ele, destacando a camada de abstração do banco de dados, conhecida como DAL (*Database Abstraction Layer*), que elimina a possibilidade de *SQL Injection*. Dentre outras prevenções proporcionadas pela camada DAL, tem-se a validação de formulários e bloqueio dos mesmos contra ameaça de falsificação de solicitação entre sites (*Cross Site Request Forgeries*), prevenção contra *Cross Site Scripting*, armazenamento de senhas como *hashes*, dentre outras funcionalidades que proporcionam segurança ao usuário e facilitam essa tarefa ao desenvolvedor (PIERRO, 2010; RIBEIRO, 2017).

## 2.6 Trabalhos Relacionados

Soluções de controle de acesso com IoT e RFID já são presentes na literatura, enfatizando não apenas o monitoramento e controle de acesso por meio de objetos inteligentes, mas também colaborando para soluções mais otimizadas e autônomas. Algumas destas soluções são descritas e discutidas a seguir.

O trabalho de Araújo e outros (ARAÚJO et al., 2016) apresenta um sistema de monitoramento de *tags* RFID cujos dados cadastrados e informações persistidas encontram-se em um banco de dados online. Em termos de *hardware*, o objeto inteligente proposto é composto por um microcontrolador da família PIC, *display LCD*, leitor RFID, *buzzer* e relé, além de *software* embarcado Linux baseado em *Raspberry Pi* para controle lógico de suas funcionalidades. Os autores desenvolveram funcionalidades de cadastro de *tags* RFID, envio de informações (Data, Horário e ID) em caso de *tags* previamente cadastradas e, como resposta interativa ao usuário,

o acionamento de relé, LEDs, recebimento de sinal sonoro através do *buzzer* e mensagem pelo *display LCD*. Os autores evidenciaram como resultados positivos o monitoramento das *tags* e a responsividade do sistema ao usuário, além de estratégias adotadas para contornar problemas de conexão com a internet, evitando fragilidades ao sistema.

De maneira análoga, a solução de Teixeira (TEIXEIRA, 2011) propõe um sistema de controle de fluxo de pessoas com RFID para fins de registro de ponto de colaboradores de uma empresa, incluindo a geração de relatórios. As funcionalidades desta solução foram divididas em dois *softwares* desenvolvidos pelo autor, sendo um para monitoramento de informações adquiridas pelo leitor RFID e tratamento destas para persistência em banco de dados; e outro que permite a interação do usuário com o sistema, para gerenciamento e acesso direto ao conteúdo do banco de dados. Em ambos, a linguagem Java e *plugins* compatíveis a ela foram utilizados, sendo produzida uma aplicação *Desktop* para o *software* de gerenciamento.

No projeto de Teixeira foram utilizados elementos de *hardware* mais robustos, tais como um microcomputador *Intel Celeron D CPU*, que permite conexão com o leitor RFID por meio de porta serial RS-232, um leitor RFID modelo DE210-R2/C com antena integrada e *Tags* RFID do modelo *TC Clamshell Personalité* do tipo passiva encapsuladas em cartão. O foco do trabalho desenvolvido foi, portanto, a implementação da aplicação, pondo a teste a interação do usuário com o sistema e a responsividade do sistema na geração de relatório, sendo ambos atingidos com sucesso.

O projeto de Fonseca e outros, por sua vez, levou em conta a proposição de um sistema de controle de acesso motivado pela necessidade de segurança no acesso a determinadas áreas de uma empresa ou residência (FONSECA; SILVA; MORAES, 2017). Estes autores efetuaram a implementação de um protótipo avaliado como eficaz e de baixo custo utilizando a plataforma de desenvolvimento Arduino. Para o projeto de *hardware* os autores também utilizaram em conjunto um módulo leitor RFID, fechadura elétrica Intelbras modelo FFX1000, que é ativada através de um módulo relé, e um módulo Ethernet Shield W5100 que permite a comunicação cliente/servidor do protótipo com o banco de dados utilizando o protocolo TCP/IP.

Como identificador único de usuário, foram utilizadas *tags* RFID encapsuladas em forma

de cartão, cujas informações passavam por processo de validação e o retorno do sistema se fazia visível a partir do monitor serial da plataforma Arduino, indicando liberação ou retenção de acesso. Além do projeto de *hardware*, os autores desenvolveram um projeto de *software* utilizando o servidor Xampp e linguagem PHP, para realização de cadastro de *tags* no sistema e responsável por realizar a integração e gerenciamento web do banco de dados. De acordo com os autores, este projeto alcançou êxito em seu objetivo, proporcionando um dispositivo seguro para acesso a ambientes restritos a partir da validação de *tags* RFID, sendo, sobretudo, de baixo custo.

O sistema de reserva de salas e controle de entrada e saída proposto por Almeida et al. aborda o desenvolvimento de um projeto envolvendo o conceito de IoT e identificadores RFID/NFC (*Near Field Communication*), a partir da utilização de um protótipo de *hardware* para controle de ambientes (ALMEIDA et al., 2018). Os componentes utilizados para a estrutura física do projeto foram uma placa Arduino, um módulo PN532 para leitura RFID e NFC e um sistema de transmissão de dados. O *hardware* desenvolvido é responsável por enviar solicitações de acesso via *Web Services* à aplicação, e esta, encarregada de validar a solicitação a partir de informações pré-cadastradas no Banco de Dados.

O projeto proposto por Almeida e colaboradores enfatiza as regras de negócio estabelecidas na aplicação Web onde são definidas as restrições para agendamento de salas. Essa aplicação é desenvolvida com HTML, CSS e Javascript em sua estrutura *front-end* e PHP em sua estrutura *back-end*, além de utilização de Banco de Dados MySQL para persistência de dados, sugerindo a automatização de controle de acesso e evidenciando sua utilização em um ambiente de agendamento escolar.

Considerando os trabalhos afins já existentes na perspectiva de sistemas de controle de acesso baseados em IoT e a solução proposta que está sendo desenvolvida no escopo deste trabalho, é natural que esta tenha similares, diferenças, vantagens e também limitações. Em termos análogos, citam-se tecnologias similares, baseadas em Ambiente Arduino, e uma aplicação web para integração das informações. Apesar destas semelhanças, distingue-se no tocante à utilização do protocolo MQTT para comunicação entre o protótipo e o orquestrador de dados,

que serão detalhados na seção seguinte, a interatividade do protótipo e o usuário a partir do visor LCD, e a existência de uma aplicação web provendo relatórios de acesso e agendamentos detalhados das salas. Além do que foi mencionado, a natureza aberta do projeto permite a sua livre reprodução e modificação.

A solução proposta no escopo deste trabalho, diferentemente da literatura relacionada, mostra-se especialmente adaptável no controle de diferentes ambientes, permitindo escalabilidade no número de protótipos existentes. Além disso, é especialmente voltada para uso em âmbito acadêmico e escolar, auxiliando na tarefa cotidiana de gerenciar estes espaços e zelar pela sua conservação. Dessa maneira, ressalta-se sua importância no contexto de novas soluções no âmbito de Internet das Coisas e enfatiza-se a possibilidade de contribuições da comunidade *open-source* ao projeto como um todo, em termos de *hardware* e *software*, visto que é aberto e de baixo custo.

# Capítulo 3

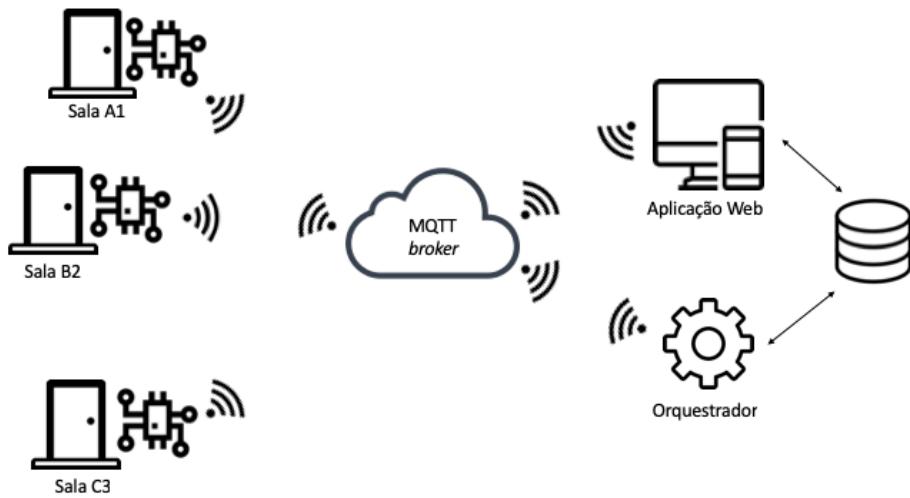
## Solução Proposta

Esta seção tem por objetivo apresentar o detalhamento dos elementos que compõem a solução proposta para o cenário endereçado. Para tanto, inicialmente tem-se uma visão geral da solução proposta na Seção 3.1, seguido dos diagramas de caso de uso elaborados para elicitação dos requisitos, detalhados na Seção 3.2. O projeto de *hardware* é apresentado na Seção 3.3 e, posteriormente, são apresentados os elementos do projeto de *software* na Seção 3.4.

### 3.1 Visão Geral da Solução Proposta

A solução idealizada neste trabalho, caracterizada por um sistema de controle de acesso a múltiplos ambientes baseado em Internet das Coisas, é composta essencialmente por quatro elementos principais: (*i*) projeto de *hardware* dos objetos inteligentes e (*ii*) *software* embarcado dos objetos inteligentes; (*iii*) o orquestrador de dados de saída dos objetos inteligentes e (*iv*) painel administrativo, composto por uma aplicação web. Estes dois últimos elementos conectam-se à um banco de dados, responsável pela persistência dos dados deste sistema de controle de acesso. A Figura 3.1 ilustra a ideia geral da disposição dos elementos no sistema proposto, apresentando também setas direcionais que elicitam o fluxo de informações e as relações existentes.

Figura 3.1: Ideia geral da solução proposta. Fonte: Própria (2018)



A ideia geral da solução proposta consiste essencialmente na proposição do projeto de um objeto inteligente que deve ficar em cada sala a ser controlada. Este objeto, apresentado no escopo deste trabalho na condição de protótipo, deve ser instalado em cada sala, garantindo o acesso do usuário responsável por meio de uma *tag* RFID em horário previamente agendado. Este objeto é munido de um *display*, para que o usuário seja capaz de realizar a inspeção visual dos dados recebidos (acesso autorizado ou não, por exemplo) e para auxiliar no cadastro de novas *tags*. Além de controlar o acesso às salas, quando estas são liberadas o objeto inteligente também ativa a alimentação elétrica de um dispositivo interno da sala, podendo ser, por exemplo, um *datashow* ou ar-condicionado. As seções a seguir irão detalhar o projeto de *hardware* e *software* deste objeto.

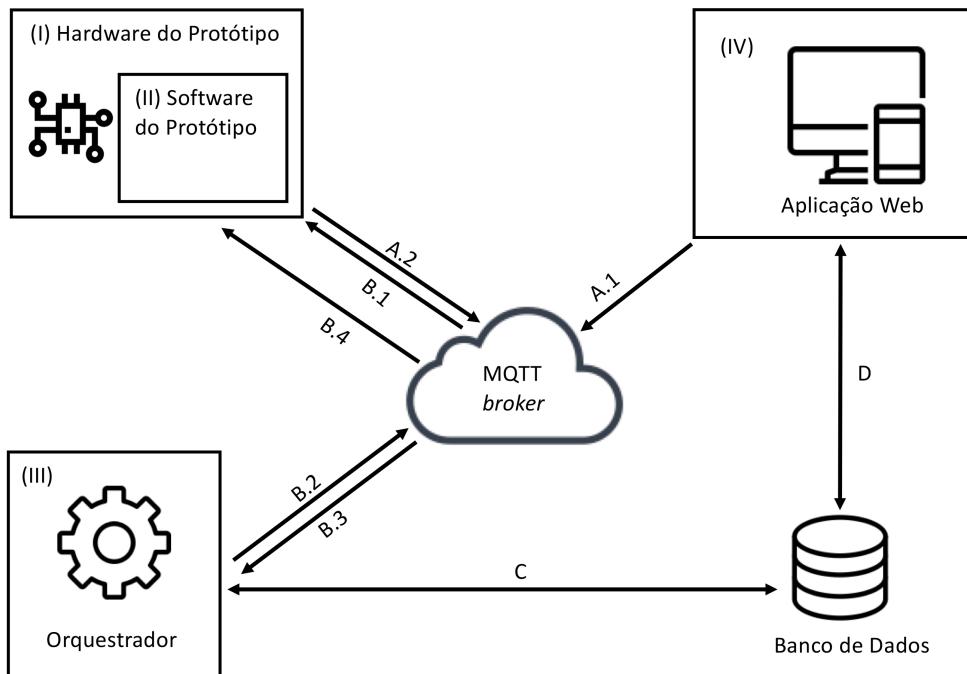
É especialmente interessante ressaltar que o objeto inteligente não é único no escopo do solução. Pelo contrário, várias instâncias desse objeto inteligente coexistem, cada qual controlando o acesso de um ambiente específico. O projeto de *hardware* e a padronização na nomenclatura destes objetos permite a sua utilização simultânea e sem conflitos no monitoramento de diversas salas.

Em termos de *software*, tem-se que a aplicação web e o orquestrador de dados da saída dos objetos inteligentes consistem nos responsáveis pela administração remota dos diversos objetos

inteligentes existentes. Considerando o contexto de IoT, em que estes objetos comunicam-se com a Internet, considera-se a utilização de um *broker* MQTT como intermédio para troca de informações, respeitando o protocolo *publish/subscribe*.

Detalhando um pouco mais o fluxo de informações na solução proposta, conforme Figura 3.2, tem-se primeiramente a aplicação web, que trata-se de um painel administrativo no qual há as funcionalidades para cadastrar usuários, estabelecer horários de utilização e afins, fazendo, para tanto, sucessivas consultas ao banco de dados (vide D). Para finalização de cadastros de usuários, ou de administrador, necessita-se realizar associação de *tag* ao seu perfil. Para tanto, a aplicação web publica no tópico a ação correspondente para o objeto disponível, habilitando a funcionalidade desejada (vide A.1). Este tópico é então repassado ao objeto pelo *broker* MQTT (vide A.2) e ativa a leitura de *tag* específica para cadastro no respectivo objeto. Além do que foi exposto, a aplicação web provê relatórios aos usuários acerca da pontualidade no acesso a cada sala, auxiliando não só no gerenciamento do acesso, mas oferecendo também uma informação de mais alto-nível característica de soluções no âmbito de Internet das Coisas.

Figura 3.2: Esquema de fluxo de informação e componentes da solução proposta.



Quando o objeto é utilizado para cadastrar novas *tags*, tem-se essencialmente um fluxo de

saída de dados que precisa ser endereçado apropriadamente. A *tag* é lida e publicada num tópico específico (vide B.1), o *broker* encaminha esta publicação para o orquestrador (vide B.2), que persiste as informações no banco (vide C), retorna os parâmetros e publica no tópico, indicando o sucesso ou não da operação (vide B.3), que é finalmente repassada ao objeto para exibição no *display* que será visualizado pelo usuário (vide B.4). Diante dessa necessidade, o *software* embarcado no objeto é responsável por abrir um canal de comunicação utilizando o protocolo MQTT e definir os tópicos de saída, facilitando esta comunicação.

A comunicação dos módulos é realizado pela Internet, e suas trocas de dados são baseadas no protocolo MQTT, cuja topologia foi descrita anteriormente na Seção 2.4. O *broker* MQTT utilizado, *CloudMQTT*, encontra-se na nuvem e permite, portanto, o controle remoto dos ambientes associados aos objetos. O conjunto formado por esses objetos inteligentes distintos compõe uma solução para melhor gerenciamento de acesso a espaços físicos, facilitando o acesso dos usuários autorizados e coletando métricas acerca da pontualidade dos mesmos. O projeto de cada elemento integrante da solução proposta será minunciosamente descrito nas seções subsequentes.

## 3.2 Diagramas de Caso de Uso

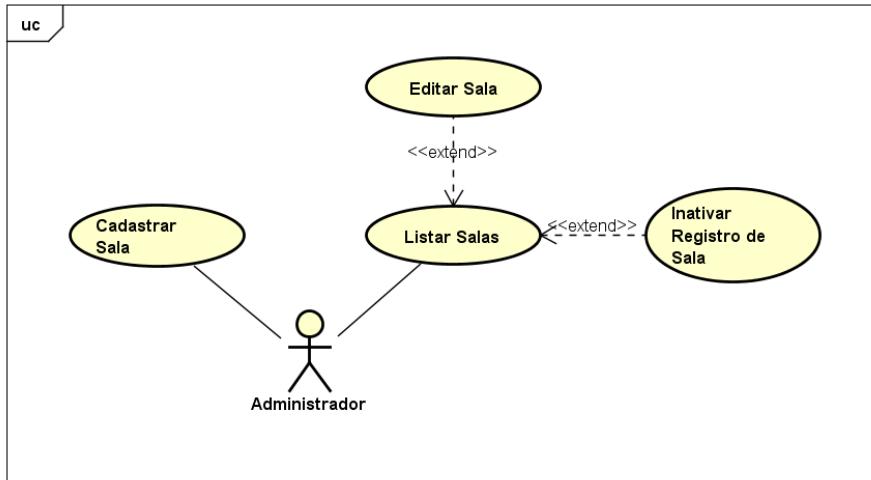
Diante do processo de estruturação da solução proposta, foi possível identificar cinco módulos principais que contemplam as diferentes funcionalidades que serão providas. Além disso, foram identificados dois atores principais: o *administrador*, responsável pelo cadastro dos principais elementos do sistema; e o *usuário*, que terá acesso físico às dependências e poderá consultar seus dados a respeito da utilização das mesmas.

Os módulos identificados são descritos brevemente a seguir, juntamente com o detalhamento dos diagramas de caso de uso. Uma descrição completa das funcionalidades providas pode ser vista no Apêndice A.

1. **Módulo de Gerenciamento de Salas.** Neste módulo, concentram-se as funcionalidades relativas à manutenção de salas, cujos acessos deverão ser controlados pelo sistema. O

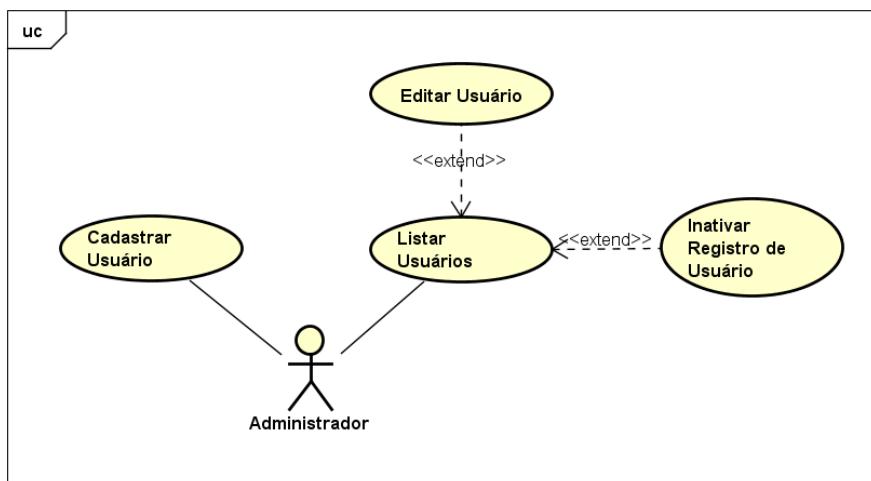
ator principal é o administrador, que definirá características fundamentais dessas salas (bloco, número e descrição), cadastrando-as e/ou modificando-as. O diagrama de caso de uso que enumera as funcionalidades desse módulo encontram-se ilustradas na Figura 3.3.

Figura 3.3: Caso de Uso: Módulo de Gerenciamento de Salas



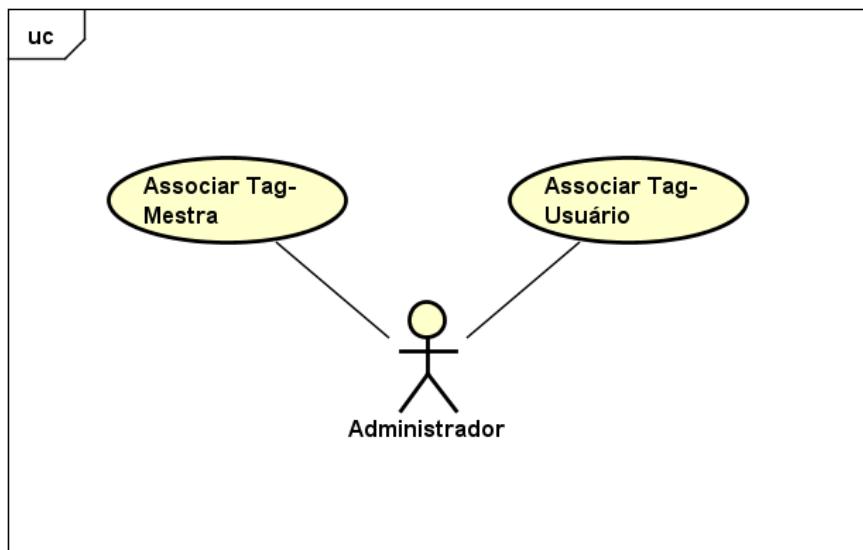
2. **Módulo de Gerenciamento de Usuários.** No módulo referente a usuários concentram-se as funcionalidades relativas à manutenção dos usuários que terão seus acessos monitorados e concedidos pelo sistema. O ator principal é o administrador, que definirá os dados desses usuários, cadastrando-os e/ou modificando-os. A Figura 3.4 ilustra o diagrama de caso de uso associado a esse módulo.

Figura 3.4: Caso de Uso: Módulo de Gerenciamento de Usuários



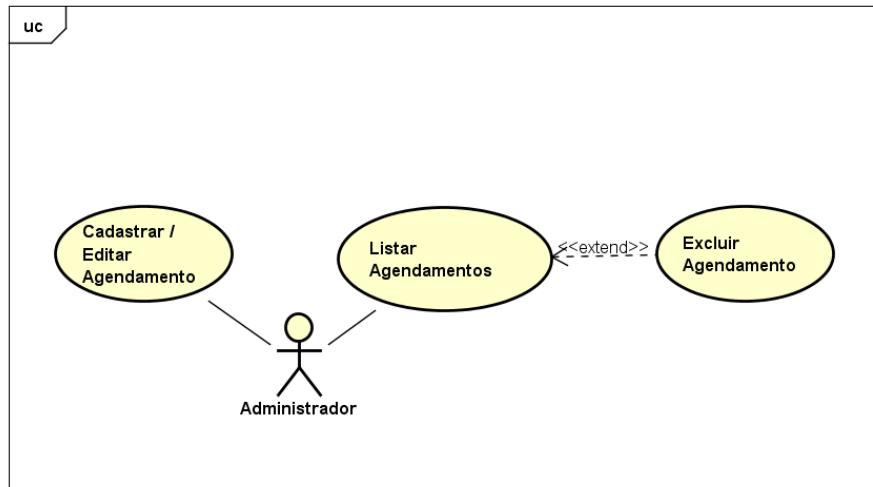
3. **Módulo de Gerenciamento de *Tags*.** As funcionalidades pertinentes a este módulo definem a maneira na qual o administrador, que é tido como ator principal, irá relacionar *tags* a usuários previamente cadastrados no sistema. A diferenciação quanto ao tipo de *tag*, mestra ou comum, deverá ser considerada, como mostra o diagrama de caso de uso da Figura 3.5. Considera-se a *tag* mestra aquela utilizada pelo administrador, que detém do acesso irrestrito à ambientes e a *tag* comum, aquela utilizada por usuários comuns e lhes dão acesso aos ambientes mediante agendamento prévio.

Figura 3.5: Caso de Uso: Módulo de Gerenciamento de *Tags*



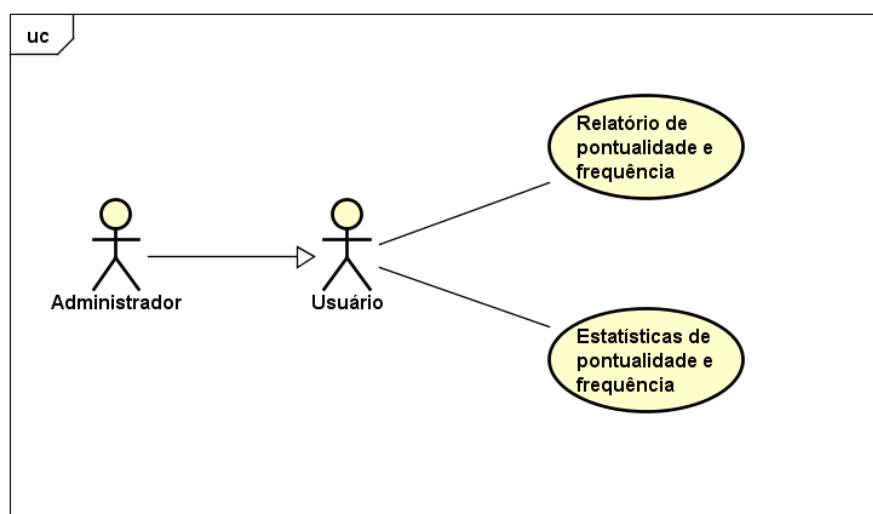
4. **Módulo de Gerenciamento de Agendamentos.** Este módulo é composto por funcionalidades de cadastro e mantimento de agendamentos de usuários de acordo com horários e salas, englobando, portanto, todos os elementos previamente cadastrados no sistema. O ator principal é o administrador, o qual realizará as atividades necessárias para êxito das funcionalidades, como mostra a Figura 3.6.

Figura 3.6: Caso de Uso: Módulo de Gerenciamento de Agendamentos



5. **Módulo de Gerenciamento de Relatórios.** Neste módulo, cujo diagrama de caso de uso encontra-se ilustrado na Figura 3.7, concentram-se as funcionalidades relativas aos relatórios de acesso às salas, os quais poderão ser consultados pelos usuários em geral e também pelo administrador. Este módulo visa prover informações gerais sobre o acesso de um determinado usuário às salas que lhes foram agendadas, indicando, por exemplo, a taxa de pontualidade.

Figura 3.7: Caso de Uso: Módulo de Gerenciamento de Relatórios



### 3.3 Projeto do Hardware do Objeto Inteligente

A descrição do *hardware* do objeto inteligente consiste na definição e concepção do arranjo dos componentes que proporcionam o controle de acionamento de fechadura, da leitura de *tags* RFID e do estabelecimento de uma interface de saída de dados permitindo a inspeção visual do acesso concedido. Considerando estes elementos, os componentes utilizados no projeto de *hardware* do sistema de controle de acesso proposto são descritos a seguir, juntamente com suas funções específicas na composição da solução proposta:

- **NodeMCU - Modelo v3 Lolin.** Trata-se de um microcontrolador que disponibiliza portas lógicas para a conexão física entre ele e outros componentes e, por meio de um *software* embarcado, realiza a conexão lógica dos mesmos, o que possibilita a entrada, saída e manipulação de dados trocados entre estes elementos. No contexto considerado, será o elemento central de processamento do objeto inteligente;
- **Módulo Leitor RFID - Modelo Mfrc522 Mifare.** É responsável pela entrada de dados no sistema físico do projeto, pois, a partir da leitura de *tags* RFID, inicia-se o processo de validação e permissão de usuários aptos a acessarem o ambiente controlado;
- **Visor LCD 16x2 - Modelo Backlight Azul.** Este componente permite ao usuário, que deseja acessar um ambiente controlado, a inspeção visual de informações retornadas pelo sistema após processamento de sua *tag*, ou seja, a aprovação ou rejeição do seu acesso mediante prévio agendamento;
- **Módulo Serial I2C para Display LCD - Modelo FC-113.** Esse módulo é responsável pela supressão de pinos na utilização do visor LCD, uma característica importante diante da quantidade limitada de pinos no microcontrolador *NodeMCU*. Com a utilização deste módulo torna-se possível controlar o visor LCD com apenas 2 pinos lógicos;
- **Módulo Relé com 2 canais - Modelo SRD-05VDC-SL-C.** Responsável pelo acionamento de abertura de porta do ambiente controlado e da alimentação do dispositivo

interno. Nesse protótipo é utilizado um módulo de relés com 2 canais os quais são acionados conforme sinal digital enviado pelo microcontrolador;

- **LEDs.** No âmbito deste projeto, são utilizados dois LEDs para abstração de dispositivos controláveis. A utilização do LED de cor verde abstrai a fechadura física a ser acionada mediante a liberação da corrente, uma vez que existem diferentes modelos de fechaduras com características particulares de voltagem, tipo de conector, corrente, por exemplo , cuja utilização poderia transmitir uma noção incorreta de que apenas aquele tipo de fechadura poderia vir ser utilizado nesta solução. O LED de cor vermelha, de maneira análoga, abstrai o dispositivo interno à sala que será ativado posteriormente à liberação. Como referência, quando os LEDs estão acesos, tem-se a representação da sala aberta e do dispositivo interno ligado, enquanto que, quando apagados, tem-se a representação da sala fechada e dispositivo interno desligado.

A escolha do microcontrolador *NodeMCU* foi motivada por sua capacidade de suprir as necessidades de processamento e realização de tarefas, propostas neste projeto de *hardware*, e sobretudo seu baixo custo quando comparados com a plataforma Arduino ou *Raspberry Pi*, por exemplo. De maneira geral, o microcontrolador Arduino possui menor custo que a plataforma utilizada, entretanto, para suprir as necessidades de comunicação à rede *Wi-Fi* seria necessária a aquisição de um módulo *Wi-Fi* à parte (a citar, o módulo ESP-01) cuja conexão faria uso de portas lógicas do Arduino, o que reduziria a quantidade de pinos para a interação com os demais componentes deste projeto. O *Raspberry Pi* por sua vez, disponibilizaria uma quantidade maior de memória e processamento, que diante da necessidade do sistema seria desperdiçada, além de ter custo mais alto quando comparado ao *NodeMCU*.

Após a definição dos elementos de composição do projeto de *hardware* do objeto inteligente, foi construído um diagrama esquemático, com o auxílio do *software Fritzing* (FRITZING, 2018), conforme ilustrado na Figura 3.8, no qual é possível identificar os componentes mencionados, as conexões definidas e a devida interação entre eles. Na Figura 3.9 encontra-se o esquema elétrico detalhado deste projeto, também construído com o auxílio do *software Fritzing* (FRITZING, 2018).

Figura 3.8: Diagrama esquemático do projeto de *hardware* do protótipo do objeto inteligente. Os símbolos + e – denotam os polos do circuito elétrico que alimentam a fechadura e o dispositivo interno.

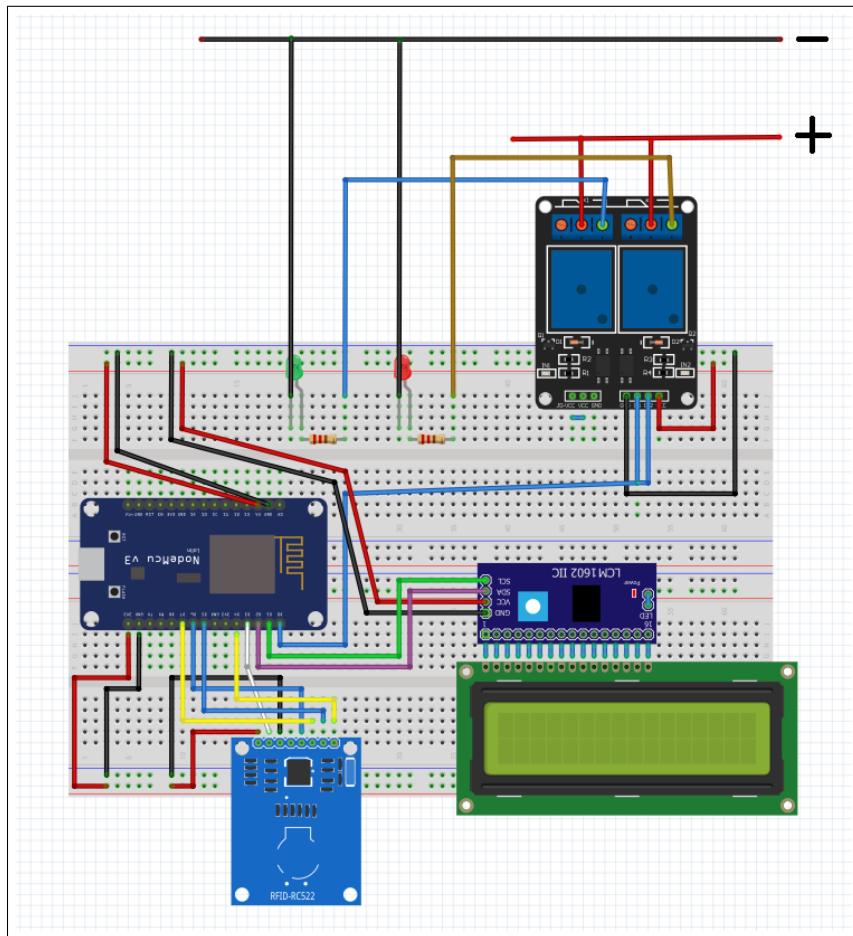
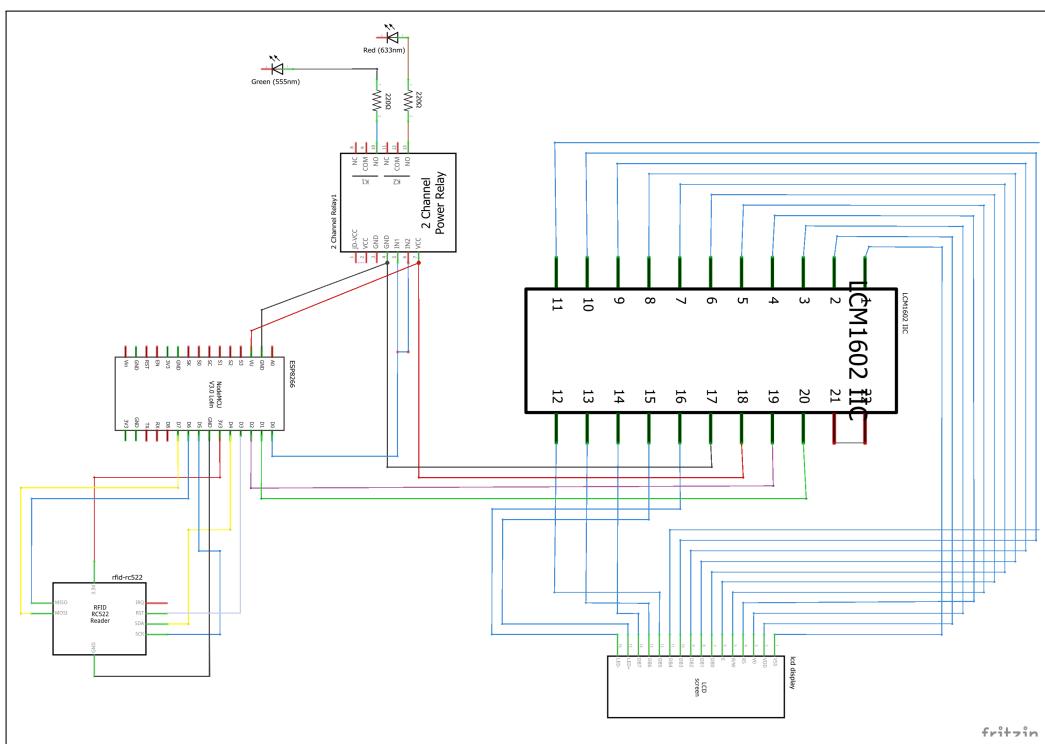


Figura 3.9: Esquema elétrico do projeto de *hardware*.



## 3.4 Projeto de Software

O projeto de *software* da solução proposta neste trabalho, é constituído por três partes distintas, a citar: *software* embarcado no objeto inteligente, aplicação web e orquestrador de saída de dados do objeto. A seguir, cada uma dessas três partes é detalhada para um melhor entendimento.

### Software Embarcado no Objeto Inteligente

O *software* embarcado no objeto inteligente compreende a porção de código desenvolvida para realizar a conexão lógica dos elementos físicos do *hardware* do objeto, assim como a publicação e inscrição deste protótipo em tópicos de comunicação com o MQTT *Broker*.

Este *software* foi desenvolvido no Ambiente de Desenvolvimento Integrado do Arduino (IDE Arduino) dada sua compatibilidade com a plataforma *NodeMCU* e os demais elementos físicos presentes na solução. A IDE Arduino permite a utilização de bibliotecas que tratam as características específicas dos elementos utilizados no protótipo do objeto, tornando a interação entre eles mais fácil e transparente ao programador. A seguir são listadas as bibliotecas utilizadas e os componentes que elas controlam:

- **MFRC522.** Responsável por controlar o módulo RFID, proporcionando funções que auxiliam a leitura e manipulação de informações das *tags* RFID;
- **SPI.** É a biblioteca que define um protocolo de dados seriais síncronos usado por microcontroladores para se comunicar com um ou mais dispositivos periféricos rapidamente em curtas distâncias, nesse caso, dá suporte para interação entre o leitor RFID e o *NodeMCU*;
- **LiquidCrystal\_I2C.** Facilita o controle do visor LCD em união com seu módulo conversor I2C, por meio de funções de saída de informação e controle de posição na matriz do LCD;
- **Wire.** Esta biblioteca permite a comunicação com dispositivos I2C / TWI, dando suporte à conexão do microcontrolador ao *display* LCD;

- **ESP8266WiFi.** Biblioteca responsável por realizar a conexão do módulo *Wi-fi* da placa *NodeMCU* a redes disponíveis, por meio de funções de conexão;
- **PubSubClient.** Trata-se da biblioteca que disponibiliza funções do protocolo MQTT para que o microcontrolador possa conectar-se ao MQTT *Broker*, inscrevendo-se ou publicando em tópicos definidos para comunicação;
- **ArduinoJson.** Essa biblioteca disponibiliza funções para padronização de mensagens enviadas no formato Json.

Além de funções de comunicação entre componentes e com a rede sem fio, o *software* embarcado realiza também processamentos simples de dados, como a construção de subtópicos para consultas de informações ao banco de dados do sistema, assim como definição de atividades decorrentes às respostas de tópicos aos quais está inscrito. De maneira geral, o *software* embarcado realiza a interpretação de informações e as converte em impulsos digitais para o controle físico do ambiente a ele associado.

## Aplicação Web

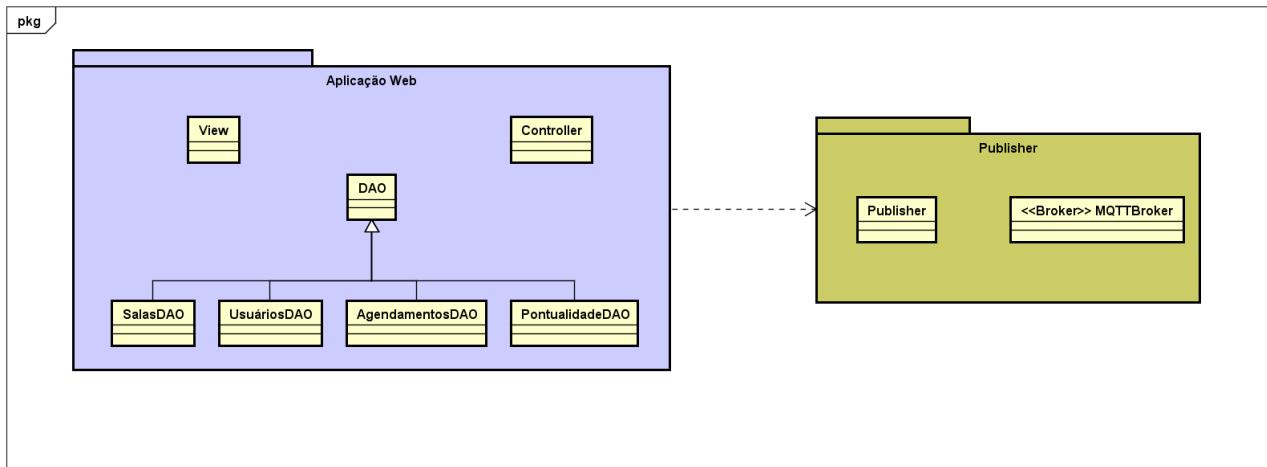
A aplicação web do sistema compreende o projeto de *software* responsável por proporcionar uma interação de alto-nível com os objetos inteligentes que compõem a solução de controle de acesso. Para tanto, caracteriza-se por um painel administrativo que, para o usuário administrador, permitirá a realização de procedimentos como cadastro de novos objetos inteligentes, inserção de novos usuários e também para inclusão de agendamentos que darão validade ao acesso. No caso de usuários comuns, este painel administrativo permite consultas aos dados relativos à pontualidade. Para implementação deste componente, propõe-se então o *framework* Web2py, fazendo uso do padrão arquitetural MVC, responsável por separar a representação dos dados de sua apresentação aos usuários.

O *framework* Web2py proporciona uma interface administrativa para a fácil alteração do código da aplicação web, cuja estrutura principal é gerada de maneira automática para cada nova aplicação criada e nela podem ser identificados os componentes relativos aos modelos,

controladores e visões (PIERRO, 2010). Na solução proposta, essa estrutura tende a se manter, em virtude do padrão arquitetural adotado, mas novos elementos e formatações são incluídas à medida que as funcionalidades são implementadas.

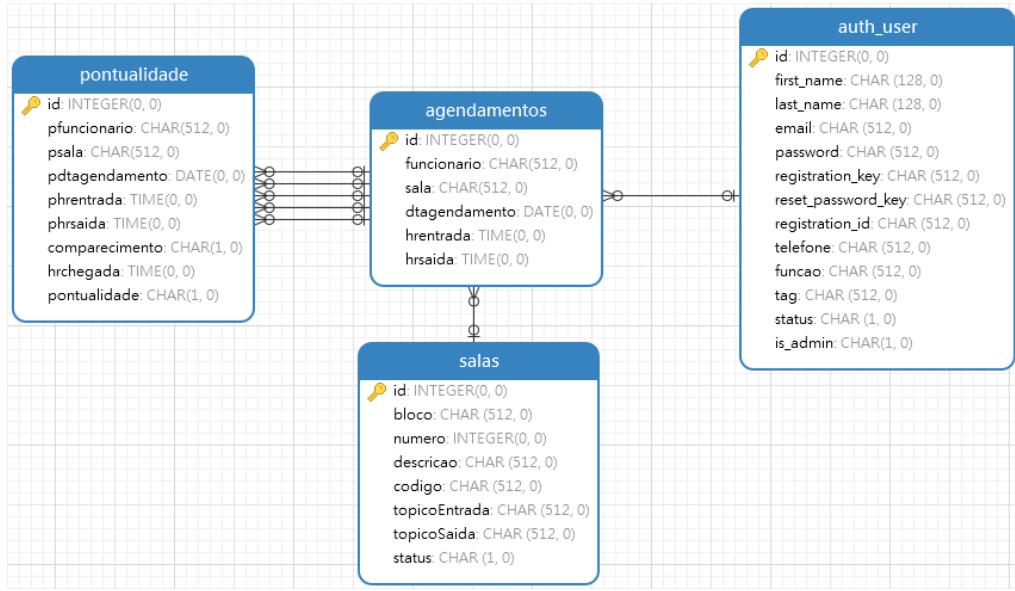
O Diagrama de pacotes apresentado na Figura 3.10 ilustra a organização geral da aplicação web proposta, composta pelos pacotes Aplicação Web e Módulo Publisher. O pacote Aplicação Web inclui a estrutura básica definida pelo *framework* com *Views*, *Controllers* e *Models*, estes últimos que definem as tabelas do banco de dados a serem manipuladas pela aplicação a partir de objetos de acesso a dados (DAO, do inglês *Data Access Object*). O Módulo Publisher, por sua vez, consiste nas classes Publisher e MQTTBroker, que fazem a intermediação entre aplicação web e objeto inteligente das salas, para as funcionalidades de associação de *tags*.

Figura 3.10: Diagrama de pacotes da aplicação web do sistema de controle de acesso.



Tendo em vista o domínio da aplicação web, verificou-se que há a necessidade de desenvolver quatro classes de acesso à camada de dados, sendo estas relativas aos usuários, salas, agendamentos e registros de pontualidade, cujos atributos e relações encontram-se detalhados no diagrama de entidade-relacionamento da Figura 3.11.

Figura 3.11: Diagrama de entidade-relacionamento da aplicação web do sistema de controle de acesso.



Acima são representadas as relações entre entidades do banco de dados considerando as regras de negócio estabelecidas pelo sistema. Os processos de cadastro de salas e usuários são independentes mas são necessários para o cadastro de agendamentos e geração de registros de pontualidade. No cadastro de cada novo agendamento, identifica-se como chave estrangeira o campo funcionário, a partir do campo email da tabela auth\_user que representa o cadastro de usuários, e o campo sala a partir do campo código da tabela salas. Na criação de registros de pontualidade, consideram-se chaves estrangeiras os campos pfuncionario, psala, pdtagendamento, phreentrada e phrsaida, todos associados aos campos de nome semelhante da tabela agendamentos.

Considerando a arquitetura MVC da aplicação web, os casos de uso foram então detalhados em diagramas de sequência, visando esclarecer a relação entre os componentes e as trocas de mensagens entre eles para a efetiva implementação deste módulo. Nas Figuras 3.12 e 3.13, em especial, são apresentados os diagramas de sequência que melhor definem a interação entre a aplicação web e o módulo Publisher, enfatizando a dependência do primeiro em relação ao segundo, visto que a comunicação da aplicação web para acesso remoto a uma sala só ocorre se a mensagem enviada pela mesma ao MQTTBroker alcançar seu fim, o objeto inteligente da sala desejada.

Figura 3.12: Diagrama de sequência para a Associação de Tag-Usuário.

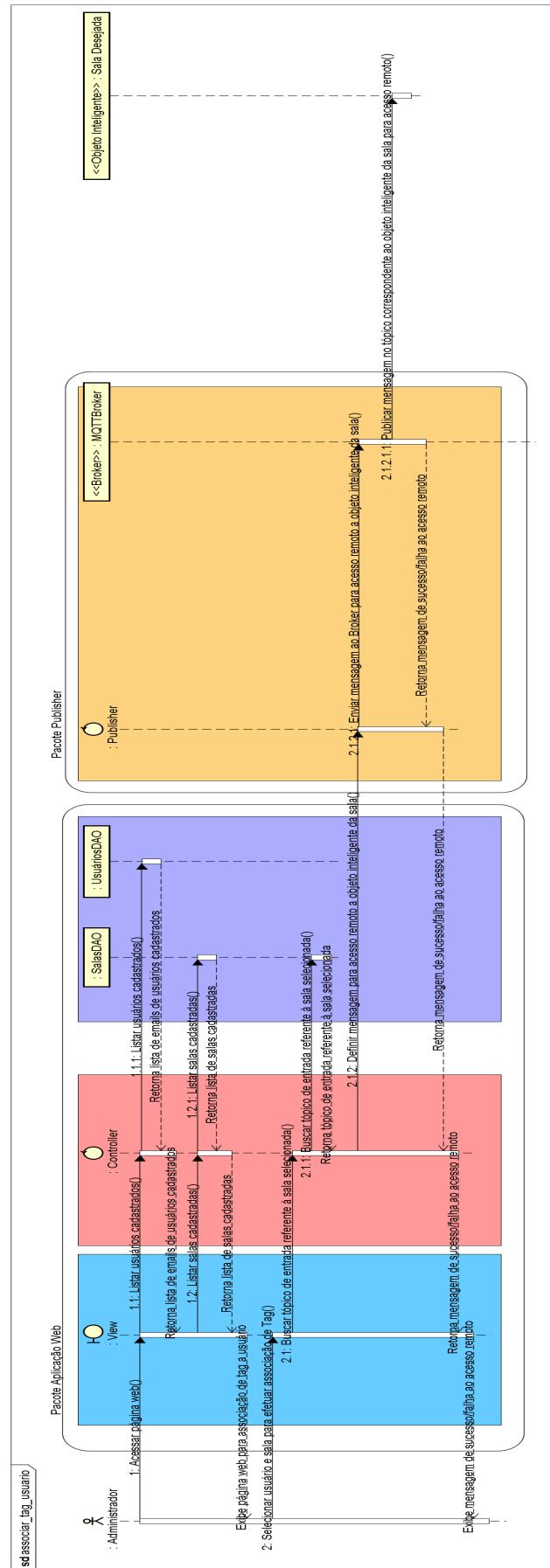
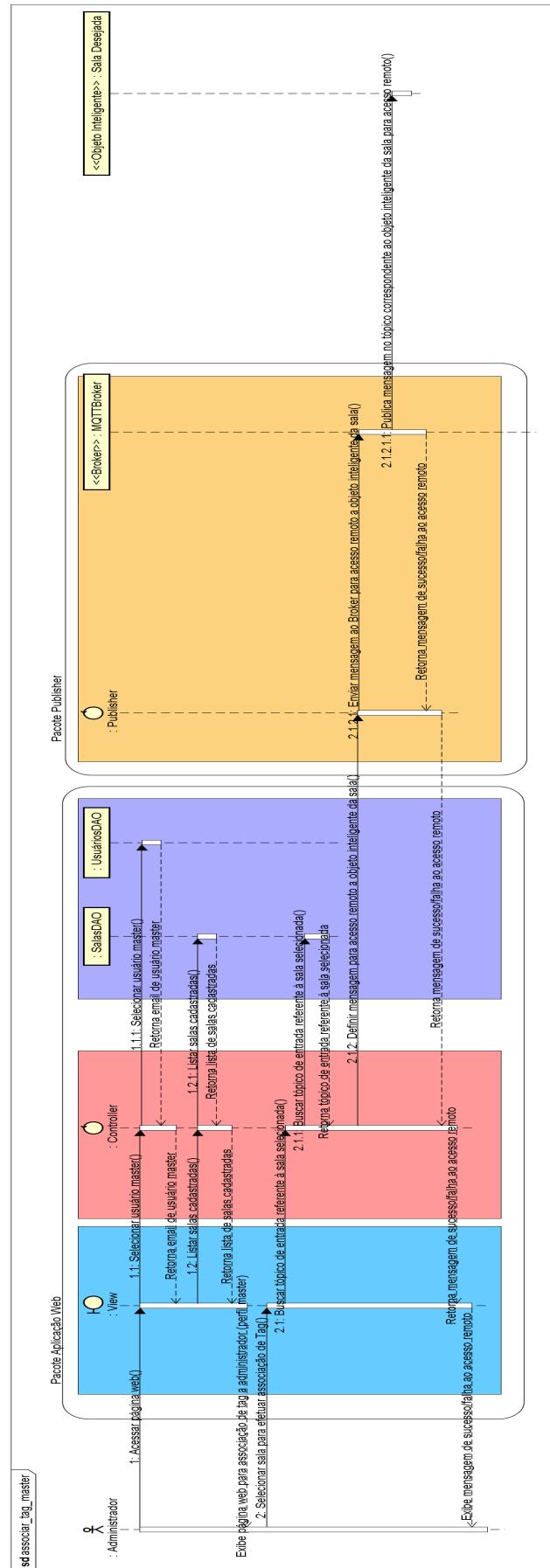


Figura 3.13: Diagrama de sequência para a Associação de *Tag-Master*.

Os demais diagramas de sequência, relacionados aos casos de uso remanescentes, são ilustrados no Apêndice B, sendo detalhadas todas as trocas de mensagens entre os componentes da arquitetura MVC para efetivação das funcionalidades do sistema.

De maneira geral, o uso do *framework* Web2py mostra-se eficiente para a proposta de implementação definida, sobretudo no tocante às páginas web, com as quais os usuários interagem com a aplicação, pois a aparência das funções são pré-definidas automaticamente, sem a necessidade de criação dos arquivos HTML ou CSS. Apesar da comodidade, não se exclui a possibilidade de modificar e personalizar tal implementação incluindo conteúdos no formato HTML, RSS, XML, JSON, dentre outros, a fim de melhorar a usabilidade. Em particular, considera-se a utilização do *framework* Bootstrap para promover estas melhorias, gerando páginas web mais intuitivas e visualmente agradáveis.

## Orquestrador de Saída de Dados do Objeto

O Orquestrador de saída de dados do objeto é um *software*, desenvolvido na linguagem Python, que otimiza o processamento de dados solicitados pelo objeto e realiza atividades específicas de acordo com esta solicitação. Suas principais atribuições são:

- Realizar a conexão lógica com o MQTT Broker, utilizando a biblioteca `paho.mqtt.client`, o que possibilita a inscrição e publicação a tópicos relacionados aos objetos;
- Identificar os padrões de tópicos utilizados para comunicação com os objetos por meio de consultas ao banco de dados;
- Estabelecer funções para validação de agendamento e cadastro de novas *tags* de acordo com o canal de comunicação (tópico) de recebimento;
- Validar informações de agendamento a partir de uma comparação entre a *tag* recebida e a informação persistida no banco de dados;
- Realizar a associação de uma nova *tag* a um usuário, a partir da persistência desse dado ao banco de dados.

Esse módulo, utiliza o banco de dados inicialmente criado pela aplicação web e o acessa com auxilio da biblioteca `sqlite3`, que proporciona funções para fácil manipulação de dados em bancos SQL, mas preservando a sintaxe da linguagem Python. Além das duas bibliotecas mencionadas anteriormente, o orquestrador faz uso também das seguintes bibliotecas: `json`, para padronizar a resposta retornada, no formato JSON, e `time`, que estabelece intervalos de espera entre funções de conexão.

Para melhor estruturar a codificação do orquestrador, foi realizada a subdivisão das funcionalidades em três partes distintas, separadas em arquivos, a citar: `subscriber_pc.py`, `DAL.py` e `publisher_pc.py`. O arquivo `subscriber_pc.py` é responsável por ficar sempre ativo, na escuta de tópicos de saída dos protótipos, esperando por mensagens a serem validadas. Dependendo do tópico que está recebendo a mensagem, o programa executará uma rotina diferente, podendo ser um tópico de validação de agendamento ou de associação de *tag* a usuários.

Ambas as rotinas redirecionam o fluxo de execução do programa para o arquivo `DAL.py`, responsável por validar ou persistir as informações recebidas pelo objeto às informações contidas nas tabelas do banco de dados do sistema. É nesse programa onde definem-se as funções SQL para busca e inserção de dados no sistema. E por fim, o `publisher_pc.py` é definido, com a função de publicar as informações de aceite ou rejeição que retornam como resposta das sentenças SQL. Esse componente, assim como o `subscriber_pc.py`, possui um bloco de configuração dos parâmetros de conexão ao MQTT Broker, que irá variar de acordo com o cenário, refletindo, por exemplo, detalhes da conexão *Wi-Fi* como nome de rede e senha de acesso.

# Capítulo 4

## Resultados e Discussão

Esta seção descreve os resultados obtidos do desenvolvimento da solução proposta, levando em consideração o cumprimento das atividades previstas na metodologia. A apresentação desses resultados respeita a estrutura e a subdivisão em módulos previamente apresentada no Capítulo 3 e está organizada da seguinte forma: inicialmente, para favorecer um melhor entendimento, há uma apresentação do padrão de tópicos de comunicação na Seção 4.1; em seguida, na Seção 4.2, é apresentado o resultado da implementação do projeto de *hardware* e a implementação do projeto de *software* do objeto inteligente é descrita na Seção 4.3. Os resultados obtidos pelo desenvolvimento da aplicação web do sistema são apresentados na Seção 4.4; na Seção 4.5 são descritos os resultados obtidos pela implementação do Orquestrador e por fim, na Seção 4.6 é realizada uma análise comparativa entre o projeto desenvolvido neste trabalho e os projetos apresentados pelos trabalhos relacionados que são descritos no Capítulo 2.

### 4.1 Padronização dos Tópicos de Comunicação

Para comunicarem-se entre si, o objeto inteligente, a aplicação web e o orquestrador fazem uso do protocolo de comunicação MQTT baseado no padrão *publisher/subscriber*. A mensagem enviada por um cliente *publisher* é composta pela mensagem que ele deseja transmitir e um tópico de publicação, que determina o canal de comunicação a ser usado. O cliente *subscriber*, por sua vez, inscreve-se nos tópicos específicos de seu interesse.

Os tópicos de publicação e de inscrição são definidos por um título, podendo ser criados subtópicos relacionados utilizando barras e inserindo subtítulos ao nome principal. Neste projeto foram especificados dois principais tópicos para a interação entre módulos: (i) o tópico de entrada e (ii) o tópico de saída.

O tópico de entrada, definido pelo título `topicoEntrada`, representa o canal de transmissão de informação cujo fluxo é no sentido orquestrador-objeto ou aplicação web-objeto. O padrão de construção desse tópico dá-se pela regra `topicoEntrada/codigo_da_sala/funcionalidade_executada` e permite a inclusão de novos objetos inteligentes e novas funcionalidades a partir desta definição. As funcionalidades definidas neste projeto para a relação aplicação web-objeto, são `topicoEntrada/codigo_da_sala/associarTagUser`, `topicoEntrada/codigo_da_sala/associarTagMaster`. Para o fluxo orquestrador-banco-objeto tem-se a função definida por `topicoEntrada/codigo_da_sala/confirmarAgendamento`.

De maneira análoga, define-se o tópico de saída pelo título `topicoSaida`, representando o canal de transmissão de informação com fluxo contrário, cujo sentido é objeto-orquestrador. O padrão de construção de tópicos é semelhante, dado por: `topicoSaida/codigo_da_sala/funcionalidade_executada`. O objeto relaciona-se com o orquestrador para a persistência ou consulta de informações no banco por meio das funções definidas seguintes nos tópicos: `topicoSaida/codigo_da_sala/salvarTagUser`, `topicoSaida/codigo_da_sala/salvarTagMaster`, `topicoSaida/codigo_da_sala/validarAgendamento`.

Enfatiza-se a importância deste padrão na criação de tópicos para que os módulos do sistema de controle de acesso consigam interagir entre si de maneira fluida, automatizada e em conformidade com as informações persistidas no banco no momento do cadastro de um novo ambiente a ser monitorado. Além disso, possibilita um número grande de objetos simultaneamente e evita conflitos. Este padrão será citado de maneira recorrente nas próximas seções quando forem mencionados aspectos da implementação de códigos de comunicação.

## 4.2 Implementação do Projeto de Hardware

Para a efetiva implementação do projeto de *hardware* da solução proposta, foram utilizados componentes de baixo custo que suprissem as necessidades elencadas inicialmente. Para tanto, fez-se a aquisição de alguns componentes, cujo custo, com vistas a ilustrar a viabilidade financeira da solução proposta, são apresentados na Tabela 4.1.

Tabela 4.1: Custo total dos componentes para implementação de um protótipo do objeto inteligente. Estes preços foram praticados em Manaus no mês de Setembro de 2018.

Quantidade	Descrição do Componente	Custo Unitário (R\$)
1	Microcontrolador NodeMCU	50,00
1	Display LCD 16 × 2	24,00
1	Módulo serial I2C para LCD 16 × 2	15,00
1	Módulo RFID MFRC-522	28,00
1	Módulo relé 2 canais	22,00
1	Protoboard 400 furos	18,00
1	Protoboard 830 furos	25,00
1	Kit Jumpers 60 unidades	21,00
2	LEDs Ultra Brilho 3mm	0,50
2	Resistores 220 Ω	0,60
1	Adaptador Tomada Plug Fonte USB	10,00
<b>Custo Total</b>		<b>215,20</b>

Soluções comerciais que proveem fechaduras eletrônicas custam, em média, de R\$ 400,00 a R\$ 1.200,00 em valores consultados em Abril de 2019. Algumas destas soluções podem ser administradas remotamente, mas a ampla maioria não oferece suporte à agendamentos, registro de horários de acesso e nem tampouco interligação para acionamento de outros dispositivos. Embora os custos da solução proposta não contemplem a fechadura em si, modelos compatíveis com a solução em questão, tais como fechaduras elétricas, possuem custo variando entre R\$ 75,00 e R\$ 150,00 em valores correntes, que resultariam em um custo máximo total de R\$ 364,10, ainda abaixo dos menores valores de soluções prontas encontrados.

É importante ressaltar que os componentes apresentados na Tabela 4.1 precisam ser contemplados de acordo com as especificações de pinagem definidas anteriormente, conforme diagrama da Figura 3.8. Este detalhamento é apresentado a seguir.

Para que o usuário possa realizar a inspeção visual dos dados de acesso processados pelo

NodeMCU e exibidos no objeto, utilizou-se um *display LCD* com auxílio do módulo I2C, permitindo o uso de menos pinos do microcontrolador. A conexão entre o microcontrolador e o Módulo serial I2C, apresentada na Tabela 4.2, envolve apenas 4 pinos, em que GND e VCC que controlam a alimentação do módulo, e SDA e SCL que representam a linha de dados e linha de *clock*, respectivamente. Enfatiza-se a utilidade deste módulo serial no protótipo do objeto proposto por reduzir a quantidade de pinos necessários para o funcionamento do *display LCD*.

Tabela 4.2: Especificação da conexão de pinos entre NodeMCU e módulo serial I2C.

<b>NodeMCU</b>	<b>Módulo Serial I2C</b>
GND	GND
VU	VCC (5V)
D2	SDA
D1	SCL

Para leitura de *tags*, utilizou-se o módulo RFID, que possui sete pinos de conexão ao microcontrolador, como identificado na Tabela 4.3, e determina uma comunicação serial do tipo SPI (*Serial Peripheral Interface*). Os pinos SDA e SCK representam, assim como no módulo serial I2C, uma linha de dados e uma linha de *clock* para a comunicação síncrona. Os pinos MISO (*Master In Slave Out*) e MOSI (*Master Out Slave In*) definem a troca de dados em duas direções. Os pinos GND e VCC alimentam o circuito do módulo, cuja tensão necessária é de 3.3V, e o pino RST é utilizado para resetar o módulo a partir de um sinal digital.

Tabela 4.3: Especificação da conexão de pinos entre NodeMCU e módulo RFID.

<b>NodeMCU</b>	<b>Módulo RFID</b>
D4	SDA
D5	SCK
D7	MOSI
D6	MISO
- (não conectado)	IRQ
GND	GND
D3	RST
3V	VCC (3.3V)

O último módulo é responsável pelo controle e acionamento da fechadura do ambiente controlado a partir da conexão entre o microcontrolador e o módulo de relés, com pinagem

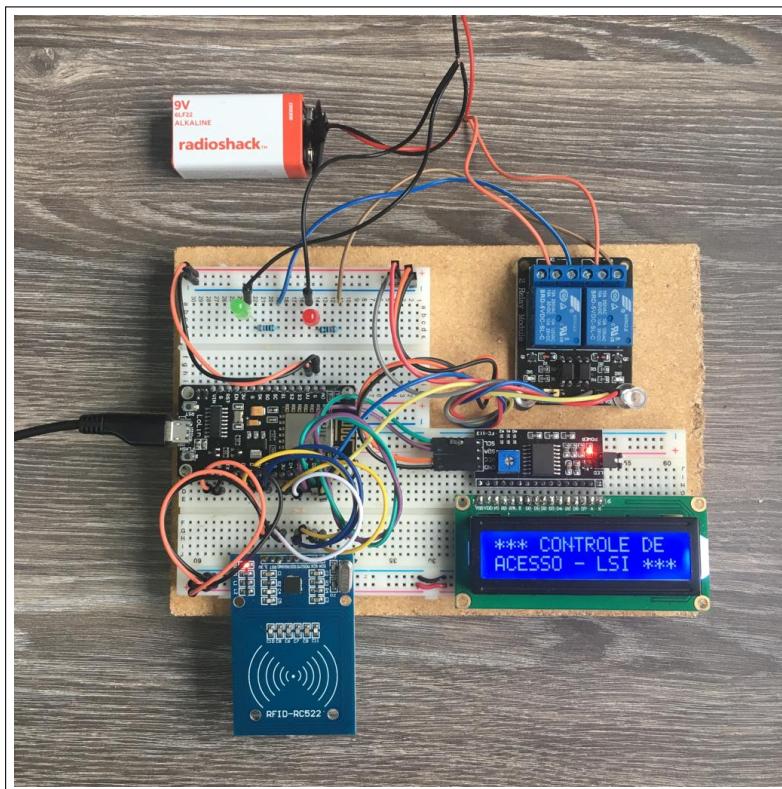
detalhada na Tabela 4.4. Considerando a disponibilidade das peças no comércio local, utilizou-se um módulo de 2 canais que possui como pinos GND e VCC para alimentação, cuja tensão necessária é de 5V, além dos pinos de entrada de sinal digital IN1 e IN2 que acionam o relé a partir da presença ou não de sinal.

Tabela 4.4: Especificação da conexão de pinos entre NodeMCU e módulo relé.

NodeMCU	Módulo Relé
GND	GND
D0	IN1
D0	IN2
VU	VCC (5V)

Como produto da aquisição destes componentes e da realização das conexões conforme especificado, foi elaborado um protótipo do *hardware* do objeto inteligente, ilustrado na Figura 4.1. A bateria disponível no protótipo faz alusão à alimentação elétrica externa para a fechadura e para o dispositivo interno. Este protótipo é uma prova de conceito do projeto de *hardware* apresentado na solução proposta.

Figura 4.1: Protótipo do *hardware* do objeto inteligente.



## 4.3 Implementação do Software Embarcado

O *software* implementado para controle de componentes e processamento de dados, foi desenvolvido na IDE do Arduino, em virtude desta ferramenta permitir a utilização de bibliotecas específicas de cada componente e facilitar o *upload* do código na plataforma NodeMCU.

Por meio da criação de funções, foram estabelecidos procedimentos para que a comunicação entre o protótipo, orquestrador e a aplicação web fosse efetivada, e como forma de identificar as etapas de comunicação entre esses elementos, optou-se por definir fluxogramas para a descrição de processos e subprocessos dessa interação, representada na Figura 4.2.

O algoritmo estabelecido pelo *software* embarcado na plataforma NodeMCU comprehende funcionalidades básicas, tais como a inclusão de bibliotecas para instanciação de objetos, especificações de pinos de controle para manipulação de componentes físicos, definições de tópicos para comunicação do *hardware*, padronizada como descrito na Seção 4.1, e um *setup* de comunicação. Este *setup* é responsável por inicializar os componentes físicos do protótipo, como ponto de partida do processo, incluindo a conexão do módulo ao servidor Broker MQTT para que suas mensagens possam ser enviadas e recebidas para funcionamento do sistema.

Uma vez conectado à rede *Wi-Fi* disponível, inicia-se então a função de escuta por tópicos de interação, identificados por Tópico 1, 2 e 3 que podem disparar os respectivos subprocessos 1, 2 e 3. O Tópico 1 corresponde à chamada realizada pela aplicação web ao protótipo para que seja aberta comunicação para leitura de *tag* e efetuada a atualização da informação do cadastro de administrador (Master), como indica a Figura 4.3.

De forma similar ocorre com o Tópico 2 e a ativação do Subprocesso 2, correspondente à chamada realizada pela aplicação web para leitura de *tag* e efetuada a atualização da informação do cadastro do usuário (User), como indica a Figura 4.4.

Por fim, o Subprocesso 3, ativado pelo Tópico 3, é responsável pela validação de agendamento a partir da leitura de *tag*, sendo possível realizar a abertura de Sala e inspeção visual do resultado por meio do *Display LCD*, como identificado na Figura 4.5.

Figura 4.2: Fluxograma que identifica etapas gerais do processo de comunicação entre módulos do projeto.

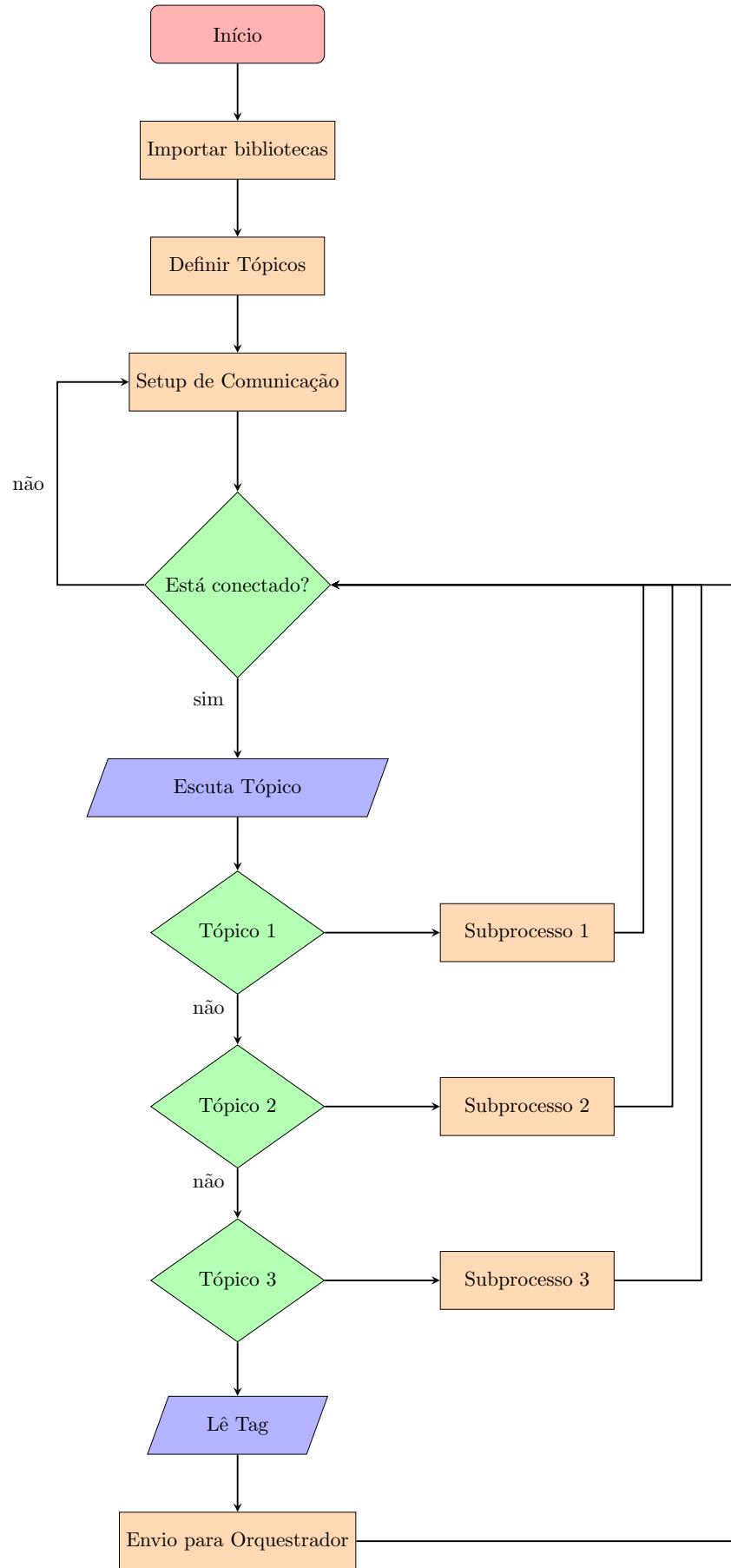


Figura 4.3: Fluxograma que identifica etapas de cadastramento de *Tag Master*.

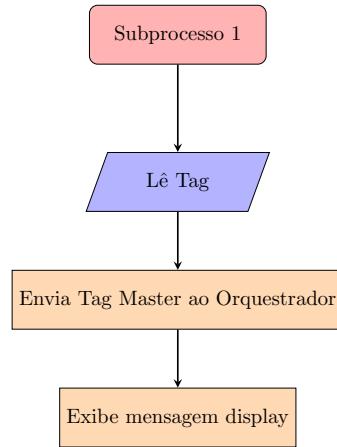


Figura 4.4: Fluxograma que identifica etapas de cadastramento de *Tag User*.

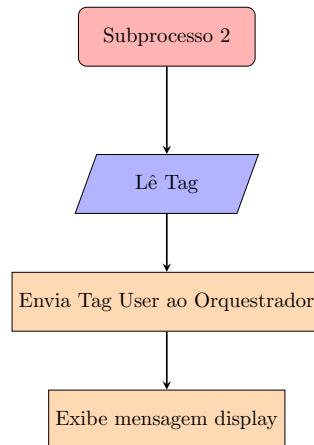
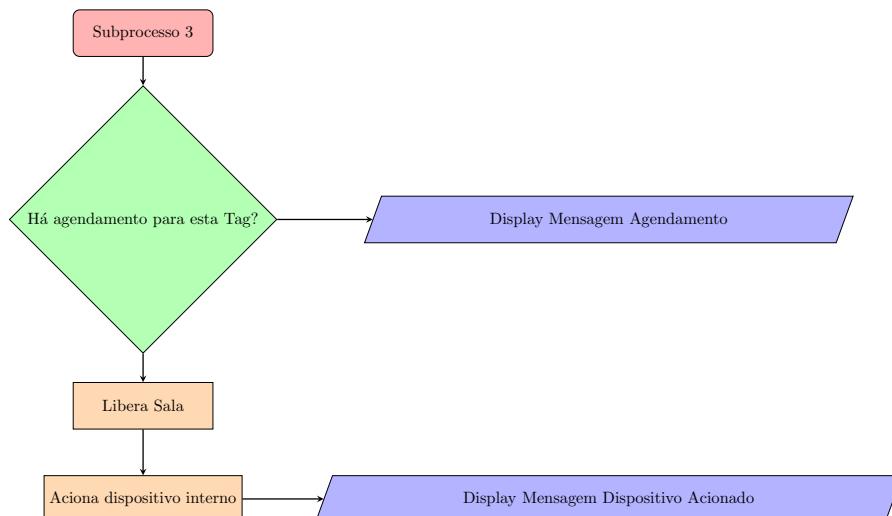


Figura 4.5: Fluxograma que identifica etapas de verificação de registros de Agendamento a partir da *Tag* de usuário.



De maneira geral, a estrutura do *software* embarcado define padrões para reconhecimentos de informações de acordo com as funcionalidades implementadas tanto na aplicação web quanto no orquestrador, permitindo então a execução das funcionalidades almejadas para o objeto proposto. O Apêndice D contém a versão completa do código-fonte elaborado para este fim.

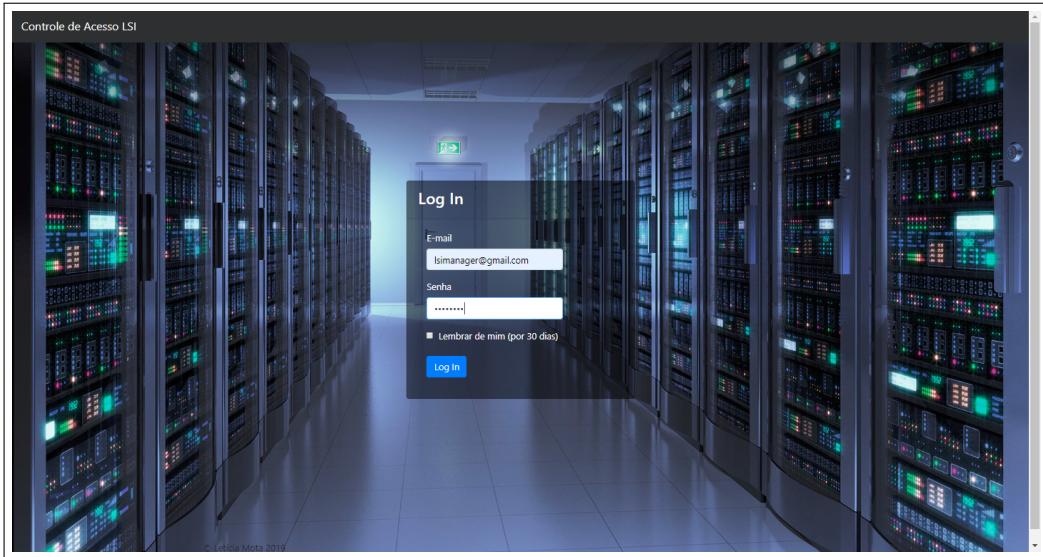
## 4.4 Implementação da Aplicação Web

A aplicação web deste projeto tem por função principal manter dados sobre usuários, salas, agendamentos e gerar relatórios de pontualidade e taxa de frequência, conforme descrito anteriormente na Seção 3.2. Para o administrador, esta aplicação atua como um painel administrativo das salas, usuários e agendamentos. Para os demais perfis de usuário, tal aplicação é útil para consulta dos seus dados de pontualidade.

Para implementar a aplicação web adotou-se o *framework* Web2Py, que permitiu o desenvolvimento dos requisitos elencados, a realização de testes preliminares de usabilidade de telas e de entrada e saída de dados. A curva de aprendizado para utilização deste *framework* foi considerada moderada. Algumas melhorias necessárias nos aspectos de usabilidade, entretanto, não puderam ser endereçadas diretamente com os recursos unicamente disponíveis neste *framework*, pois iriam demandar o desenvolvimento e personalização de muitos componentes para *web*, o que poderia comprometer o tempo disponível. Assim, para implementar as melhorias desejadas, utilizou-se o *framework* Bootstrap.

A tela correspondente à primeira interação do usuário com a aplicação web diz respeito à tela de login, ilustrada na Figura 4.6. O usuário deve possuir cadastro ativo no sistema e de acordo com seu perfil de acesso serão disponibilizadas funcionalidades pertinentes a ele.

Figura 4.6: Página de login da aplicação web.



A tela principal da aplicação, ilustrada na Figura 4.7, apresenta a ferramenta, seu propósito e algumas informações gerais. Na parte superior esquerda encontra-se a barra de menus que irá corresponder às funcionalidades do sistema atribuídas ao perfil do usuário. Se o usuário for o administrador, as opções de cadastros, relatórios e listagem de registros gerais do sistema são habilitadas. No caso do usuário comum, os registros são filtrados para o usuário logado e não é disponibilizado o menu de cadastros.

Figura 4.7: Página principal da aplicação web após autenticação do usuário.

Primeiramente, no tocante ao cadastro de usuários, tem-se a demanda pelo preenchimento

dos campos requeridos: nome, sobrenome, e-mail, senha, telefone e função, além dos campos *tag* e status que também são apresentados no formulário, mas preenchidos pelo sistema conforme as regras de negócio definidas. Para o preenchimento do campo *tag* foi criada uma tela de associação desta informação ao perfil do usuário previamente cadastrado, enquanto o campo status é marcado sempre que é realizado um novo cadastro, identificando que o cadastro está ativo.

Figura 4.8: Página de cadastro de usuários

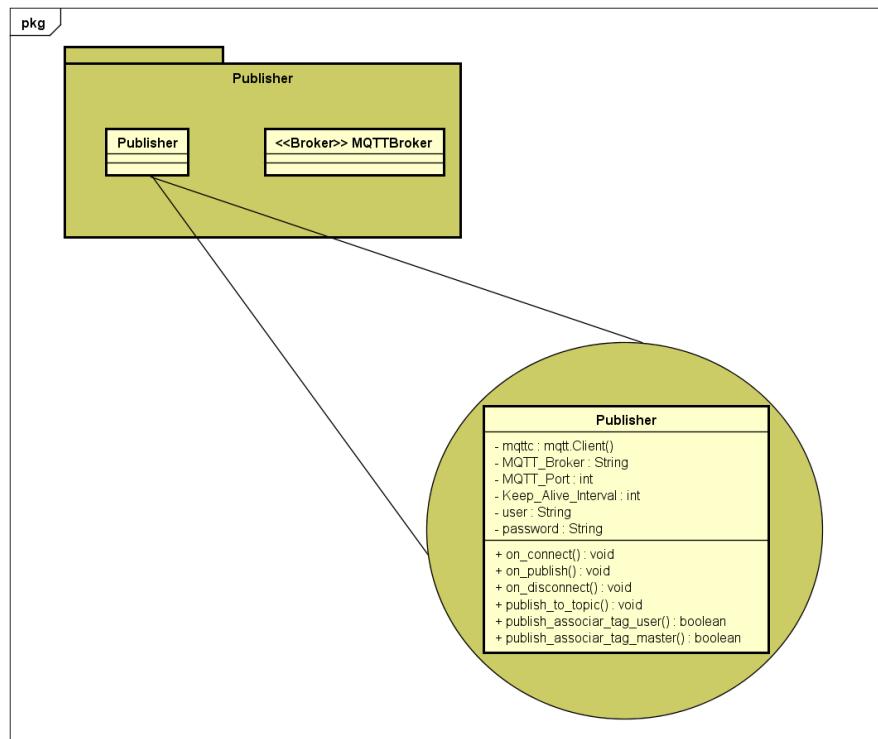
A associação de *tags* a usuário são complementares ao cadastro de usuários, seja ele comum ou administrador, pois inclui a informação da *tag* associada ao seu perfil. No desenvolvimento dessas funcionalidades foram definidas como informações básicas o e-mail do usuário cuja *tag* será associada e a sala cujo objeto inteligente funcionará como cadastradora. Para o caso de associação de *tag* ao perfil de administrador, no entanto, define-se apenas a sala, visto que a informação deste perfil é única e o sistema já o reconhece. As *views* implementadas são apresentadas nas Figuras 4.9 e 4.10.

Figura 4.9: Página de associação de *tag* a perfil master.

Figura 4.10: Página de associação de *tag* a perfil usuário.

Os formulários para associação de *tags* são desenvolvidos no *controller* da aplicação web e os filtros para seleção de usuários e salas cadastradas são definidos conforme sintaxe do *framework* Web2py, em que a linguagem Python é utilizada. Ao submeter as informações de usuário e sala selecionados, o sistema realiza uma chamada ao arquivo *Publisher* que atua como API para a conexão ao MQTT Broker e envio de solicitação ao objeto inteligente para a ativar sua função de cadastradora. Na Figura 4.11 ilustra-se o diagrama UML utilizado como base para posterior implementação deste módulo, em que são detalhados atributos e métodos. É importante notar que a chamada da aplicação web ao *Publisher* é feita de maneira interna, de forma transparente para o usuário administrador da aplicação.

Figura 4.11: Diagrama detalhado da classe Publisher.



No tocante aos usuários, a funcionalidade de listagem dos mesmos encontra-se ilustrada na Figura 4.12. Nesta listagem são apresentados todos os usuários cadastrados e suas respectivas informações, além do atalho para adição de novo registro e a opção para edição de informações do cadastro.

Figura 4.12: Listagem de usuários cadastrados.

Registro de Usuários						2 registros encontrados
Adicionar Registro						
Nome	Sobrenome	E-mail	Telefone	Função	Tag	
Elisa	Guedes	ebgcosta@uea.edu.br	12341234	Professor(a)	None	<a href="#">Editar</a>
Letícia	Mota	leticiaamota@gmail...	12344321	Professor(a)	86 85 16 7E	<a href="#">Editar</a>

© Letícia Mota 2019

Quanto ao desenvolvimento de *views* referentes às salas, houve a implementação de cadastro e listagem, conforme Figuras 4.13 e 4.14. No formulário de cadastro, são solicitados preenchimento dos campos bloco, número e descrição enquanto os campos código, tópico de entrada, tópico de saída e cadastro ativo são preenchidos pelo sistema segundo regras de negócio definidas. As montagens dos tópicos são definidas no *controller* da função e representam o tópico de comunicação do padrão MQTT utilizados para interação do objeto inteligente aos demais módulos do sistema projetado. Na tela de registro de sala são apresentadas as informações básicas além dos atalhos para adição de novos registros e para edição de cadastro já existente.

Figura 4.13: Página de cadastro de salas.

Cadastro de Salas

Bloco  
A

Número  
23

Descrição  
Sala de computação

Código  
A23

Tópico de Entrada  
topicoEntrada/A23

Tópico de Saída  
topicoSaída/A23

Cadastro Ativo

**Enviar**

© Letícia Mota 2019

Figura 4.14: Listagem de salas cadastradas.



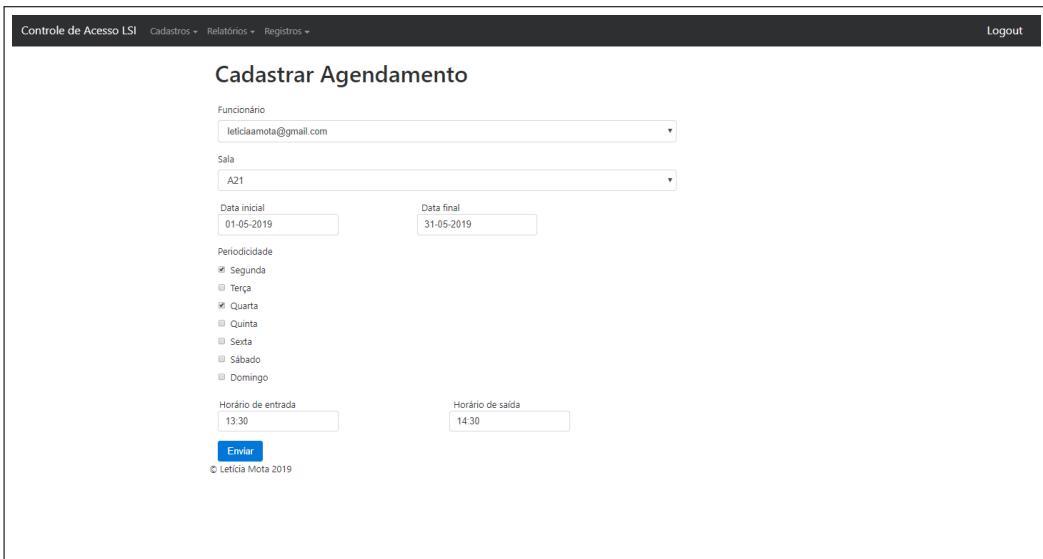
The screenshot shows a table titled 'Registro de Salas' (Room Registration). The columns are labeled 'Código' (Code), 'Descrição' (Description), 'Tópico de Entrada' (Entry Topic), 'Tópico de Saída' (Exit Topic), and 'Editar' (Edit). There are two entries:

Código	Descrição	Tópico de Entrada	Tópico de Saída	
A21	Sala de computação	topicoEntrada/A21	topicoSaída/A21	<a href="#">Editar</a>
A22	Laboratório de co...	topicoEntrada/A22	topicoSaída/A22	<a href="#">Editar</a>

At the bottom left, it says '© Leticia Mota 2019'.

Em relação aos agendamentos, foi implementada a funcionalidade correspondente à sua inserção. Na tela resultante, ilustrada na Figura 4.15, há menus *drop-down* para escolha do usuário e sala associados, além dos campos data inicial e final para registros de agendamentos, opções de dias para registrar periodicidade e também campos para definição de horários de entrada e saída, os quais marcam o período de acesso do usuário ao ambiente.

Figura 4.15: Página de cadastro de agendamento.



The screenshot shows a form titled 'Cadastrar Agendamento' (Schedule Registration). It includes fields for 'Funcionário' (Employee) set to 'leticiaamota@gmail.com', 'Sala' (Room) set to 'A21', 'Data inicial' (Initial Date) set to '01-05-2019', 'Data final' (Final Date) set to '31-05-2019', 'Periodicidade' (Frequency) with 'Segunda' (Monday) checked, 'Horário de entrada' (Entry Time) set to '13:30', 'Horário de saída' (Exit Time) set to '14:30', and an 'Enviar' (Send) button. At the bottom left, it says '© Leticia Mota 2019'.

Por último, quanto à pontualidade, são apresentadas informações que associam os agendamentos realizados aos acessos concedidos, gerando o relatório apresentado na Figura 4.16. Esta tela representa a visualização de registros do ponto de vista do administrador, em que há a escolha do usuário e, em seguida, são mostrados o cumprimento ou não dos seus últimos agendamentos. Para um usuário simples, tem-se a tela apenas com seus dados conforme Figura 4.17. E de maneira gráfica, é possível visualizar estas informações a partir da página ilustrada na Figura 4.18.

Figura 4.16: Listagem de registros de agendamentos.

The screenshot shows a table titled "Relatório de Pontualidade e Frequência" with the following data:

Funcionário	Sala	Data agendamento	Horário de entrada	Horário de saída	Comparecimento	Horário de chegada	Pontualidade
leticiamota@gmail...	A21	21-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	16-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	15-05-2019	00:25:00	01:25:00	<input checked="" type="checkbox"/>	00:26:00	<input checked="" type="checkbox"/>
leticiamota@gmail...	A21	14-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	14-05-2019	15:15:00	16:15:00	<input checked="" type="checkbox"/>	15:16:00	<input checked="" type="checkbox"/>
leticiamota@gmail...	A21	04-05-2019	19:20:00	20:30:00	<input checked="" type="checkbox"/>	20:21:00	<input type="checkbox"/>

6 registros encontrados  
© Leticia Mota 2019

Figura 4.17: Listagem de registros de pontualidade e frequência.

The screenshot shows a table titled "Relatório de Pontualidade e Frequência" with the following data:

Funcionário	Sala	Data agendamento	Horário de entrada	Horário de saída	Comparecimento	Horário de chegada	Pontualidade
leticiamota@gmail...	A21	21-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	16-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	15-05-2019	00:25:00	01:25:00	<input checked="" type="checkbox"/>	00:26:00	<input checked="" type="checkbox"/>
leticiamota@gmail...	A21	14-05-2019	23:10:00	23:59:00	<input type="checkbox"/>	None	<input type="checkbox"/>
leticiamota@gmail...	A21	14-05-2019	15:15:00	16:15:00	<input checked="" type="checkbox"/>	15:16:00	<input checked="" type="checkbox"/>
leticiamota@gmail...	A21	04-05-2019	19:20:00	20:30:00	<input checked="" type="checkbox"/>	20:21:00	<input type="checkbox"/>

6 registros encontrados  
© Leticia Mota 2019

Figura 4.18: Estatísticas de registros de pontualidade e frequência.



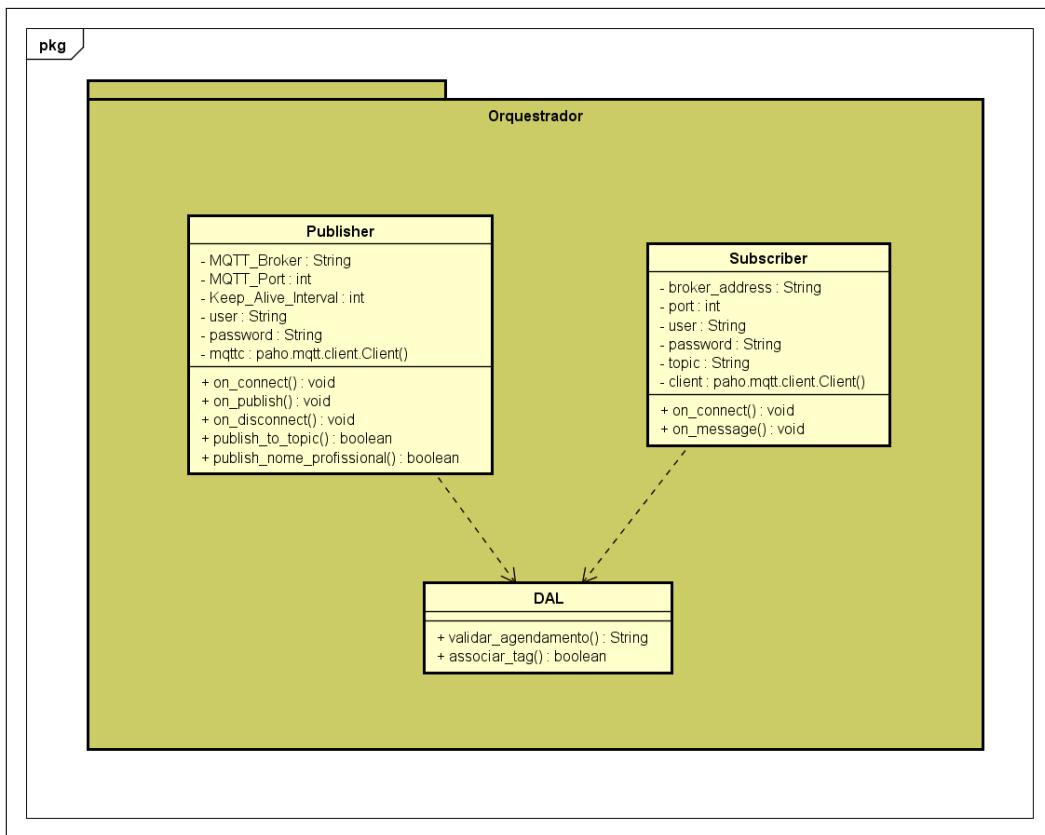
Ao final, métricas foram obtidas ao final desta implementação, com vistas a refletir aspectos quantitativos da aplicação web desenvolvida. Citam-se: 613 linhas de código Python implementadas; 13 *views*, isto é, páginas HTML com as quais o usuário irá interagir; 18 funções no *Controller* da arquitetura da aplicação; e, por fim, 111 linhas de código CSS para personalização da interface com o usuário com vistas a melhoria de usabilidade.

## 4.5 Implementação do Orquestrador

A implementação do módulo orquestrador baseia-se na estruturação das classes *Publisher*, *Subscriber* e *DAL* para a comunicação com o objeto inteligente, segundo o protocolo MQTT, e interação com o banco de dados do sistema.

Este módulo orquestrador deve ser executado no servidor e sempre ficará ativo, sendo a classe *Subscriber* responsável por receber as mensagens enviadas pelos protótipos, atuando como um *event listener*. Para análise e persistência de informações no banco de dados do sistema, a classe *DAL*, referente à *Data Access Layer*, é utilizada, enquanto a classe *Publisher* é empregada para proceder com respostas direcionadas ao objeto inteligente. A relação entre essas classes pode ser observada no diagrama de classe do módulo orquestrador, ilustrado na Figura 4.19.

Figura 4.19: Diagrama de classes do módulo orquestrador.



As principais funcionalidades do orquestrador consistem na associação de *tags* à usuários e validação dos agendamentos. Em ambas as situações, o objeto inteligente identifica uma *tag* e publica essa informação em um tópico de acordo com a funcionalidade requisitada, estando

a classe *Subscriber* aguardando esta publicação para então proceder com o processamento das informações ali contidas utilizando a classe *DAL* como uma auxiliar para fazer as consultas necessárias ao banco de dados. No tocante à classe *DAL*, foram definidas duas funções relativas às consultas necessárias: *associar\_tag()*, para associar uma *tag* lida no objeto inteligente ao respectivo usuário; e *validar\_agendamento()*, que consiste em checar se há agendamento previamente cadastrado para o usuário associado à *tag* lida na data e horário em questão. Por fim, a classe *Publisher* retorna ao protótipo a resposta obtida destas consultas, que serão processadas pelo objeto e produzirão saídas no visor LCD ou acionamento do módulo de relé para liberação de acesso.

A implementação deste módulo foi realizada na linguagem Python e fez uso das bibliotecas *paho.mqtt.client*, para conexão ao Broker MQTT; *json*, para formatação de mensagens trocadas entre objetos e orquestrador; e *sqlite3*, que permite a utilização de funções SQL para consulta e persistência de informação ao banco de dados. O desenvolvimento deste módulo proporciona a integração entre os objetos inteligentes e os demais componentes da solução proposta. As consultas realizadas diretamente em SQL pela classe *DAL*, em especial, promovem um baixo tempo de resposta nas requisições feitas pelo usuário junto aos objetos, colaborando para um controle de acesso mais eficiente. O código-fonte completo das classes integrantes do módulo orquestrador encontra-se disponível no Apêndice C.

## 4.6 Análise Comparativa

Uma vez implementados todos os componentes da solução previamente apresentados, colaborando no processo de validação, partiu-se então para uma análise destes resultados em relação aos trabalhos análogos existentes na literatura, abordados anteriormente na Seção 2.6. Uma síntese desta análise comparativa encontra-se apresentada na Tabela 4.5.

No tocante às similaridades entre a solução propostas e os trabalhos já existentes na literatura, menciona-se o uso do Ambiente Arduino e a existência de uma aplicação web para integrar as informações de gerenciamento. Em relação às diferenças, menciona-se o uso do protocolo MQTT, a existência de um orquestrador de dados, a produção de relatórios de pontualidade e

maior personalização na criação de agendamentos. Menciona-se ainda a natureza *open-source* do projeto, que permite a sua livre reprodução e modificação.

Tabela 4.5: Tabela comparativa da solução proposta em relação a outras alternativas existentes.

<b>Projeto</b>	Plataforma embarcada	<b>Características</b>		
		Padrão de comunicação	Gerenciamento de dados Web	Aplicação Web
Solução Proposta (ARAÚJO et al., 2016) (TEIXEIRA, 2011) (FONSECA; SILVA; MORAES, 2017) (ALMEIDA et al., 2018)	NodeMCU	MQTT	Sim	Sim
	PIC	Serial (Tx de Freq. 9600)	Sim	Não
	Microcomputador	Serial (RS-232)	Não	Não
	Arduino	TCP-IP (Ethernet Shield W5100)	Sim	Não
	Arduino	Web-service	Sim	Sim

<b>Projeto</b>	<b>Características</b>		
	Inspeção visual por LCD	Escalável à múltiplos ambientes	Código aberto para reprodução
Solução Proposta (ARAÚJO et al., 2016) (TEIXEIRA, 2011) (FONSECA; SILVA; MORAES, 2017) (ALMEIDA et al., 2018)	Sim	Sim	Sim
	Sim	Não especificado	Não
	Não	Sim	Não
	Não	Não especificado	Não
	Não	Sim	Não

De maneira geral, ressalta-se que a solução proposta mostra-se especialmente adaptável e escalável em cenários com múltiplos ambientes a serem controlados, utilizando uma comunicação sem fio, arbitrando a comunicação entre elementos do sistema e minimizando as possibilidades de colisão ou perda de mensagens, quer seja pela definição dos tópicos ou pelo padrão MQTT adotado. No âmbito de IoT, além de lidar com o gerenciamento dos ambientes e das regras de acesso, provê uma informação adicional que pode aumentar o apelo pela sua utilização prática: a geração de relatórios de pontualidade. Esta funcionalidade, em especial, não encontra-se presente nas soluções análogas consultadas na literatura.

# Capítulo 5

## Considerações Finais

Este trabalho consistiu no desenvolvimento de uma proposta de controle de acesso a múltiplos ambientes baseada em Internet das Coisas. Tomou-se como ponto de partida o desejo de mitigar as dificuldades típicas de instituições de ensino em que a cada tempo de aula um funcionário específico realiza a abertura das salas para uso docente, causando possível sobrecarga ao funcionário e eventuais atrasos em horário de pico. A solução proposta, entretanto, pode contemplar outros cenários além deste inicialmente considerado, desde que abranjam múltiplos ambientes, disponibilidade de rede *Wi-Fi*, agendamento único de usuário para ambiente em horário pré-estabelecido e possibilidade de monitoramento da pontualidade.

O desenvolvimento da solução proposta culminou na criação de um sistema de controle de acesso composto por objetos inteligentes, com projeto de *hardware* e *software* associado, um painel administrativo em forma de aplicação web e também um orquestrador, não visível aos usuários, mas que administra a saída de dados do protótipo e permite uma integração com as informações produzidas e exibidas na aplicação web. A comunicação entre estas partes ocorre com o uso do protocolo de comunicação MQTT, a partir de tópicos padronizados, evitando a colisão de informações que trafegam no sistema, provenientes dos múltiplos objetos inteligentes existentes. Além destas características, a solução proposta conjuga uma arquitetura de IoT e considera também a utilização de tecnologias como RFID, plataforma Python e ambiente de desenvolvimento do Arduino.

O projeto para o desenvolvimento dos objetos inteligentes, assim como para os demais mó-

dulos desta solução, foram definidos e implementados no escopo deste trabalho como prova de conceito. Também foi realizado um comparativo entre a solução desenvolvida e soluções análogas à esta encontradas na literatura, em que foi possível destacar o baixo custo dos objetos inteligentes por conta da configuração de *hardware* escolhida, a adequação a múltiplos ambientes e o caráter de código aberto e de livre reprodução. Em relação à este último ponto, o repositório do projeto contempla instruções para configuração do ambiente e execução dos módulos: <<https://github.com/leticiamota-uea/leticiamota-tcc>>.

Alguns resultados obtidos no decorrer deste trabalho foram compartilhados em um artigo completo publicado no VIII Encontro Regional de Computação e Sistemas de Informação (MOTA; GUEDES, 2019). Porém, neste então, com vistas a concluir o trabalho proposto, algumas melhorias já foram incorporadas e apresentadas ao longo deste texto.

Em trabalhos futuros, sugere-se a adição de regras de negócio que permitam aos usuários realizarem seus próprios agendamentos. Além disso, sugere-se a evolução do design do protótipo do objeto, por exemplo, por meio de impressões 3D, que permitam encapsular melhor a fiação e posicionar o *display* mais adequadamente, tornando o objeto mais ergonômico para os usuários.

O desenvolvimento deste trabalho considerou um problema prático e permitiu a aplicação de tecnologias de vanguarda relacionadas à Internet das Coisas. Em especial, para a formação da aluna que o desenvolveu, destacam-se a fundamentação teórica adquirida em matérias do curso de graduação, a citar: Engenharia de *Software*, para a especificação de requisitos e diagramação dos mesmos; Bancos de Dados, para a definição da estrutura do banco de dados utilizado e realização de persistência de informações; Redes de Computadores, que contribuiu no projeto de comunicação do sistema a partir do protocolo MQTT; além das disciplinas de Arquitetura e Organização de Computadores e Microcontroladores, que fundamentaram o conhecimento de *hardware* para o desenvolvimento deste projeto.

# Referências Bibliográficas

- ALMEIDA, I. R. de et al. Sistema para reserva de salas e controle de entrada e saída utilizando tecnologias de comunicação sem fio de curto alcance. *Computer On The Beach*, IX, 2018.
- ARAÚJO, P. H. M. et al. Controle de acesso rfid utilizando o princípio de internet das coisas. In: . Picos, Piauí: [s.n.], 2016.
- CHEN, M.; CHEN, S. *RFID Technologies for Internet of Things*. Estados Unidos: Springer, 2016.
- CORREA, R. P. da S. et al. Simulação de aplicações utilizando o protocolo de comunicação mqtt com aplicações em ambientes industriais. *CEEL*, XIV, 2016.
- DUARTE, M. *Primeiros Passos com Web2py: Desenvolvimento Ágil para Web*. Brasil: Independently Published, 2017.
- EINSTRONIC. *Introduction to NodeMCU ESP8266*. 2017. Disponível em <<https://einstronic.com/wp-content/uploads/2017/06/NodeMCU-ESP8266-ESP-12E-Catalogue.pdf>>. Acessado em 24 de junho de 2019.
- ESPRESSIF. *ESP8266EX Datasheet*. 2018. Disponível em <[https://jgamblog.files.wordpress.com/2018/02/esp8266ex\\_datasheet\\_en.pdf](https://jgamblog.files.wordpress.com/2018/02/esp8266ex_datasheet_en.pdf)>. Acessado em 24 de junho de 2019.
- FONSECA, J. A.; SILVA, L. da; MORAES, W. dos S. *Protótipo de Sistema de Identificação: Aplicação da Plataforma Arduino para controle de acesso*. 2017. Monografia (Bacharel em Sistema de Informação), FAPI(Faculdade de Pindamonhangaba), São Paulo, Brasil.
- FRITZING. *Fritzing - Electronics made easy*. 2018. Disponível em: <<http://fritzing.org/download/>>.
- GARCIA, K. M. *Sistema de Controle de Acesso Veicular Utilizando a Tecnologia RFID*. 2013. Monografia (Especialista em Desenvolvimento de Produtos Eletrônicos), IFECT SC(Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina), Santa Catarina, Brasil.
- GLOVER, B.; BHATT, H. *RFID Essentials*. Estados Unidos: O'Reilly, 2006.
- GODOY, P. V. C. *Tecnologia RFID: Uma proposta de sistematização na gestão hospitalar*. 2011. Monografia (Bacharel em Engenharia Elétrica), USP(Universidade de São Paulo - São Carlos), São Paulo, Brasil.
- HILLAR, G. C. *MQTT Essentials - A Lightweight IoT Protocol*. Reino Unido: Packt Publishing, 2017.

- IEEE. *Towards a definition of the Internet of Things (IoT)*. 2015. Disponível em <<https://iot.ieee.org/>>. Acessado em 24 de junho de 2019.
- II, P. J. S. *RFID for Dummies*. Estados Unidos: Wiley, 2005.
- JAVED, A. *Building Arduino Projects for the Internet of Things*. Estados Unidos: Apress, 2016.
- LEHPAMER, H. *RFID Design Principles*. Estados Unidos: Artech House, 2012.
- LONGO, L. *Internet das Coisas: Uso de Sensores e Atuadores na Automação de um Protótipo Residencial*. Pato Branco, Paraná, Brasil: [s.n.], 2015. Trabalho de Conclusão de Curso, Bacharelado em Engenharia de Computação. Universidade Tecnológica Federal do Paraná.
- MORAIS, J. *O que é ESP8266 - A família ESP e o Node MCU*. 2017. Disponível em <<https://portal.vidadesilicio.com.br/o-que-esp8266-nodemcu/>>. Acessado em 24 de junho de 2019.
- MOTA, L. da S.; GUEDES, E. B. Controle de acesso a múltiplos ambientes com internet das coisas. In: . Manaus, Amazonas: FUCAPI, 2019. p. 1–10.
- P, P. *A Beginner's Guide to the ESP8266*. 2017. Disponível em <<https://tttapa.github.io/ESP8266/Chap01%20-%20ESP8266.html>>. Acessado em 24 de junho de 2019.
- PANDIAN, M. P. *RFID for Libraries: A Practical Guide*. Estados Unidos: Chandos, 2010.
- PEREIRA, C. E. P.; CARVALHO, F. V. de. A internet das coisas (iot): Cenário e perspectivas no Brasil e aplicações práticas. In: . Minas Gerais, Brasil: [s.n.], 2017. p. 1–7.
- PIERRO, M. di. *Web2py Enterprise Web Framework*. Estados Unidos: Lulu.com, 2010.
- PRADO, N. R. da S. A.; PEREIRA, N. A.; POLITANO, P. R. Dificuldades para a adoção de rfid nas operações de uma cadeia de suprimentos. *ENEGET*, XXVI, 2006.
- RIBEIRO, P. A. F. *Uma Plataforma Web para Gerenciamento de Dados e Geração de Boletins Meteorológicos do LabInstru*. 2017. Monografia (Bacharel em Engenharia de Computação), UEA(Universidade do Estado do Amazonas), Amazonas, Brasil.
- SAHOO, S. S. *MQTT - A Practical Guide*. 2018. Disponível em <<https://goo.gl/o4xHUF>>. Acessado em 24 de junho de 2019.
- SCHWARTZ, M. *Internet of Things with Arduino*. Reino Unido: Packt Publishing, 2016.
- SCHWARTZ, M. *Internet of Things with ESP8266*. Reino Unido: Packt Publishing, 2016.
- SOLUTIONS, O. M. *Controle de acesso veicular para condomínios - Tag RFID*. 2011. Disponível em: <<http://www.oxxcode.com.br/controle-de-acesso-condominio-tag-rfid/>>.
- TECHNOLOGY, H. *ESP8266 NodeMCU WiFi Devkit*. 2017. Disponível em: <<http://www.handsontec.com/pdf\learn/esp8266-V10.pdf>>.

TEIXEIRA, T. *Controle de Fluxo de Pessoas Usando RFID*. 2011. Monografia (Tecnólogo em Sistemas de Telecomunicações), IFSC (Instituto Federal de Santa Catarina), Santa Catarina, Brasil.

THOMSEN, A. *Como programar o módulo ESP8266 NodeMCU*. 2016. Disponível em: <<https://www.filipeflop.com/blog/esp8266-nodemcu-como-programar/>>.



# Apêndice A

## Detalhamento dos Casos de Uso

Quadro A.1: Caso de Uso 1 – Cadastrar Sala

### Caso de Uso CDU01: Cadastrar Sala

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja cadastrar novas salas no sistema.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema.

**Pós-condições (Garantia de Sucesso):**

- Criação de objeto sala para monitoramento.

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. O administrador escolherá a opção “Salas” na aba “Cadastros” do menu principal da aplicação.
2. O sistema exibirá um formulário de cadastro de nova sala que deve ser preenchido pelo administrador.
3. O administrador do sistema informará os seguintes dados da sala a ser cadastrada: Bloco de localização da sala, numero da sala e descrição da sala.
4. O sistema irá criar, a partir das informações concedidas, os campos: código da sala, tópico de entrada, tópico de saída e definirá o status da sala como ativo (true), por padrão.
5. O sistema efetivará a criação do objeto sala.
6. O sistema exibirá ao administrador a mensagem: “Sala cadastrada com sucesso”.

Quadro A.2: (Continuação) Caso de Uso 1 – Cadastrar Sala

**Extensões (ou Fluxo Alternativo):**

- a) **Campos obrigatórios não preenchidos:** Caso algum campo do formulário de cadastro de nova sala não tenha sido preenchido, o sistema notificará o administrador, informando-lhe os campos que não foram preenchidos.
- b) **Sala já cadastrada e ativa:** Caso o administrador informe a combinação de campos “*bloco e número*” que já esteja associado a uma sala ativa no sistema, será emitida uma mensagem de alerta: “*Esta sala já está cadastrada no sistema*”.
- c) **Sala já cadastrada, mas inativa:** Caso o administrador informe a combinação de campos “*bloco e número*” que já esteja associado a uma sala inativa no sistema, será emitida uma caixa de alerta com a mensagem “*Esta sala já está cadastrada, mas inativa no sistema, deseja reativá-la?*” e botões “*ativar*” e “*cancelar*” para que o administrador finalize o procedimento conforme a necessidade.

### Quadro A.3: Caso de Uso 2 – Listar Salas

#### Caso de Uso CDU02: Listar Salas

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja visualizar uma lista contendo as salas cadastradas no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.

#### Pós-condições (Garantia de Sucesso):

- É apresentada uma listagem contendo as salas cadastradas no sistema, ordenadas a partir dos respectivos códigos das mesmas.
- Essa listagem contempla os códigos, blocos, números, descrições, tópicos de entrada e saída e status das salas.
- Ao lado de cada registro de sala cadastrada, encontram-se as opções de editar e inativar sala.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “Salas” na aba “Registros” do menu principal da aplicação.
2. O sistema exibirá o resultado da pesquisa na forma de uma lista de salas.

#### Extensões (ou Fluxo Alternativo):

- a) **Nenhum registro encontrado:** Não foram encontradas salas previamente cadastradas no sistema. Será emitida a mensagem: “Não há salas cadastradas”.

Quadro A.4: Caso de Uso 3 – Editar Sala

### Caso de Uso CDU03: Editar Salas

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja editar o registro de uma sala previamente cadastrada no sistema.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema
2. Ter listado as salas (vide Caso de Uso CDU02 – Listar salas)

**Pós-condições (Garantia de Sucesso):**

- Edição realizada no registro de uma determinada sala.

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. Na linha referente à sala que será feita a edição de seu registro, clicar na opção “*Editar sala*” que é representada pelo ícone de um lápis.
2. Um formulário de edição dos dados da sala será apresentado, exibindo todos os campos passíveis de edição.
3. O administrador modificará os campos que julgar necessário e clicará no botão “*Concluir Edição*”.
4. O sistema registrará as modificações e emitirá a mensagem “*Cadastro alterado com sucesso*”.

**Regras de Negócio:**

RN01 - O formulário de edição apresentará todos os campos referentes à sala, mas os campos “*código da sala*”, “*tópico de entrada*”, “*tópico de saída*” e “*status*” não poderão ser modificados.

Quadro A.5: Caso de Uso 4 – Inativar Registro de Sala

### Caso de Uso CDU04: Inativar Registro de Salas

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja inativar o registro de uma sala previamente cadastrada no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema
2. Ter listado as salas (vide Caso de Uso CDU02 – Listar salas)

#### Pós-condições (Garantia de Sucesso):

- Inativação de registro de uma determinada sala.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. Na linha referente à sala que será feita a inativação de seu registro, clicar na opção “Inativar sala” que é representada pelo ícone de uma lixeira.
2. Ao clicar no ícone de inativação, uma mensagem de confirmação é exibida: “Deseja inativar a sala?”.
3. Caso o administrador confirme a inativação, o sistema irá alterar o campo status da sala para “false” e emitirá a mensagem “Sala inativada com sucesso”.

#### Extensões (ou Fluxo Alternativo):

- a) **Negação na mensagem de inativação de sala:** Caso o administrador responda não para a pergunta “Deseja inativar a sala?”, o sistema abortará o processo de inativação e retornará à página contendo a listagem das salas.

Quadro A.6: Caso de Uso 5 – Cadastrar Usuário

### Caso de Uso CDU05: Cadastrar Usuário

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja cadastrar novos usuários no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.

#### Pós-condições (Garantia de Sucesso):

- Criação novo usuário no sistema.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “*Usuários*” na aba “*Cadastros*” do menu principal da aplicação.
2. O sistema exibirá um formulário de cadastro de novo usuário que deve ser preenchido pelo administrador.
3. O administrador do sistema informará os seguintes dados do usuário a ser cadastrado: Nome, Sobrenome, e-mail, telefone e função.
4. O sistema irá criar automaticamente uma senha para o usuário. Esta senha é gerada de maneira aleatória por 6 dígitos numéricos
5. O sistema irá preencher o campo *Tag* como NULL, para posterior associação *Tag-Usuário*, e definirá o status do usuário como ativo (true), por padrão.
6. O sistema exibirá ao administrador a mensagem: “*Usuário cadastrado com sucesso*”.

#### Extensões (ou Fluxo Alternativo):

- a) **Campos obrigatórios não preenchidos:** Caso algum campo do formulário de cadastro de novo usuário não tenha sido preenchido, o sistema notificará o administrador, informando-lhe os campos que não foram preenchidos.
- b) **Usuário já cadastrado e ativo:** Caso o administrador informe no campo “*e-mail*” um registro que já esteja cadastrado no sistema, será emitida uma mensagem de alerta: ““*Este usuário já está cadastrado no sistema*”.
- c) **Usuário já cadastrado, mas inativo:** Caso o administrador informe no campo “*e-mail*” um registro que já esteja cadastrado no sistema, será emitida uma caixa de alerta com a mensagem “*Este usuário já está cadastrado, mas inativo no sistema, deseja reativá-lo?*” e botões “*ativar*” e “*cancelar*” para que o administrador finalize o procedimento conforme a necessidade.

### Quadro A.7: Caso de Uso 6 – Listar Usuários

#### Caso de Uso CDU06: Listar Usuários

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja visualizar uma lista contendo os usuários cadastrados no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.

#### Pós-condições (Garantia de Sucesso):

- É apresentada uma listagem contendo os usuários cadastrados no sistema, ordenados a partir dos respectivos nomes dos usuários.
- Essa listagem contempla os Nomes, Sobrenomes, e-mails, telefones, funções, *tags* e status do cadastro do usuário.
- Ao lado de cada registro de usuário cadastrado, encontram-se as opções de editar e inativar usuário.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “*Usuários*” na aba “*Registros*” do menu principal da aplicação.
2. O sistema exibirá o resultado da pesquisa na forma de uma lista de usuários.

#### Extensões (ou Fluxo Alternativo):

- a) **Nenhum registro encontrado:** Não foram encontrados usuários previamente cadastrados no sistema. Será emitida a mensagem: *“Não há usuários cadastrados”*.

### Quadro A.8: Caso de Uso 7 – Editar Usuário

#### Caso de Uso CDU07: Editar Usuários

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja editar o registro de um usuário previamente cadastrado no sistema.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema
2. Ter listado os usuários (vide Caso de Uso CDU06 – Listar usuários)

**Pós-condições (Garantia de Sucesso):**

- Edição realizada no registro de um determinado usuário.

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. Na linha referente ao usuário que será feita a edição de seu registro, clicar na opção “*Editar usuário*” que é representada pelo ícone de um lápis.
2. Um formulário de edição dos dados do usuário será apresentado, exibindo todos os campos passíveis de edição.
3. O administrador modificará os campos que julgar necessário e clicará no botão “*Concluir Edição*”.
4. O sistema registrará as modificações e emitirá a mensagem “*Cadastro alterado com sucesso*”.

**Regras de Negócio:**

RN01 - O formulário de edição apresentará todos os campos referentes ao usuário, mas os campos “*senha*” e “*status*” não poderão ser modificados.

Quadro A.9: Caso de Uso 8 – Inativar Registro de Usuário

### Caso de Uso CDU08: Inativar Registro de Usuário

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja inativar o registro de um usuário previamente cadastrado no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema
2. Ter listado os usuários (vide Caso de Uso CDU06 – Listar usuários)

#### Pós-condições (Garantia de Sucesso):

- Inativação de registro de um determinado usuário.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. Na linha referente ao usuário que será feita a inativação de seu registro, clicar na opção “Inativar usuário” que é representada pelo ícone de uma lixeira.
2. Ao clicar no ícone de inativação, uma mensagem de confirmação é exibida: “Deseja inativar o usuário?”.
3. Caso o administrador confirme a inativação, o sistema irá alterar o campo status do usuário para “false” e emitirá a mensagem “Usuário inativado com sucesso”.

#### Extensões (ou Fluxo Alternativo):

- a) **Negação na mensagem de inativação de sala:** Caso o administrador responda não para a pergunta “Deseja inativar o usuário?”, o sistema abortará o processo de inativação e retornará à página contendo a listagem dos usuários.

Quadro A.10: Caso de Uso 9 – Associar *Tag-Usuário*

### Caso de Uso CDU09: Associar *Tag-Usuário*

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja associar uma *Tag* a um usuário previamente cadastrado no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.
2. Existir usuário cadastrado no sistema (vide Caso de Uso CDU05 – Cadastrar usuário).
3. Existir sala cadastrada no sistema (vide Caso de Uso CDU01 – Cadastrar sala).

#### Pós-condições (Garantia de Sucesso):

- Cadastro de *Tag*, no perfil do usuário, realizado com sucesso.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “Associar Tag-Usuário” na aba “Cadastros” do menu principal da aplicação.
2. O sistema exibirá um formulário referente ao cadastro associativo da *tag* ao usuário.
3. O administrador do sistema selecionará o email do usuário ao qual a *tag* será associada e o código da sala onde a leitura da *tag* será feita, conforme a disponibilidade informada pelo sistema.
4. O sistema irá disponibilizar o código da sala para seleção caso esta esteja disponível (não esteja sendo usada por algum usuário ou não esteja reservada para algum usuário).
5. Após a seleção de ambos os campos, o administrador terá a opção de “Ativar Leitor de Tag” ou “Cancelar”.
6. Ao Ativar o Leitor de *Tag*, o módulo da sala escolhida será ativado e ficará aguardando a apresentação da *tag* para finalizar a associação.
7. O administrador deverá ir até o módulo escolhido para finalizar a associação *Tag-Usuário*.
8. Após a leitura da *tag* o sistema enviará ao módulo a mensagem “Associação realizada com sucesso” que será mostrada no display do módulo.

Quadro A.11: (Continuação) Caso de Uso 9 – Associar *Tag-Usuário*

**Extensões (ou Fluxo Alternativo):**

- a) **Negação na mensagem de ativar o leitor de *Tag*:** Caso o administrador na opção “*Cancelar*” no formulário de associação de *tag*, o sistema abortará o processo de associação e retornará à página principal da aplicação.
- b) ***Tag* já associada:** Caso a *tag* lida pelo módulo já esteja associada a um usuário, o sistema informara a mensagem “*Tag já está associada*” e abortará o processo de associação.

Quadro A.12: Caso de Uso 10 – Associar *Tag-Mestra*

### Caso de Uso CDU10: Associar *Tag-Master*

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja associar uma *Tag-Master* para administração do sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.

#### Pós-condições (Garantia de Sucesso):

- Cadastro de *Tag-Master* realizado com sucesso.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “Associar *Tag-Master*” na aba “Cadastrros” do menu principal da aplicação.
2. O sistema exibirá um formulário referente ao cadastro associativo da *tag* ao Administrador.
3. O administrador do sistema selecionará código da sala onde a leitura da *tag* será feita, conforme a disponibilidade informada pelo sistema.
4. O sistema irá disponibilizar o código da sala para seleção caso esta esteja disponível (não esteja sendo usada por algum usuário ou não esteja reservada para algum usuário).
5. Após a seleção da sala, o administrador terá a opção de “Ativar Leitor de Tag” ou “Cancelar”.
6. Ao Ativar o Leitor de *Tag*, o módulo da sala escolhida será ativado e ficará aguardando a apresentação da *tag* para finalizar a associação.
7. O administrador deverá ir até o módulo escolhido para finalizar a associação *Tag-Master*.
8. Após a leitura da *tag* o sistema enviará ao módulo a mensagem “Associação realizada com sucesso” que será mostrada no display do módulo.

#### Extensões (ou Fluxo Alternativo):

- a) **Negação na mensagem de ativar o leitor de *Tag*:** Caso o administrador na opção “Cancelar” no formulário de associação de *tag*, o sistema abortará o processo de associação e retornará à página principal da aplicação.
- b) ***Tag* já associada:** Caso a *tag* lida pelo módulo já esteja associada a um usuário, o sistema informara a mensagem “*Tag* já está associada” e abortará o processo de associação.

Quadro A.13: Caso de Uso 11 – Cadastrar / Editar Agendamento

### Caso de Uso CDU11: Cadastrar / Editar Agendamento

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja cadastrar, ou editar, um Agendamento no sistema.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema.

**Pós-condições (Garantia de Sucesso):**

- Cadastro de Agendamento realizado com sucesso.

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. O administrador escolherá a opção “Agendamentos” na aba “Cadastros” do menu principal da aplicação.
2. O sistema exibirá um formulário de cadastro de agendamento para um usuário que deve ser preenchido pelo administrador.
3. O administrador do sistema selecionará a sala na qual o agendamento será realizado, o nome do usuário cujo agendamento será cadastrado, incluirá os campos: horário de entrada, horário de saída, código da sala, dias de repetição e período de repetição.
4. Ao finalizar o preenchimento do formulário o administrador deve clicar em “Finalizar Agendamento”.
5. O sistema emitirá a seguinte mensagem em caixa de diálogo: “Confirmar Cadastro de Agendamento?” com os botões “Confirmar” e “Cancelar”.
6. O administrador clicará no botão “Confirmar” para efetivar o cadastro do agendamento.
7. O sistema irá verificar a possibilidade de agendamento e exibirá ao administrador a mensagem: “Agendamento cadastrado com sucesso”.

Quadro A.14: (Continuação) Caso de Uso 11 – Cadastrar / Editar Agendamento

**Extensões (ou Fluxo Alternativo):**

- a) **Campos obrigatórios não preenchidos:** Caso algum campo do formulário de cadastro de novo agendamento não tenha sido preenchido, o sistema notificará o administrador, informando-lhe os campos que não foram preenchidos.
- b) **Choque de Horário:** Caso já exista um agendamento para o dia da semana e horário informados, será emitida uma mensagem de alerta: “*Já existe agendamento ativo para o dia e horário informados*”.
- c) **Negação na mensagem de confirmar agendamento:** Caso o administrador clique em “*Cancelar*” na caixa de diálogo de confirmação, o sistema abortará o processo de cadastro de agendamento e retornará à página principal da aplicação.

Quadro A.15: Caso de Uso 12 – Listar Agendamento

### Caso de Uso CDU12: Listar Agendamento

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja visualizar os agendamentos cadastrados no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.

#### Pós-condições (Garantia de Sucesso):

- É apresentada uma planilha contendo os agendamentos cadastrados no sistema, correspondentes à determinada sala selecionada.
- Essa planilha contempla o Código da Sala, os Dias, Horários e Nome do usuário conforme seus agendamentos.
- Ao lado de cada registro de agendamento cadastrado, encontra-se a opção de excluir um agendamento específico.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. O administrador escolherá a opção “Agendamentos” na aba “Registros” do menu principal da aplicação.
2. O administrador selecionará a sala cujos agendamentos serão exibido.
3. O sistema exibirá o resultado da pesquisa na forma de uma planilha contendo os agendamentos existentes para a sala informada.

#### Extensões (ou Fluxo Alternativo):

- a) **Nenhum agendamento encontrado:** Não foram encontrados agendamentos previamente cadastrados no sistema. Será exibida uma planilha com os valores “disponíveis” conforme os horários cadastrados.

Quadro A.16: Caso de Uso 13 – Excluir Agendamento

### Caso de Uso CDU13: Excluir Agendamento

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador

**Interessados e Interesses:**

- Administrador: Deseja excluir um agendamento previamente cadastrado no sistema.

#### Pré-condições:

1. Administrador precisa estar autenticado no sistema.
2. Ter listado os agendamentos (vide Caso de Uso CDU12 – Listar agendamentos)

#### Pós-condições (Garantia de Sucesso):

- Excluir agendamento de determinada sala/usuário.

#### Cenário de Sucesso Principal (ou Fluxo Básico):

1. Na célula referente ao agendamento que será feita a exclusão, clicar na opção “*Excluir Agendamento*” que é representada pelo ícone de uma lixeira.
2. Ao clicar no ícone de exclusão, uma mensagem de confirmação é exibida: “*Deseja excluir agendamento?*”.
3. Caso o administrador confirme a exclusão, o sistema irá excluir o registro e emitirá a mensagem “*Agendamento excluído com sucesso*”.

#### Extensões (ou Fluxo Alternativo):

- a) **Negação na mensagem de inativação de sala:** Caso o administrador responda não para a pergunta “*Deseja excluir agendamento?*”, o sistema abortará o processo de inativação e retornará à página contendo a listagem de agendamentos.

Quadro A.17: Caso de Uso 14 – Relatório de Pontualidade e Frequência

### Caso de Uso CDU14: Relatório de Pontualidade e Frequência

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador e Usuário

**Interessados e Interesses:**

- Administrador: Deseja visualizar o relatório de pontualidade de determinado usuário.
- Usuário: Deseja visualizar seu relatório de pontualidade.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema.
2. Usuário precisa estar autenticado no sistema.

**Pós-condições (Garantia de Sucesso):**

- É apresentada uma planilha contendo um relatório mensal de pontualidade do usuário.
- Essa planilha contempla Datas, Horários e salas as quais o usuário teve acesso.

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. O administrador ou usuário escolherá a opção “Relatório de Pontualidade” na aba “Relatórios” do menu principal da aplicação.
2. O administrador selecionará o nome do usuário cujo relatório deve ser exibido; No caso do usuário, o sistema definirá o nome do usuário pelo login de acesso.
3. O sistema exibirá o resultado da pesquisa na forma de uma planilha contendo os dados existentes para o usuário informado.

**Extensões (ou Fluxo Alternativo):**

- a) **Nenhum registro encontrado:** Não foram encontrados dados para este usuário no sistema. Será exibida a seguinte mensagem “Usuário sem dados disponíveis”.

Quadro A.18: Caso de Uso 15 – Estatísticas de Pontualidade e Frequência

### Caso de Uso CDU15: Estatísticas de Pontualidade e Frequência

**Escopo:** Aplicação Web Controle de Acesso

**Autor principal:** Administrador e Usuário

**Interessados e Interesses:**

- Administrador: Deseja visualizar estatísticas de pontualidade e frequência do usuário.
- Usuário: Deseja visualizar estatísticas de pontualidade e frequência sobre seus agendamentos.

**Pré-condições:**

1. Administrador precisa estar autenticado no sistema.
2. Usuário precisa estar autenticado no sistema.

**Pós-condições (Garantia de Sucesso):**

- É apresentada uma planilha contendo um relatório acerca da taxa de frequência do usuário em cada mês.
- Essa planilha contempla Mês e Percentual de assiduidade do usuário a seus agendamentos

**Cenário de Sucesso Principal (ou Fluxo Básico):**

1. O administrador ou usuário escolherá a opção “Relatório de Taxa de Frequência” na aba “Relatórios” do menu principal da aplicação.
2. O administrador selecionará o nome do usuário cujo relatório deve ser exibido; No caso do usuário, o sistema definirá o nome do usuário pelo login de acesso.
3. O sistema exibirá o resultado da pesquisa na forma de uma planilha contendo os dados existentes para o usuário informado.

**Extensões (ou Fluxo Alternativo):**

- a) **Nenhum registro encontrado:** Não foram encontrados dados para este usuário no sistema. Será exibida a seguinte mensagem “Usuário sem dados disponíveis”.

# Apêndice B

## Diagramas de Sequência

Figura B.1: Diagrama de sequência para o Cadastro de Sala.

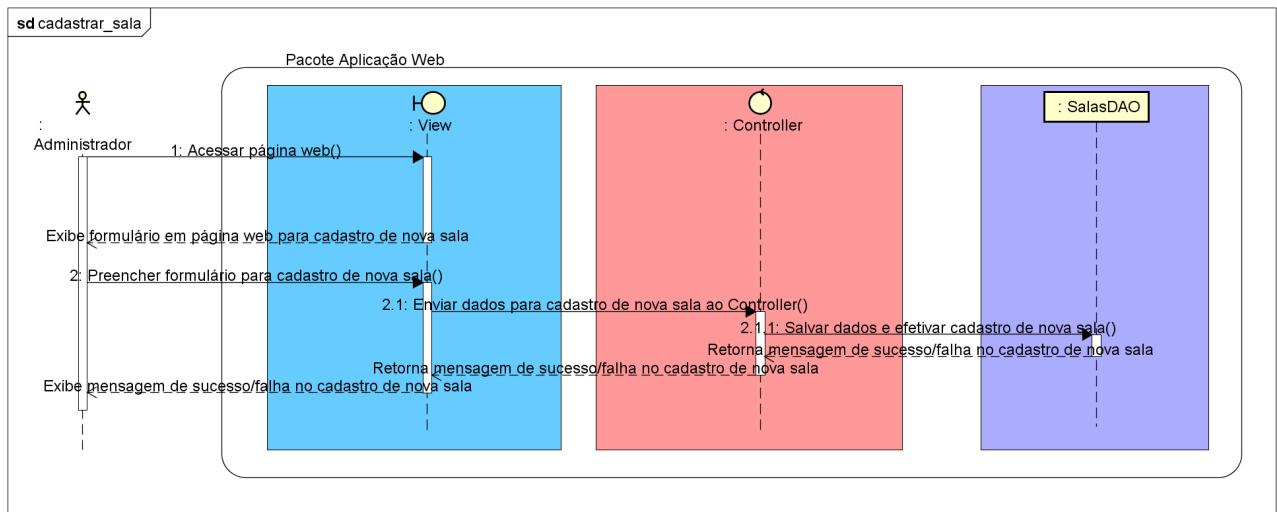


Figura B.2: Diagrama de sequência para a Listagem de Salas Cadastradas.

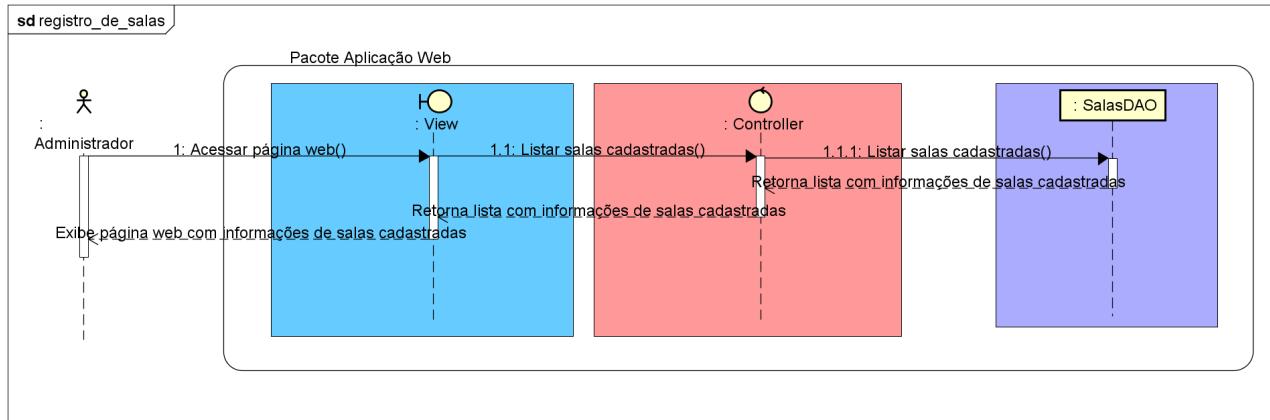


Figura B.3: Diagrama de sequência para a Edição de Cadastro de Sala.

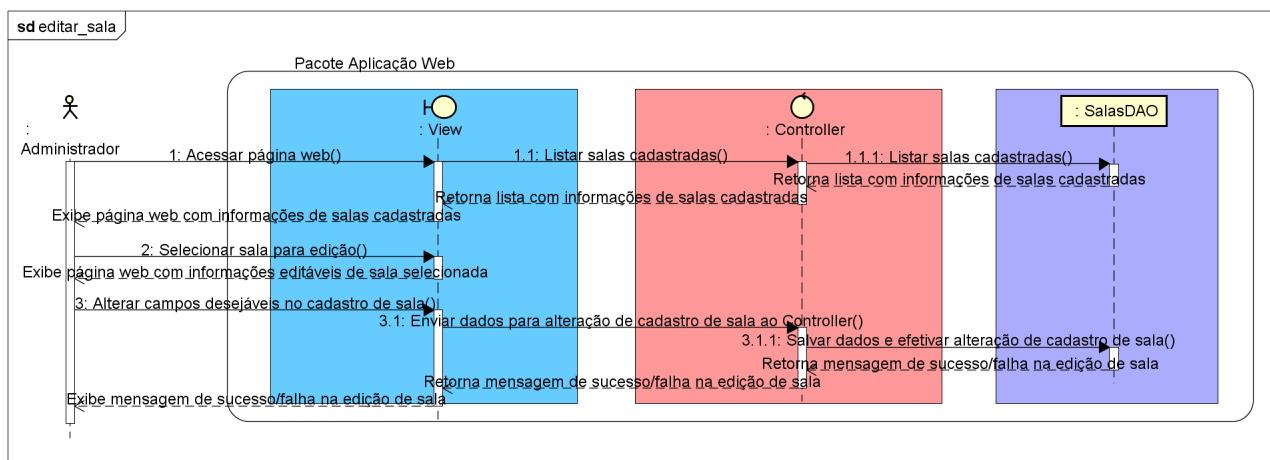


Figura B.4: Diagrama de sequência para a Inativação de Cadastro de Sala.

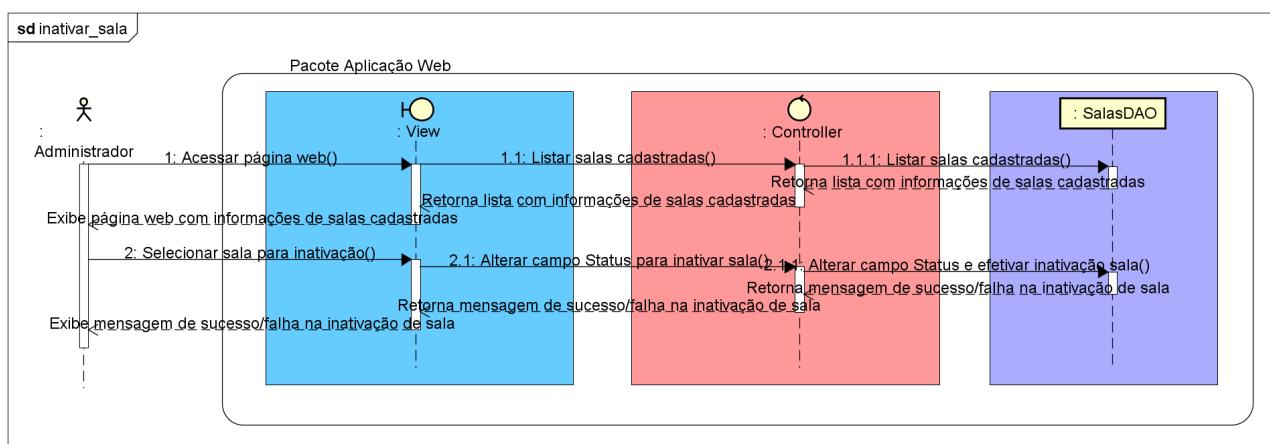


Figura B.5: Diagrama de sequência para o Cadastro de Usuário.

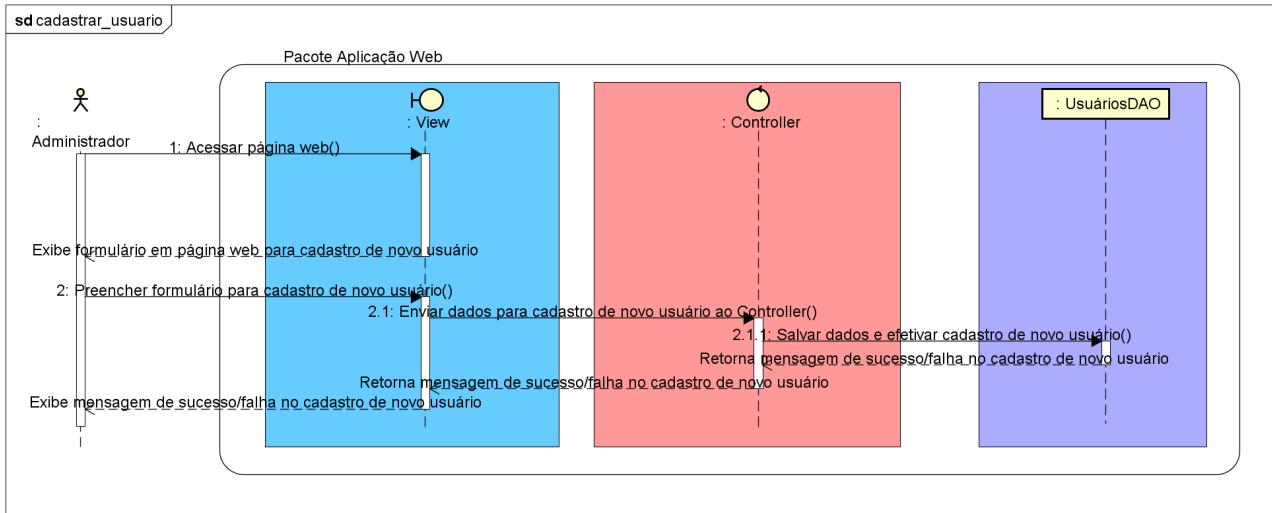


Figura B.6: Diagrama de sequência para a Listagem de Usuários Cadastrados.

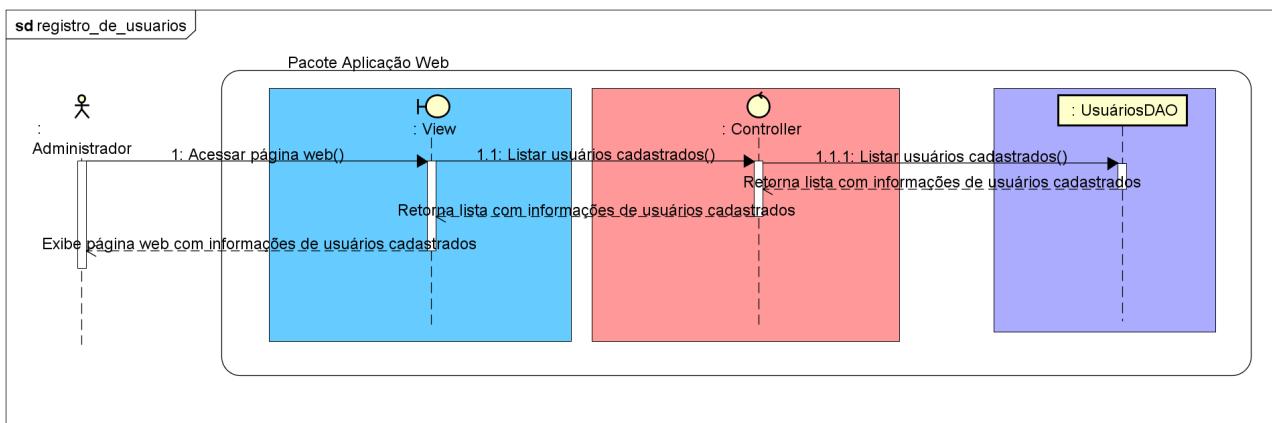


Figura B.7: Diagrama de sequência para a Edição de Cadastro de Usuário.

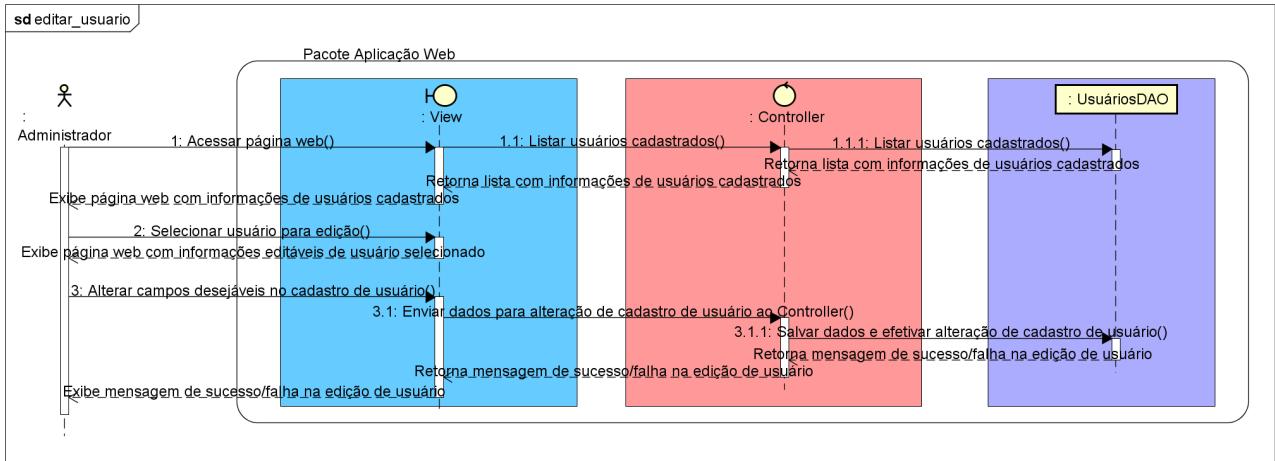


Figura B.8: Diagrama de sequência para a Inativação de Cadastro de Usuário.

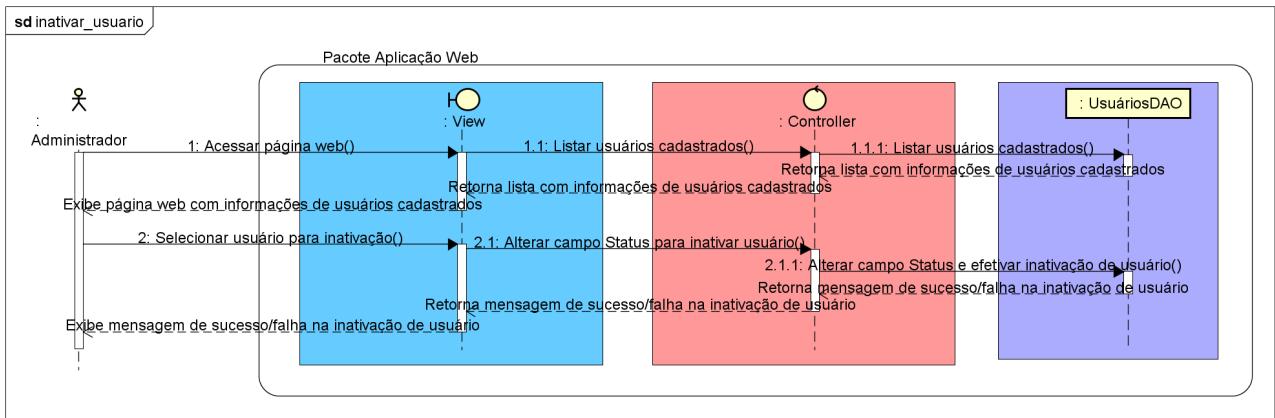


Figura B.9: Diagrama de sequência para o Cadastro de Agendamento.

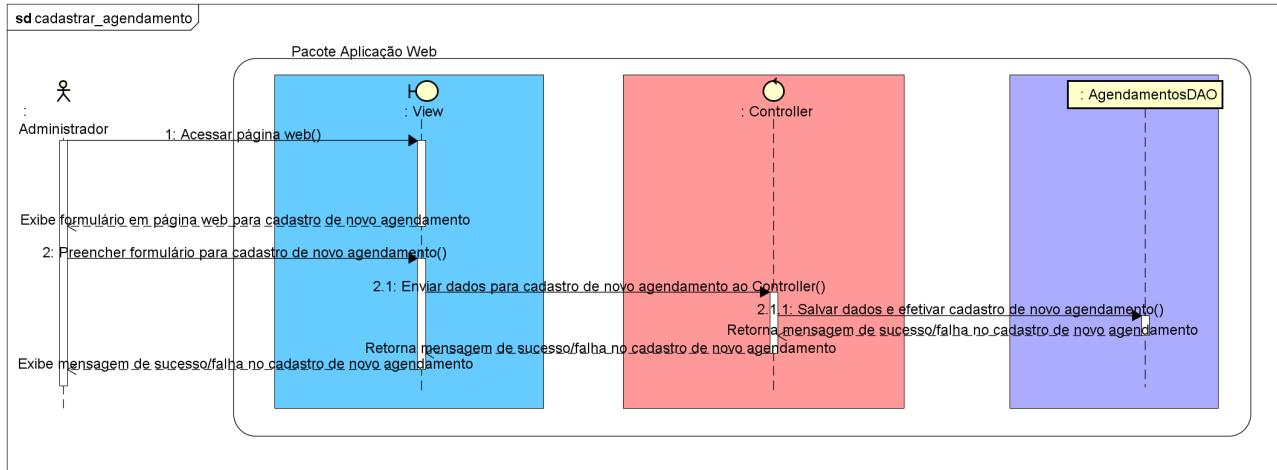


Figura B.10: Diagrama de sequência para a Listagem de Agendamentos Cadastrados.

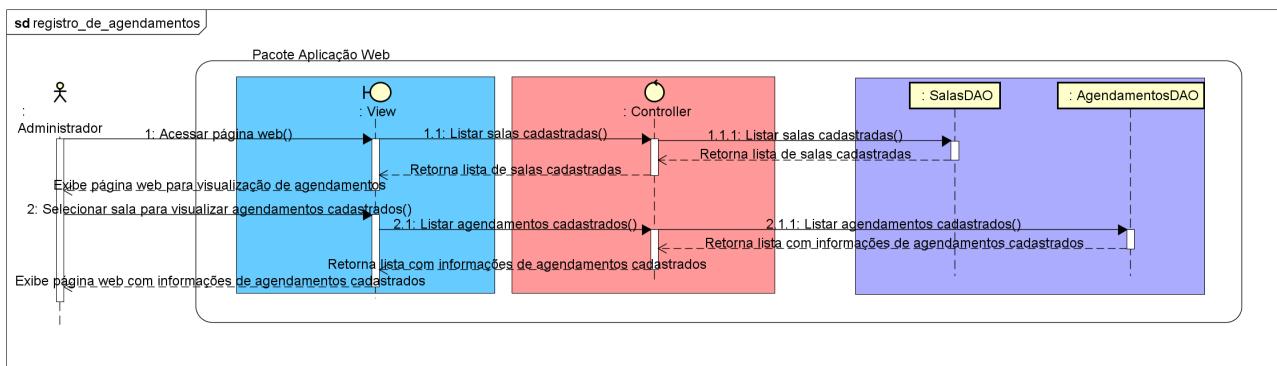


Figura B.11: Diagrama de sequência para a Edição de Cadastro de Agendamento.

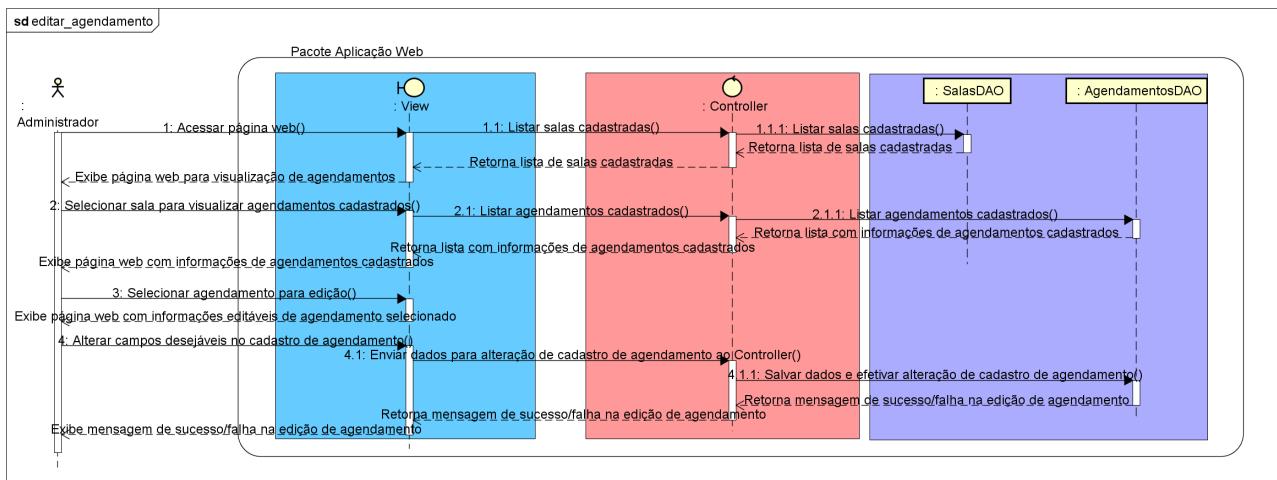


Figura B.12: Diagrama de sequência para a Exclusão de Cadastro de Agendamento.

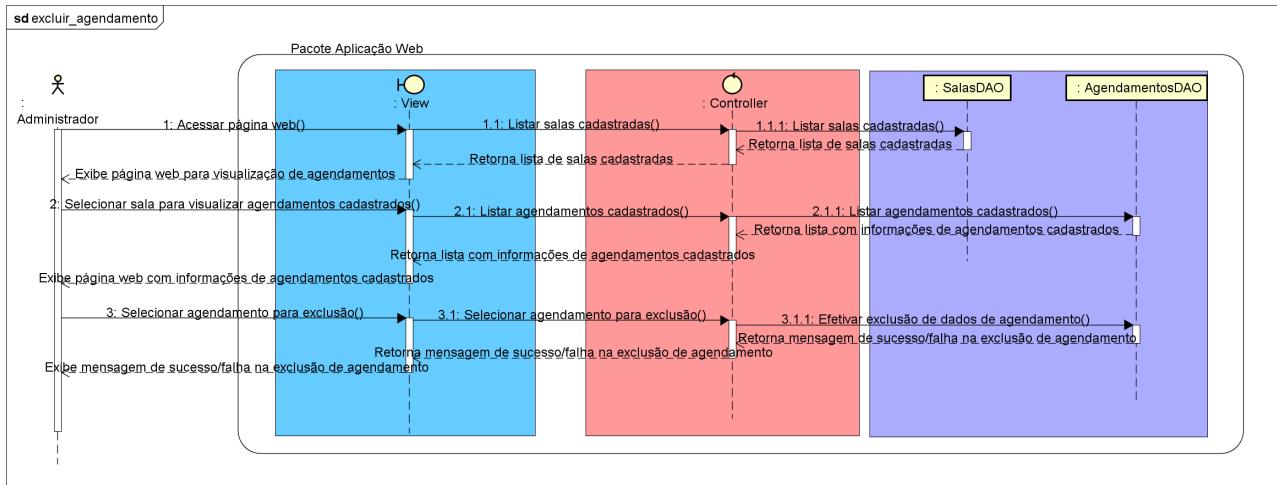


Figura B.13: Diagrama de sequência para Relatório de Pontualidade.

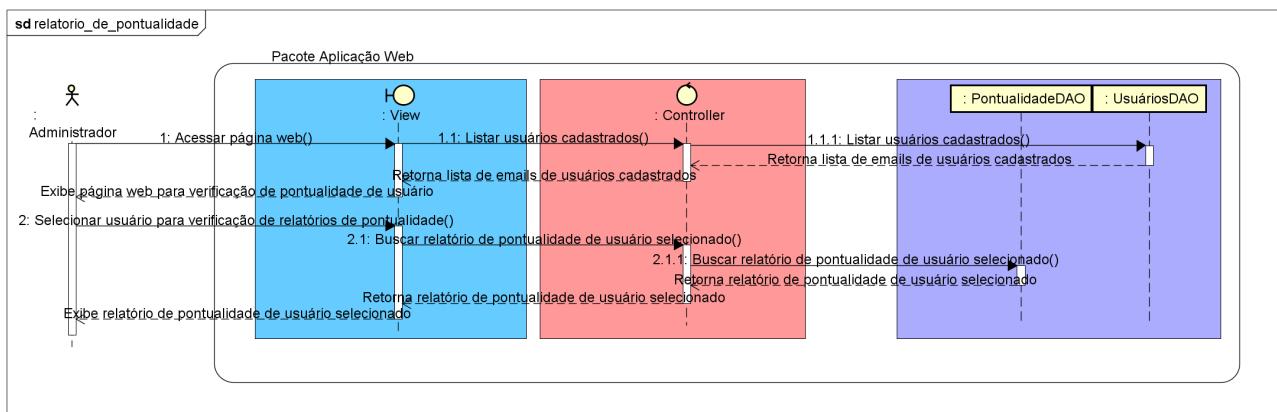
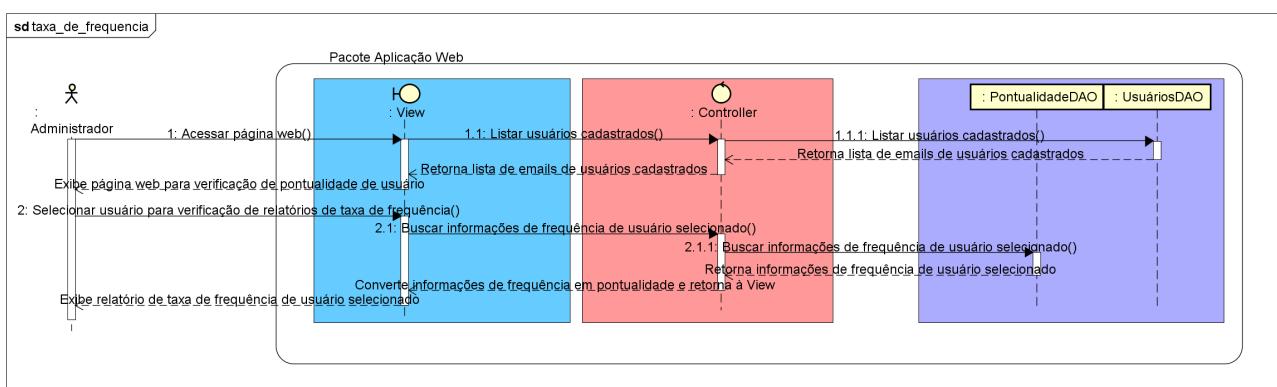


Figura B.14: Diagrama de sequência para Relatório de Taxa de Frequência.





## Apêndice C

# Código-Fonte do Orquestrador Publisher/Subscriber

Figura C.1: Classe do pacote orquestrador responsável por inscrever-se em tópicos de saída de objetos inteligentes.

```

import paho.mqtt.client as mqttClient
import time, json, datetime
from dateutil.parser import parse
from DAL import validar_agendamento
from DAL import associar_tag
from publisher_pc import publish_nome_profissional

#=====
#      MQTT CONNECTION
#=====
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker")
        global Connected
        Connected = True
    else:
        print("Connection failed")

#=====
#      MQTT DATA RECEIVER
#=====
def on_message(client, userdata, message):
    print ("")
    print ("Message received: " + str(message.payload))
    print ("")
    #Separacao de infos contidas em topico
    infoTopico = message.topic.split('/')
    #infoTopico gets topic and subtopics

```

```

#infoTopico[1] gets codSala
#infoTopico[2] gets function

if(infoTopico[2] == "validarAgendamento"):
    #variaveis sala, funcionario, hora_saida(hrsaida)
    sala = infoTopico[1]
    funcionario,hrsaida =
        validar_agendamento(message.topic, message.payload)
    hatual = datetime.datetime.now().strftime("%H:%M:00")

    if (hrsaida != None):
        ##### TIME DELTA DEFINITION #####
        datetime_atual = datetime.datetime.
            strftime(hatual, '%H:%M:%S')
        datetime_saida = datetime.datetime.
            strftime(hrsaida, '%H:%M:%S')
        deltatime=str((datetime_saida-datetime_atual).
            seconds * 1000)
        #####
    else:
        deltatime = None

#definicao de mensagem json
mensagemPython = {
    "funcionario": funcionario,
    "deltaT": deltatime
}

# convert into JSON:
mensagemJson = json.dumps(mensagemPython)

#funcao publish - file: publish_pc
publish_nome_profissional(sala,mensagemJson)

if(infoTopico[2] == "salvarTagUser"):
    status = associar_tag(message.topic, message.payload)
    print("Salvar Tag User STATUS: ", status)
    pass

if(infoTopico[2] == "salvarTagMaster"):
    status = associar_tag(message.topic, message.payload)
    print("Salvar Tag Master STATUS: ", status)
    pass

=====
Connected = False      #global variable for the state of the connection
=====

#      MQTT CONFIG
broker_address= "m12.cloudmqtt.com"      #Broker address
port = 19729                      #Broker port
user = "qxqwnxyw"                  #Connection username
password = "ktJEqMyhPOVx"          #Connection password
topic = "topicoSaida/#"           #Mqtt topic

client = mqttClient.Client("subscriber-pc")      #create new instance
client.username_pw_set(user, password=password)  #set username and password
client.on_connect= on_connect                 #attach function to callback
client.on_message= on_message                #attach function to callback

```

---

```
client.connect(broker_address, port=port)           #connect to broker

#=====
client.loop_start()          #start the loop

while Connected != True:    #Wait for connection
    time.sleep(0.1)

client.subscribe(topic)

try:
    while True:
        time.sleep(1)

except KeyboardInterrupt:
    print ("exiting")
    client.disconnect()
    client.loop_stop()
```

Figura C.2: Classe do pacote orquestrador responsável por manipular e analisar dados conforme solicitado por objetos inteligentes.

```

import json
import sqlite3
import datetime
from datetime import date, timedelta
import time

# SQLite DB Name
DB_Name = "C:\\\\..."      #CAMINHO DO BANCO DE DADOS DO SISTEMA WEB
=====
# Database Manager Class

class DatabaseManager():
    def __init__(self):
        self.conn = sqlite3.connect(DB_Name)
        self.conn.execute('pragma foreign_keys = on')
        self.conn.commit()
        self.cur = self.conn.cursor()

    def add_del_update_db_record(self, sql_query, args=()):
        self.cur.execute(sql_query, args)
        self.conn.commit()
        return

    def select_record(self,sql_query, args=()):
        self.cur.execute(sql_query,args)
        registro = self.cur.fetchone()[0]
        return registro

    def select_agendamento(self,sql_query,args=()):
        self.cur.execute(sql_query,args)
        usuario,email,sala,dtagendamento,hrentrada,hrsaida =
            self.cur.fetchone()
        return usuario,email,sala,dtagendamento,hrentrada,hrsaida

    def update_tag_user(self,sql_query, args=()):
        self.cur.execute(sql_query,args)
        self.conn.commit()
        if(self.cur.rowcount==1):
            status = True
        else:
            status = False
        return status

    def __del__(self):
        self.cur.close()
        self.conn.close()

```

```

#=====
#      RECEBE ID DA TAG - BUSCA AGENDAMENTO EM SALA/DATA/HORA NO BD
def validar_agendamento(Topic, jsonData):
    dbObj = DatabaseManager()

    #Separacao de infos contidas em topico
    infoTopico = Topic.split('/')

    #Configura a mensagem json recebida
    mensagemPython = json.loads(jsonData)

    #Variaveis para consulta
    sala = infoTopico[1]
    data_ag = date.today()
    tag = mensagemPython['Tag']
    hrchegada = datetime.datetime.now().strftime("%H:%M:00")

    admin_tag=dbObj.select_record("select tag from auth_user
                                    where is_admin == 'T'")

    if(admin_tag == tag):
        #intervalo de acesso para administrador (hora_atual
        #+ 10 minutos)
        horaadm = (datetime.datetime.now() + datetime.
                    timedelta(minutes=10)).strftime("%H:%M:00")
        usuario = 'Admin'
        hrsaida = horaadm

    else:
        #chama a funcao de busca do bd
        try:
            usuario,email,sala,dtagendamento,hrentrada,hrsaida
            = dbObj.select_agendamento("select
                u.first_name, a.funcionario, a.sala,
                a.dtagendamento, a.hrentrada, a.hrsaida
                from agendamentos as a join auth_user as u
                on a.funcionario = u.email where
                a.sala = (?) and a.dtagendamento = (?)
                and u.tag = (?) and ((?) between
                a.hrentrada and a.hrsaida)",[sala,data_ag,
                tag,hrchegada])
            try:
                tolerancia=(datetime.datetime.strptime
                            (hrentrada,'%H:%M:%S') + datetime
                            .timedelta(minutes=15))
                            .strftime("%H:%M:00")

                if(hrchegada > tolerancia):
                    pontualidade = 'F'
                else:
                    pontualidade = 'T'

            except:
                print("ERRO NA TOLERANCIA")
        except:
            print("ERRO NA BUSCA DO BD")
    else:
        print("ERRO NA VALIDACAO DE USUARIO")

```

```

        print(usuario, email, sala, dtagendamento,
               hrentrada, hrsaida, hrchegada,
               pontualidade)

    dbObj.add_del_update_db_record("update
                                    pontualidade set comparecimento = 'T',
                                    hrchegada = (?), pontualidade = (?)
                                    where pfuncionario = (?) and
                                    psala = (?) and pdtagendamento = (?)
                                    and phrentrada = (?)",
                                    [hrchegada, pontualidade, email, sala,
                                    dtagendamento, hrentrada])

    except TypeError:
        print("Erro ao inserir dados de
              pontualidade.")

    except TypeError:
        print("Nenhum agendamento encontrado
              ---- Validar Agendamento DAL")
        return None, None

del dbObj
return usuario, hrsaida

=====
#      ASSOCIAR TAG USUARIO
def associar_tag(Topic, jsonData):
    dbObj = DatabaseManager()
    #configura a mensagem json recebida
    mensagemPython = json.loads(jsonData)
    tag = mensagemPython['Tag']
    email = mensagemPython['Email']

    number = dbObj.select_record("select count(*) from auth_user where
                                  tag = (?)", [tag])

    if(number >= 1):
        print("Tag ja associada.")
        status=False
    else:
        #chama a funcao de update do bd
        try:
            status = dbObj.update_tag_user("update auth_user
                                            set tag = (?) where Email = (?)", [tag, email])
        except TypeError:
            print("ERROR - Associar Tag DAL")
            status = False

del dbObj
return status

```

Figura C.3: Classe do pacote orquestrador responsável por publicar mensagens a tópicos de entrada de objetos inteligentes para posterior exibição no visor LCD.

```

import paho.mqtt.client as mqtt
import random, threading, json, string
from datetime import datetime

#=====
# MQTT Settings
MQTT_Broker = "m12.cloudmqtt.com"
MQTT_Port = 19729
Keep_Alive_Interval = 45
user = "qxqwnxyw"
password = "ktJEqMyhPOVx"

#=====

def on_connect(client, userdata, rc):
    if rc != 0:
        pass
        print ("Unable to connect to MQTT Broker...")
    else:
        print ("Connected with MQTT Broker: " + str(MQTT_Broker))

def on_publish(client, userdata, mid):
    pass

def on_disconnect(client, userdata, rc):
    if rc !=0:
        pass

mqttc = mqtt.Client("publisher-pc")
mqttc.username_pw_set(user, password=password)
mqttc.on_connect = on_connect
mqttc.on_disconnect = on_disconnect
mqttc.on_publish = on_publish
mqttc.connect(MQTT_Broker, int(MQTT_Port), int(Keep_Alive_Interval))

#=====
#      FUNCAO QUE EXECUTA A PUBLICACAO DE MENSAGENS NO BROKER
def publish_to_topic(topico, mensagem):
    mqttc.publish(topico,mensagem,1)
    print ("Published: " + str(mensagem) + " " + "on MQTT Topic: "
          + str(topico))
    print ("")

#=====
# RECEBE O NOME DO PROFISSIONAL ENCONTRADO NO BANCO DE DADOS E PUBLICA
# NO TOPICO "topico/resposta"

def publish_nome_profissional(sala,mensagem):
    topico = "topicoEntrada/" + sala + "/confirmarAgendamento"
    publish_to_topic(topico,mensagem)

```



## Apêndice D

# Código-Fonte do Software Embarcado do Objeto Inteligente

Figura D.1: Código-Fonte do Software Embarcado

```
// --- JSON ---
#include <ArduinoJson.h>

// --- RELE ---
#define controle D0

// --- LCD ---
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2); // Cria instancia de LiquidCrystal_I2C

// --- WIFI ---
#include<ESP8266WiFi.h>
const char* ssid = "SUA_REDE";
const char* password = "SUA_SENHA";

WiFiClient nodeMCUClient; // Cria instancia de WiFiClient

// --- MQTT ---
#include <PubSubClient.h>
const char* mqttBroker = "m12.cloudmqtt.com";
unsigned int port = 19729;
const char* mqttClientID = "ESP8266Client";
const char* mqtt_user = "qxqwnxyw";
const char* mqtt_password = "ktJEqMyhPOVx";

PubSubClient client(nodeMCUClient); // Cria instancia de PubSubClient

// --- RFID ---
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN D4
```

```

#define RST_PIN D3

MFRC522 mfrc522(SS_PIN, RST_PIN); // Cria instancia de MFRC522

// --- DEFINICAO DE TOPICOS DE CONTROLE ---

// Topico para recebimento de msg do servidor mqtt (subscriber)
const char* topicoEntrada = "topicoEntrada/A21/#";

// Aplicacao - Prototipo (abre conexao para associar tag user)
const char* associarTagUser = "topicoEntrada/A21/associarTagUser";

// Aplicacao - Prototipo (abre conexao para associar tag master)
const char* associarTagMaster = "topicoEntrada/A21/associarTagMaster";

// Aplicacao - Prototipo (confirma agendamento para abrir sala)
const char* confirmarAgendamento = "topicoEntrada/A21/confirmarAgendamento";

// Responde a solicitacao com o codigo da tag a ser associada
const char* salvarTagUser = "topicoSaida/A21/salvarTagUser";

// Responde a solicitacao com o codigo da tag a ser associada
const char* salvarTagMaster = "topicoSaida/A21/salvarTagMaster";

// Solicita validacao de agendamento de Tag lida
const char* validarAgendamento = "topicoSaida/A21/validarAgendamento";

/* flag para cadastro de Tag User/Master */
int get_card = 0;

// --- FUNCAO DE PREPARACAO DO PROTOTIPO ---
void setup() {
    WiFi.mode(WIFI_STA);

    /* RELE - inicializa pinos de saida */
    pinMode(controle, OUTPUT);
    digitalWrite(controle, HIGH);

    /* LCD - Inicializa o display LCD */
    lcd.init();
    lcd.backlight();
    lcdpattern();

    /* MQTT - Define parametros para conexao ao MQTT Broker */
    Serial.begin(115200);
    WiFiConnect();
    client.setServer(mqttBroker, port);
    client.setCallback(callback);
    reconnect();

    /* RFID - Inicializa modulo RFID */
    SPI.begin();
    mfrc522.PCD_Init();
}

```

```
// --- FUNCAO DE MSG PADRAO DO DISPLAY LCD ---
void lcdpattern(){
    lcd.setCursor(0,0);
    lcd.print("*** CONTROLE DE ");
    lcd.setCursor(0,1);
    lcd.print("ACESSO - LSI ***");
}

// --- FUNCAO DE CONEXAO A REDE WIFI ---
void WiFiConnect(){
    delay(10);
    Serial.println();
    Serial.print("Conectando-se a rede: ");
    Serial.println(ssid);
    WiFi.begin(ssid,password);

    while(WiFi.status() != WL_CONNECTED){
        delay(500);
        Serial.println(".");
    }

    Serial.print("Conectado a rede: ");
    Serial.println(ssid);
    Serial.println();
}

// --- FUNCAO DE RECONEXAO AO MQTT BROKER ---
void reconnect(){

    while (!client.connected()) {
        Serial.print("Tentando conectar-se ao MQTT Broker... ");
        if (client.connect(mqttClientID, mqtt_user, mqtt_password)) {
            Serial.println("Conectado ao MQTT Broker");

            // Funcao para inscricao em um topico
            client.subscribe(topicoEntrada,1);

        } else {
            Serial.print("Falha na conexao, rc= ");
            Serial.print(client.state());
            Serial.println("Nova tentativa em 5s.");
            delay(5000);
        }
    }
    Serial.println();
}
```

```

// --- FUNCAO LEITURA DE TAGS E PUBLICACAO DE INFORMACOES ---
void ReadTagAndPublishInfo(const char* topicoSaida, const char* msgInfo){
    // Busca novas Tags
    if ( ! mfrc522.PICC_IsNewCardPresent() )
    {
        return;
    }
    // Seleciona uma das Tags lidas
    if ( ! mfrc522.PICC_ReadCardSerial() )
    {
        return;
    }

    // Define string para UID da Tag
    Serial.println();
    Serial.println();
    Serial.print("Tag lida:");
    String tagInfo= "";
    for (byte i = 0; i < mfrc522.uid.size; i++)
    {
        Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(mfrc522.uid.uidByte[i], HEX);
        tagInfo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        tagInfo.concat(String(mfrc522.uid.uidByte[i], HEX));
    }
    tagInfo.toUpperCase();

    Serial.println();
    Serial.println();

    //print no display
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(tagInfo.substring(1));

    // Define objeto Json para envio
    DynamicJsonBuffer jBuffer;
    JsonObject& root = jBuffer.createObject();
    root["Tag"] = tagInfo.substring(1);

    if(topicoSaida == salvarTagUser){
        root["Email"] = msgInfo;
    }

    if(topicoSaida == salvarTagMaster){
        root["Email"] = msgInfo;
    }
}

```

```

char JSONmessageBuffer[100];
root.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));
Serial.print("Mensagem enviada [");
Serial.print(topicoSaida);
Serial.print("]", " ");
Serial.println(JSONmessageBuffer);
Serial.println();

// Funcao para envio de informacao
if (client.publish(topicoSaida, JSONmessageBuffer) == true) {
    Serial.println("Mensagem enviada com sucesso.");
    lcd.setCursor(0,1);
    lcd.print("Mensagem enviada");
    get_card = 1;
} else {
    Serial.println("Erro no envio da mensagem.");
    lcd.setCursor(0,1);
    lcd.print("Erro no envio");
    get_card = 1;
}

Serial.println();
Serial.println("-----");
}

// --- FUNCAO PARA TRATAMENTO DE MENSAGENS RECEBIDAS DO MQTT BROKER ---
void callback(char* topic, byte* payload, unsigned int length) {

    lcd.clear();
    Serial.print("Mensagem recebida [");
    Serial.print(topic);
    Serial.print("]", " ");

    /*Funcao associar Tag de Usuarios*/
    if(String(topic) == associarTagUser){
        String message;
        for (int i = 0; i < length; i++) {
            char c = (char)payload[i];
            message += c;
        }
        Serial.print(String(message));

        // Deserializa objeto Json
        DynamicJsonBuffer jBuffer;
        JsonObject& msg_User = jBuffer.parseObject(String(message));

        lcd.setCursor(0,1);
        lcd.print("Aguardando Tag..");
    }
}

```

```

get_card = 0;

while(get_card < 1){
    ReadTagAndPublishInfo(salvarTagUser, msg_User["Email"].asString());
    delay(1);
}
}

/*Funcao associar Tag Master*/
if(String(topic) == associarTagMaster){
    String message;
    for (int i = 0; i < length; i++) {
        char c = (char)payload[i];
        message += c;
    }
    Serial.print(String(message));

    // Deserializa objeto Json
    DynamicJsonBuffer jBuffer;
    JsonObject& msg_Master = jBuffer.parseObject(String(message));

    lcd.setCursor(0,1);
    lcd.print("Aguardando Tag..");

    get_card = 0;

    while(get_card < 1){
        ReadTagAndPublishInfo(salvarTagMaster, msg_Master["Email"].asString());
        delay(1);
    }
}

/*Funcao de confirmacao de agendamento*/
if(String(topic) == confirmarAgendamento){
    String message;
    for (int i = 0; i < length; i++) {
        char c = (char)payload[i];
        message += c;
    }
    Serial.print(String(message));

    // Deserializa objeto Json
    DynamicJsonBuffer jBuffer;
    JsonObject& msg_Validacao = jBuffer.parseObject(String(message));

    //Variaveis extraidas da mensagem json
    String funcionario = msg_Validacao["funcionario"].asString();
    String delta = msg_Validacao["deltaT"].asString();
}

```

```
if(funcionario != ""){  
    digitalWrite(controle, LOW);  
    lcd.setCursor(0,0);  
    lcd.print(funcionario);  
    lcd.setCursor(0,1);  
    lcd.print("Acesso Concedido");  
    delay(delta.toInt());  
    digitalWrite(controle, HIGH);  
}else{  
    lcd.setCursor(0,0);  
    lcd.print("Tag Invalida");  
    lcd.setCursor(0,1);  
    lcd.print("Acesso Negado");  
    delay(2000);  
}  
  
}  
  
Serial.println("-----");  
}  
  
// --- FUNCAO PRINCIPAL ---  
void loop(){  
    /*MQTT*/  
    if (!client.connected()) {  
        reconnect();  
    }  
  
    client.loop();  
  
    ReadTagAndPublishInfo(validarAgendamento, NULL);  
  
    // Funcao para inscricao em um topico  
    client.subscribe(topicoEntrada);  
    delay(2000);  
  
    lcdpattern();  
}
```