Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Breno Machado Vasconcelos Alves

Trabalho de Conclusão de Curso

Desenvolvimento de um Sistema Distribuído de Controle de Acesso por RFID com Arquitetura Sem Servidor Baseada em Computação em Nuvem

Campina Grande - PB

Fevereiro de 2019

Breno Machado Vasconcelos Alves

Trabalho de Conclusão de Curso

Desenvolvimento de um Sistema Distribuído de Controle de Acesso por RFID com Arquitetura Sem Servidor Baseada em Computação em Nuvem

Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Controle e Automação.

Orientador: Rafael Bezerra Correia Lima

Campina Grande - PB

Fevereiro de 2019

Breno Machado Vasconcelos Alves

Trabalho de Conclusão de Curso

Desenvolvimento de um Sistema Distribuído de Controle de Acesso por RFID com Arquitetura Sem Servidor Baseada em Computação em Nuvem

Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Trabalho aprovado em: Campina Grande - PB, / /

Rafael Bezerra Correia Lima,
UFCG
Professor Orientador

Campina Grande - PB

Fevereiro de 2019

Dedico este trabalho aos meus pais e à minha querida irmã.

Agradecimentos

Agradeço primeiramente aos meus pais, José Francisco e Leila, e à minha irmã, Leilane, por todo o apoio dado. Com muita fé em Deus, amor e esforço possibilitaram que nada, nunca, me faltasse. Este trabalho só existe por causa de deles.

Agradeço aos amigos e colegas que estiveram presentes nos mais diversos momentos da minha graduação: Abdias, Ana Carla, Arthur, Bahia, Barbara, Camila, Carolina, Charles, Dimas, Felipe, Harley, Indio, Isaque, Lucão, Leo, Mari, Martins, Mateus, Mateusinho, Mestre Telmo, Mona, Niago, Pablo, Pedro(s), Rubens, Soneca, Thiago, Tulio, Vinicim, Walan, Yago, Yan, Yuri, Zé Wesley e tantos outros. Foram muitas noites viradas (estudando ou farrando), trabalhos, cafés, almoços, rolês, conversas e experiências.

Agradeço aos professores Pericles Rezende de Barros, George Acioli Júnior e Rafael Bezerra Correia Lima por terem me dado a oportunidade de trabalhar com eles, pelos vários ensinamentos adquiridos e pelo apoio ao longo dessa jornada.

Por fim, aos funcionários do DEE, em especial Adail e Tchai, que sempre estiveram acessíveis, disponíveis e com uma imensa vontade em ajudar, por menor que tenha sido o problema.

“Não se é grande sem crescer
Não se cresce sem sentir
Nada existe sem porquê, portanto
vai correndo procurar
tudo aquilo que almejou
já sabendo que ao voltar
o mundo será outro.”

Móvies Coloniais de Acajú

Resumo

Em diversos ambientes o controle do acesso de pessoas é um fator crítico para a manutenção da segurança. Os sistemas de controle de acesso são tecnologias que automatizam os serviços de autenticação de usuários, garantindo confiabilidade e fornecendo dados que favorecem uma gestão mais assertiva e simplificada do negócio. A globalização sugere que esse tipo de gestão possa ser feita de forma remota. O conceito de Internet das Coisas possibilita conectar dispositivos à internet para transmitir grandes quantidades de dados e facilitar o seu acesso. Este conceito está tornando os sistemas distribuídos cada vez mais presentes na vida das pessoas. Outro paradigma que vem influenciando no cotidiano das pessoas é a Computação em Nuvem. Esses sistemas fornecem serviços que facilitam o processamento e armazenamento dos mais diversos tipos de dados. Uma forma otimizada de utilizar esse conceito e aumentar a sua eficiência é construindo aplicações conhecidas como Sem Servidor. Neste trabalho são apresentados o projeto e desenvolvimento de um sistema distribuído de controle de acesso com tecnologia RFID. O sistema apresenta uma arquitetura sem servidor e une os conceitos de Internet das Coisas e Computação em Nuvem, demonstrando a capacidade dessa junção de se tornar um padrão no mercado de aplicativos globais.

Palavras-chave: Controle de Acesso, Arquitetura Sem Servidor, Internet da Coisas, Computação em Nuvem, Sistemas Distribuídos.

Abstract

People's access control is a critical factor in maintaining security in many environments. Access control systems are technologies that automate user authentication services, ensure reliability, and provide data that favor more assertive and simplified management of the business. The Internet of Things enables you to connect devices to the internet to transmit large amounts of data and make it easier to access. This concept is making distributed systems more and more present in people's lives. Another paradigm that has been influencing people's daily lives is Cloud Computing. These systems provide services that facilitate the processing and storage of the most diverse types of data. An optimized way to use this concept and increase its efficiency is by building applications known as Serverless. This paper presents the design and development of a distributed access control system with RFID technology. The system features a serverless architecture and brings together Internet of Things and Cloud Computing concepts, demonstrating the ability of this union to become a standard in the global application market.

Keywords: Access Control, Serverless Architecture Internet of Things, Cloud Computing, Distributed computing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
AWS	Azure Web Services
CoAP	Constrained Application Protocol
CRUD	Create - Read - Update - Delete
CSS	Cascading Style Sheets
DB	Database
FAAS	Function As A Service
GPIO	General Purpose Input/Output
HVAC	Heating - Ventilation, Air Conditioning
HF	High Frequency
HTTP	HyperText Transfer Protocol
HTML	Hypertext Markup Language
IAAS	Infrastructure As A Service
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	Javascript Object Notation
LF	Low Frequency
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
OPC UA	Open Platform Communications Unified Architecture
PAAS	Platform As A Service
POCO	Plain OLD CLR Object

REST	Representational State Transfer
RFID	Radio-Frequency IDentification
SAAS	Software As A Service
SPI	Serial Peripheral Interface (SPI)
SQL	Structured Query Language
SDK	Software development kit
SLA	Service Level Agreement
TI	Tecnologia da Informação
UHF	Ultra Hihg Frequency
URL	Uniform Resource Locator

Sumário

1 INTRODUÇÃO

Os sistemas de automação de edifícios possuem em seu escopo diversos subsistemas integrados, tais como controle das condições de ambientes internos como sistemas de aquecimento, ventilação e ar-condicionado (HVAC), iluminação, segurança contra incêndios, controle de acesso e etc. (KASTNER et al., 2005). Estima-se que as primeiras casas conectadas surgiram em meados de 1960, entretanto a complexidade técnica e o alto custo de implementação fizeram com que o conceito não se tornasse uma tendência na época (HARPER, 2003).

Um novo paradigma que está rapidamente ganhando terreno no cenário das modernas comunicações sem fio é a Internet das Coisas (IoT). A ideia básica deste conceito é a presença generalizada de uma variedade de coisas ou objetos - tais como etiquetas RFID (Radio Frequency IDentification), sensores, atuadores, telefones celulares, etc. - que, através de esquemas de endereçamento únicos, são capazes para interagir uns com os outros e cooperar com seus vizinhos para alcançar objetivos comuns (GIUSTO et al., 2010).

Com o avanço das tecnologias de comunicação e sistemas embarcados, dispositivos conectados passaram a se tornar cada vez mais presentes no cotidiano das pessoas. Conforme estimativas do relatório da IDC (2013), haverá 30 bilhões de dispositivos conectados à Internet e com sensores habilitados até 2020. Esse rápido desenvolvimento da internet das coisas, das tecnologias sem fio e o crescimento do número de dispositivos conectados tornam a automação residencial em todas as casas uma possibilidade real (BHIDE, 2014).

A IoT é geralmente caracterizada por coisas pequenas do mundo real, amplamente distribuídas, com armazenamento e capacidade de processamento limitados, que envolvem preocupações relacionadas à confiabilidade, desempenho, segurança e privacidade. Por outro lado, a computação em nuvem tem recursos praticamente ilimitados em termos de capacidade de armazenamento e processamento. Dessa forma, a integração da Internet das Coisas com a computação em nuvem se apresenta como um paradigma promissor, chamado CloudIoT (BOTTA et al, 2016).

Stojkoska e Trivodaliev (2017) apresentaram uma abordagem para a utilização da Internet das Coisas para casas inteligentes, nela todos os dados de diferentes fontes são acumulados na nuvem e essa deve fornecer armazenamento e processamento dos dados. Dessa forma, o projeto e desenvolvimento de um sistema integrado utilizando CloudIoT para automação residencial se mostra como uma aplicação interessante que tende a se tornar cada vez mais popular.

Dentre os serviços relacionados a automação residencial, os sistemas de controle de acesso apresentam-se como fator importante na manutenção da segurança de determinado

local. Estes sistemas são projetados para controlar quem tem acesso a uma rede ou lugar específico e podem ser integrados a meios eletrônicos, viabilizando a comunicação desses sistemas com a internet e, consequentemente, a troca de informações. Tal conexão permite que o ambiente possa ser controlado e monitorado remotamente, além de facilitar a análise dos dados coletados pelo sistema.

1.1 Objetivos

Nesta seção são descritos os objetivos do trabalho com o intuito de esclarecer as motivações utilizadas como base na busca pelos resultados.

Este Trabalho de Conclusão de Curso tem como objetivo utilizar os conhecimentos aprendidos no ambiente acadêmico para projetar e desenvolver um sistema de controle de acesso para o Laboratório de Instrumentação Eletrônica e Controle (LIEC), localizado na UFCG, integrando os conceitos de Internet das Coisas e Computação em Nuvem. Abaixo estão listados os objetivos específicos do sistema.

- Pesquisar e analisar as principais tecnologias presentes no mercado que podem ser utilizadas na aplicação.
- Validar a integração dos conceitos de Internet das Coisas e Computação em Nuvem.
- Realizar o projeto e o desenvolvimento da aplicação utilizando padrões e práticas que exaltem as qualidades do sistema.

1.2 Organização do Trabalho

Além deste capítulo, o trabalho está organizado da seguinte forma:

- Capítulo 2: Fundamentação Teórica. Neste capítulo são apresentados os conceitos relevantes para o entendimento do trabalho.
- Capítulo 3: Materiais e Métodos. Neste capítulo são apresentadas as tecnologias utilizadas e a metodologia adotada no projeto.
- Capítulo 4: Modelagem: Aqui é descrito detalhadamente o de projeto de modelagem da aplicação.
- Capítulo 5: Resultados: Nesse capítulo são apresentados e discutidos os resultados do trabalho.
- Capítulo 6: Conclusão: Aqui são apresentadas as conclusões do trabalho.

Ao fim do documento são apresentados as referências e os anexos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos e fundamentos envolvidos no sistema proposto, no intuito de prover maior embasamento teórico ao que será explanado posteriormente. São aqui descritos os conceitos de Internet das Coisas, Computação em Nuvem, e a utilização do método RFID para sistemas de controle de acesso.

2.1 Internet das Coisas

Considerando o impacto que a internet tem na vida das pessoas, tornar-se razoável afirmar que a Internet é uma das mais importantes e poderosas criações em toda a história da humanidade. A internet das coisas (IoT) pode ser entendida como uma evolução da Internet capaz de absorver, processar e distribuir dados de diversos tipos de fontes.

2.1.1 O Conceito de IoT

Ainda não existe uma definição oficial para a internet das coisas. O conceito foi primeiramente proposto por Kevin Ashton em 1999, em suas palavras:

> Se tivéssemos computadores que soubessem tudo sobre as coisas em geral – usando dados que coletassem sem a nossa ajuda – seríamos capazes de rastrear e contar tudo, e reduzir bastante o desperdício, a perda e os custos. Nós saberíamos quando é necessário substituir, reparar ou fazer um recall de um produto, e se estão novos ou ultrapassados. Precisamos capacitar os computadores com seus próprios meios de coletar informações, para que possam ver, ouvir e cheirar o mundo sozinhos, com toda a sua glória aleatória. O RFID e a tecnologia de sensores capacitam os computadores a observar, identificar e entender o mundo sem as limitações dos dados inseridos pelos humanos.(ASHTON, 1999 apud Lopez Research LLC, 2013, 2)

Entretanto nos dias atuais o termo não se resume a um conjunto de dispositivos interoperáveis que podem ser identificados por tecnologia RFID. Uma definição mais atual seria que é uma infraestrutura de rede global dinâmica com recursos de autoconfiguração baseados em padrões e protocolos de comunicação interoperáveis; as 'coisas' físicas e virtuais em uma IoT têm identidades e atributos e são capazes de usar interfaces inteligentes e serem integradas como uma rede de informação" (Li et al. 2012a).

2.1.2 A IoT hoje

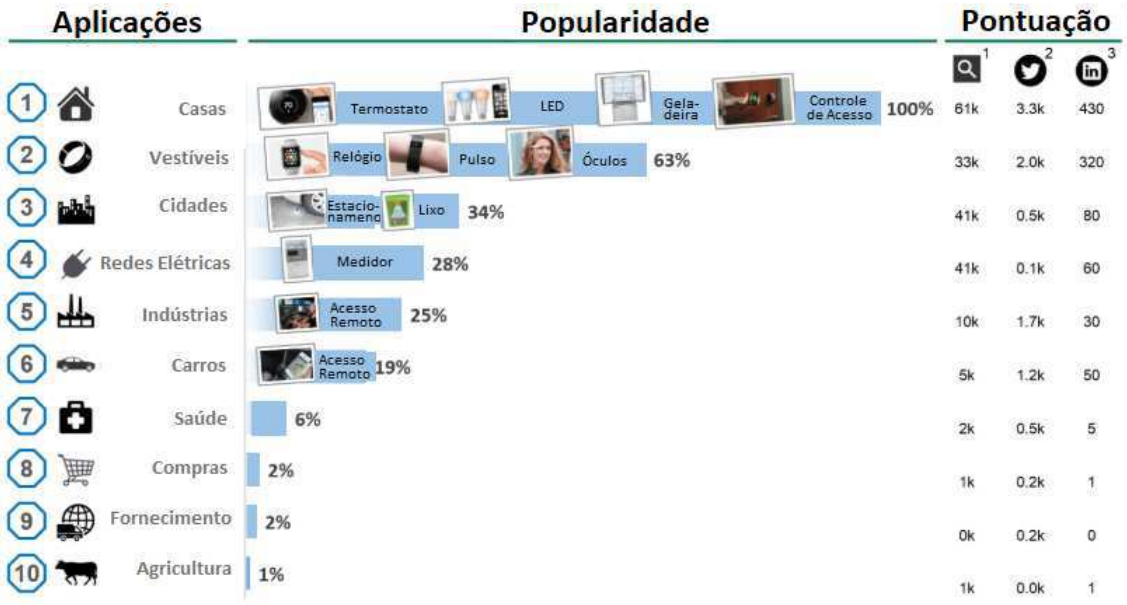
Em 2003, havia aproximadamente 6,3 bilhões de pessoas vivendo no planeta e 500 milhões de dispositivos conectados à Internet (Serviço de censo dos EUA, 2010;) Ao dividir o número de dispositivos conectados pelo população mundial, nota-se que havia menos de um dispositivo (0,08) para cada pessoa. O explosivo crescimento de smartphones e tablets elevou o número de dispositivos conectados à Internet para 12,5 bilhões em 2010, enquanto a população humana do mundo aumentou para 6,8 Bilhões, tornando o número de dispositivos conectados por pessoa igual a 1,84 e estima-se que até 2020 esse número seja mais de 3 vezes maior. (Cisco IBSG, 2010).

2.1.3 Aplicações para IoT

Quando cruzou-se o limiar de conectar mais objetos do que pessoas à Internet, abriu-se uma enorme janela de oportunidades para a criação de aplicações nas áreas de automação, sensoriamento e comunicação máquina-a-máquina(M2M). Na verdade, as possibilidades são quase sem fim.

A Figura 1 ilustra um ranking das aplicações mais populares para a internet das coisas. A pontuação foi medida utilizando o número de pesquisas de cada aplicação no Google, Twitter e Linkedin. Nota-se que a quantidade de pesquisas por casas inteligentes é muit superior Às outras aplicações mostrando o quanto é um apelo que interfere substancialmente na vida das pessoas.

Figura 1 – Ranking de popularidade das aplicações para IoT baseado no número de pesquisas feitas no Google, Twitter e Linkedin em Maio de 2014.



Fonte: https://iot-analytics.com/10-internet-of-things-applications/.

2.2 Computação em Nuvem

A proposta básica da computação em nuvem é que a provisão de recursos computacionais seja de responsabilidade de empresas especializadas ou que seja abstraído o fornecimento dos mesmos em níveis que apenas especialistas venham se preocupar em gerenciá-los e mantê-los, e ainda os mesmo sejam disponibilizados como serviços (Carr, 2008).

2.2.1 Conceito de Computação em Nuvem

Assim como IoT, não existe uma definição oficial para o conceito de Computação em Nuvem. Segundo Buyya (2008), uma nuvem é um tipo de sistema paralelo e distribuído que consiste de uma coleção de computadores virtualizados e interconectados que são provisionados de forma dinâmica e apresentados como um ou mais recursos computacionais unificados. Estes recursos são disponibilizados e controlados através de acordos relacionados aos serviços que são estabelecidos entre um prestador e um consumidor sendo definidos a partir de negociações entre as partes. Existem basicamente dois tipos plataforma de nuvem, as públicas e as privadas (BORTOLI, 2016). Alguns autores consideram a existência de um terceiro chamado nuvem híbrida, que une conceitos de ambas as públicas e privadas.

- Nuvem Pública

  A nuvem pública é definida pelas suas operações que são realizadas por prestadores de serviços. Os clientes se beneficiam de economias de escala, pois os custos de infraestrutura são divididos entre os usuários. Cada cliente paga somente pelos recursos e demandas de capacidades atendidas tornando a alternativa mais barata para manter toda a estrutura em um servidor próprio.

  Todos os clientes em uma nuvem pública compartilham dos mesmos recursos e espaços tendo as configurações de segurança e as variações de disponibilidade limitadas ao gerenciador do sistema, uma vez que são totalmente geridos pelo prestador do serviço. Azure e Amazon.com são exemplos de cloud pública.

- Nuvem Privada

  A cloud privada representa aquelas que são desenvolvidas para atender exclusivamente uma empresa de forma individual. Elas permitem que a empresa hospede aplicativos e infraestruturas completas na nuvem e dediquem maiores esforços relacionados à segurança dos acessos e controles de dados, e uma infraestrutura 100% dedicada para as demandas do negócio de um cliente.

2.2.2 Principais Características

- **Elasticidade Rápida**:

  Um recurso importante de um ambiente de nuvem é que ele fornece uma plataforma projetada para ser elástica. Assim os clientes provisionam os recursos contratados. Quando o usuário não precisa mais desse recurso e para de pagar, o recurso pode ser liberado de volta ao pool de recursos.

- **Serviços sob Demanda**

  Um serviço em nuvem deve fornecer uma maneira de medir e medir serviço. Consequentemente, um ambiente de nuvem inclui um serviço que rastreia quantos recursos um cliente usa. Em uma nuvem pública, os clientes são cobrados por unidades de recursos consumidos. Em uma nuvem privada, o gerenciamento de TI pode implementar um mecanismo de cobrança para os departamentos que aproveitam os serviços. (HURWITZ, KAUFMAN, HALPER, 2012).

- **Virtualização**:

  A virtualização é uma técnica de compartilhamento de recursos que se baseia no princípio da divisão de recursos físicos ou sistemas operacionais para medidas de controle de custos e utilização mais eficiente dos recursos (DIVAKARLA, KUMARI, 2010).

- **Independência de Localização e Repositório de Recursos**

  Os provedores de recursos computacionais são organizados para atender múltiplos usuários através de um modelo multiclientes. Para isto são utilizados diferentes recursos físicos e virtuais que podem ser atribuídos e configurados dinamicamente de acordo com a demanda de cada cliente. O usuário não conhece a localização física dos recursos computacionais, porém pode ser possível especificar sua prioridade de localização com relação a país e centro de dados através do SLA (BORGES,2011).

2.2.3 Modelos de Serviços

Existem diversos modelos de serviços de nuvem, entretanto O modelo conceitual encontrado com maior frequência na literatura é composto por três camadas. A Figura 2 ilustra esses tipos e exemplifica tipos de aplicações em que tais serviços podem ser utilizados.

- **IAAS - Infraestrutura como Serviço**

  Contém os componentes básicos da TI em nuvem e, geralmente, dá acesso aos recursos de computação e armazenamento baseados na Internet. Sendo a categoria mais

básica entre os tipos de computação em nuvem, a IaaS permite que você alugue uma infraestrutura de TI (servidores e máquinas virtuais, armazenamento, redes e sistemas operacionais) de um provedor de nuvem em uma base paga conforme o uso. (P.ex.: Google Drive, Amazon AWS) (MICROSOFT,21–).

- **PAAS - Plataforma como Serviço**

  É a camada intermediária do modelo conceitual, sendo composta por hardware virtual disponibilizado como serviço. Oferece tipos específicos de serviços como sistemas operacionais, banco de dados, serviços de mensagens, serviços de armazenamento de dados e etc. Uma PaaS fornece ambientes de desenvolvimento de software e facilita a implantação de aplicações sem os custos e complexidades relativos a compra e gerenciamento do hardware e de software adjacentes que são necessários ao ambiente de desenvolvimento (BORGES,2011).

- **SAAS - Software como Serviço**

  O software como um serviço oferece um produto completo, executado e gerenciado pelo provedor de serviços. Na maioria dos casos, as pessoas que se referem ao software como um serviço estão se referindo às aplicações de usuário final. Com uma oferta de SaaS, não é necessário pensar sobre como o serviço é mantido ou como a infraestrutura subjacente é gerenciada, você só precisa pensar em como usará este tipo específico de software. (P.ex.: Google Docs , Microsoft SharePoint Online) (AMAZON, 21–).

Figura 2 – Principais Modelos de Serviços de Nuvem.

Fonte: http://jornadaparanuvem.com.br/fundamentos-de-cloud-computing/os-tres-modelos-de-servico/(Adaptado).

Embora esses sejam os principais modelos, um outro modelo será bastante utilizado neste trabalho: um modelo serviços chamado FAAS - Função como Serviço. O FAAS é

uma chamada de procedimento remoto hospedada em serviço que aproveita a computação sem servidor para permitir a implementação de funções individuais na nuvem que são executadas em resposta a eventos (FOWLER,2016).

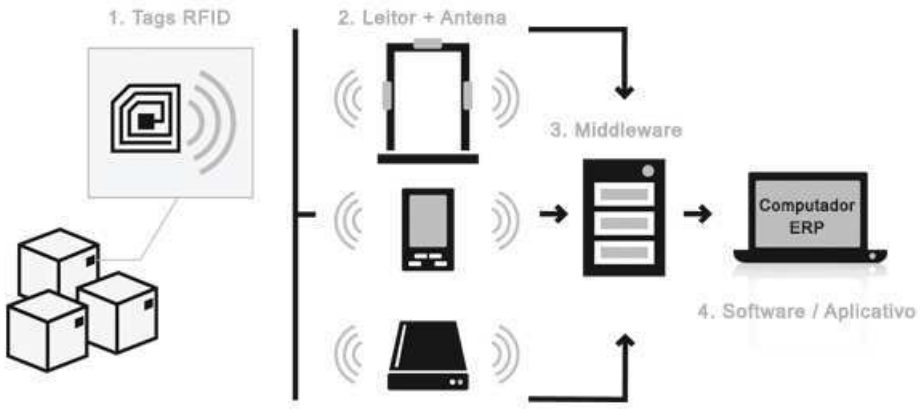
Este modelo vem sendo bastante utilizado pois é um dos fundamentos da computação sem-servidor. A computação sem servidor é um modelo de computação em nuvem que visa abstrair o gerenciamento do servidor e as decisões de infraestrutura de baixo nível. Nesse modelo, a alocação de recursos é gerenciada pelo provedor de nuvem em vez de ser pelo desenvolvedor do aplicativo.

2.3 RFID

A tecnologia de RFID é um método de identificação através de sinais de rádio. Tal tecnologia permite a captura automática de dados para identificação de objetos com dispositivos eletrônicos conhecidos como etiquetas eletrônicas, tags, ou cartões RF, que emitem sinais de radiofrequência para leitores que captam estas informações.

A Figura 3 apresenta um sistema RFID. Ele é composto, basicamente, de uma etiqueta (tag), que é composta de um chip semicondutor, uma antena e, algumas vezes, uma bateria; um leitor, composto por uma antena, um módulo RF e um módulo de controle; e um controlador, que na maioria das vezes é um computador, responsável pela parte de processamento da informação.

Figura 3 – Representação de um Sistema RFID



Fonte: http://www.afixgraf.com.br/como-funciona-rfid/.

As etiquetas RFID podem são divididas em dois tipos: passivos e ativos. Os transponders passivos utilizam a energia da onda de rádio frequência emitida pelo leitor para transmitir o seu sinal de resposta. Já os ativos contam com uma fonte de energia própria para transmitir seu sinal de resposta, aumentando o alcance.

Outra característica importante dos sistemas RFID é a frequência de operação. A frequência interfere na taxa de transferência de dados entre a etiqueta e o leitor e em seu alcance. As três frequências mais utilizadas para o sistema RFID passivo são: baixa frequência (LF), de 125kHz, alta frequência (HF), 13,56 MHz, e ultra alta frequência (UHF), operando na faixa de 860 a 960MHz. Em sistemas de controle de acesso geralmente são utilizadas tags RFID de alta frequência.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentadas algumas das principais tecnologias presentes no mercado que podem ser empregadas no desenvolvimento da aplicação. Durante esta abordagem serão enfatizadas as tecnologias utilizadas e as justificativas de suas escolhas. Posteriormente, será detalhada a metodologia usada no processo de desenvolvimento do sistema.

3.1 Materiais

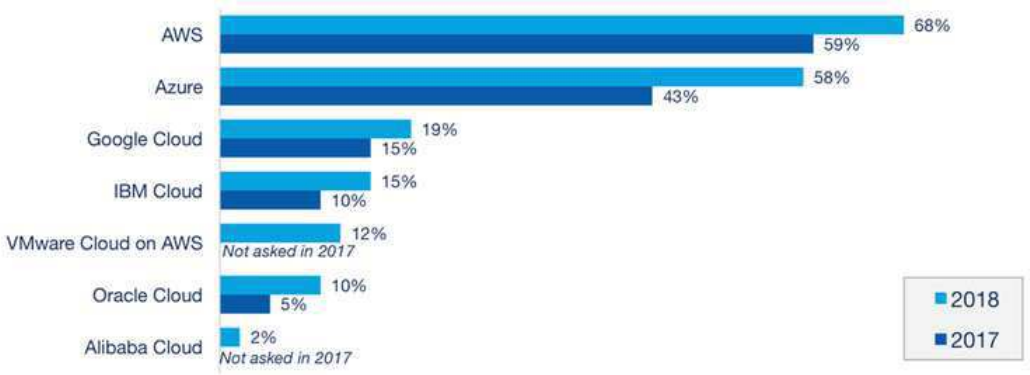
Nos dias atuais, a quantidade de produtos disponíveis capazes de solucionar um mesmo problema é muito extensa. Nesse sentido, é importante conhecer as principais tecnologias do mercado e identificar as que melhor se adequam a realidade do projeto em questão. Nesta seção serão detalhados os materiais utilizados neste trabalho.

3.1.1 Computação em Nuvem

O mercado de computação em nuvem está repleto de provedores de soluções que estão construindo produtos competitivos e inovadores. A escolha da plataforma utilizada neste projeto baseou-se nos seguintes critérios: adoção da plataforma no mercado e serviços disponíveis.

Na Figura 4 é apresentada uma comparação entre as principais plataformas de nuvem pública do mercado quanto a adoção dos seus serviços por empresas em 2017 e 2018. Observa-se que o serviço de nuvem da *Amazon, AWS,* lidera o *ranking*, porém a plataforma *Azure* da *Microsoft* teve um maior crescimento em 2018.

Figura 4 – Adoção de Empresas por Plataformas de Nuvem Púbicas: 2018 vs 2017.



Fonte: *RightScale 2018 State of the Cloud Report* , 2018.

A Tabela 1 apresenta uma comparação entre os serviços fornecidos pelas duas maiores plataformas de acordo com algumas funcionalidades necessárias no projeto proposto. Ambas as plataformas suportam as linguagens Node.js, Python, e C#. Quanto ao serviço de aplicações sem servidor, tanto o *AWS Lambda* quanto o *Azure Functions* oferecem gatilhos configuráveis e dinâmicos que podem ser usados para invocar suas funções em suas plataformas.

Tabela 1 – Alguns Serviços Fornecidos pelas plataformas *AWS* e *Azure*.

Serviços	***Amazon AWS***	***Microsoft Azure***
Aplicações sem servidor	*Lambda*	*Azure Functions*
Lógica de processo backend	-	*Azure Web Jobs*
Armazenamento de Objetos	*Simple Storage Services (S3)*	*Azure Storage*
Banco de Dados NO-SQL	*DynamoDB*	*Azure CosmosDB*
Internet das Coisas	*AWS IoT*	*Azure IoT Hub*

Fonte: Próprio autor.

Uma das principais diferenças entre as plataformas neste quesito é a disponibilidade do serviço. O *AWS Lambda* necessita invocar novas instâncias de uma função caso a mesma fique inativa por certo tempo, o que pode provocar um atraso na resposta. Enquanto isso, as funções do *Azure*, por terem como base o serviço *Web Jobs* não apresentam esse problema.

Outra diferença notável é que o banco de dados não-relacional CosmosDB da Microsoft permite a utilização de APIs já consolidadas no mercado, como SQL e MongoDB. Devido a isso e ao fato de um sistema de controle de acesso requisitar uma alta disponibilidade, a plataforma de nuvem escolhida foi a *Azure*.

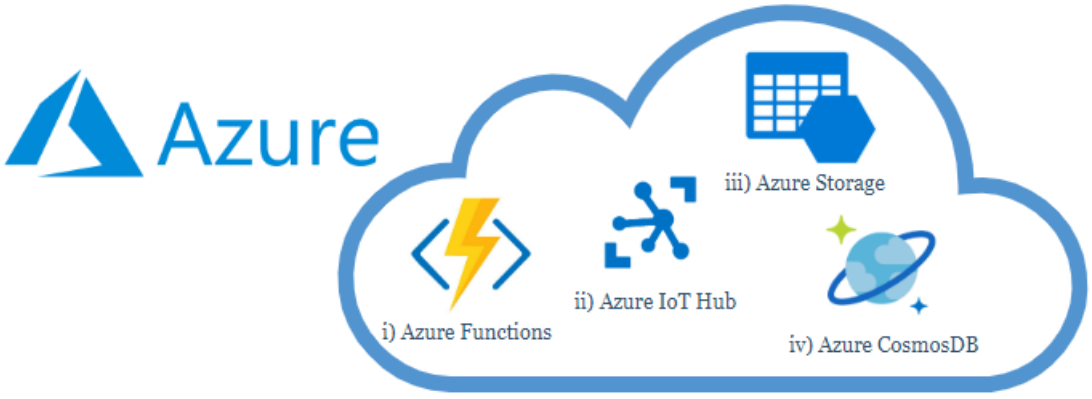
O desenvolvimento de aplicações para o Azure pode ser feito nos ambientes apresentados na Figura 5. O portal é a principal ferramenta para o gerenciamento de aplicações do Azure. O Visual Studio permite o desenvolvimento de aplicações mais complexas, além de possuir recursos de *debug* e testes automáticos. O VS Code por sua vez é um editor de texto que permite a criação de aplicações multiplataforma.

Figura 5 – Ambientes de Desenvolvimento para o Azure

Fonte: Próprio autor.

A Figura 6 apresenta os serviços de nuvem que foram utilizados no projeto. Cada serviço será detalhado a seguir.

Figura 6 – Serviços do *Azure* utilizados



Fonte: Próprio autor.

i *Azure Functions*

O *Functions* é um serviço de computação sem servidor que permite executar o código sob demanda sem precisar provisionar explicitamente ou gerenciar a infraestrutura (MICROSOFT,2017). Ele é uma ótima solução para processamento de dados, integração de sistemas, trabalhos com a IoT (Internet das coisas) e criação de APIs e micro serviços simples. Apresenta gatilhos que possibilitam sua utilização em diversas situações, como pode ser visto na Tabela 2.

ii *Azure IoT Hub*

O *Hub IoT* é um serviço gerenciado e hospedado na nuvem que atua como um hub central de mensagem para a comunicação bidirecional entre o aplicativo de IoT e os dispositivos que ele gerencia. Ele pode ser usado para criar soluções de IoT com comunicação segura e confiável entre milhões de dispositivos de IoT e um *backend* de solução de nuvem hospedado. (MICROSOFT,2018a). Dentre os recursos do *Hub IoT* pode-se destacar a sua compatibilidade com diversos protocolos e a segurança da comunicação.

O *Hub IoT* fornece um canal de comunicação seguro para o envio de dados pois apresenta uma autenticação por dispositivo, permitindo que cada dispositivo se conecte com segurança ao *Hub IoT* e possa ser gerenciado com segurança. Essa autenticação pode ser feita utilizando um token SAS ou utilizando certificados X.509 auto assinados ou autenticados por uma autoridade.

Para criar aplicativos que são executados nos dispositivos e interagem com o *Hub IoT* utilizam-se as bibliotecas de IoT do Azure. O SDK tem suporte a diversas

Tabela 2 – Gatilhos das Funções do Azure

Gatilhos	**Descrição**
HTTPTrigger	Dispara a execução do código usando uma solicitação HTTP.
TimerTrigger	Executa limpeza ou outras tarefas em lotes em uma programação predefinida.
CosmosDBTrigger	Processa documentos do Azure Cosmos DB quando eles são adicionados ou atualizados em coleções em um banco de dados NoSQL.
BlobTrigger	Processa blobs do Armazenamento do Azure quando são adicionados a contêineres.
QueueTrigger	Responde às mensagens que chegam em uma fila do Armazenamento do Azure.
EventGridTrigger	Responde a eventos entregues a uma assinatura na Grade de Eventos do Azure.
EventHubTrigger	Responde a eventos entregues a um Hub de Eventos do Azure.
ServiceBusQueueTrigger	Conecta o código a outros serviços do Azure ou serviços locais por meio da escuta de filas de mensagens.
ServiceBusTopicTrigger	Conecta o código a outros serviços do Azure ou serviços locais por meio da assinatura de tópicos.

Fonte: Próprio autor.

linguagens de programação e a alguns dos principais protocolos de comunicação para internet das coisas como HTTP, MQTT e AMQP. Além disso, é possível utilizar outros protocolos (como CoAP e OPC UA) criando um gateway personalizado para habilitar a adaptação de protocolo para o *Hub IoT*. (MICROSOFT,2018b). A Figura 7 ilustra as possíveis formas de conexão de dispositivos com o *Hub IoT*.

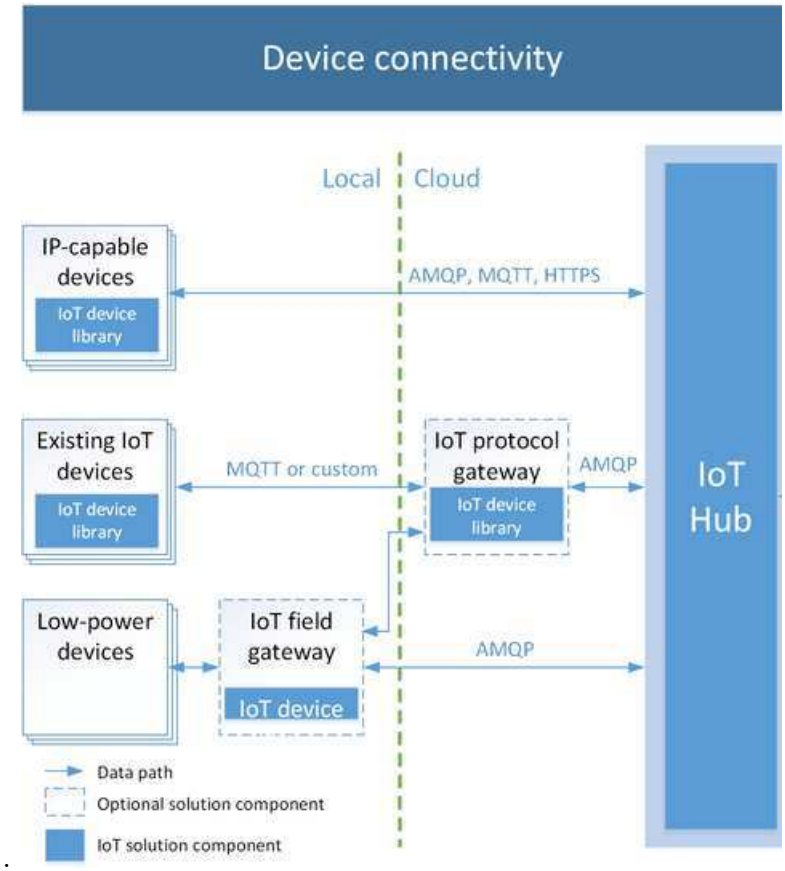
iii *Azure Storage*

O armazenamento do Azure é uma solução de armazenamento em nuvem da Microsoft para cenários de armazenamento de dados modernos. Ele oferece um armazenamento de objetos altamente escalável para objetos de dados, um serviço de sistema de arquivos para a nuvem, armazenamento de mensagens para um sistema de mensagens confiável e armazenamento de NoSQL (MICROSOFT,2019).

A Figura 8 ilustra os principais serviços de dados inclusos no armazenamento do Azure. Os recursos de fila tabela e blobs podem ser acessados a partir de aplicações REST, enquanto recursos como arquivos e blobs são projetados para as máquinas virtuais do Azure. (DENIS,2017)

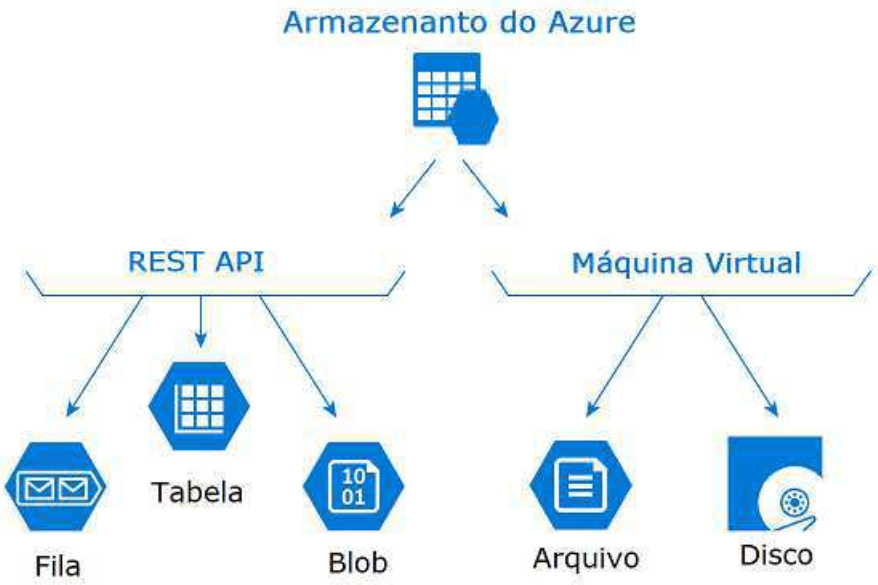
- **Filas**: O serviço Fila do Azure é usado para armazenar e recuperar mensagens. A fila de mensagens pode ser de até 64 KB de tamanho e uma fila pode conter milhões de mensagens. Filas são geralmente usadas para armazenar as listas de mensagens a serem processadas de forma assíncrona.

Figura 7 – Conexão de Dispositivos com o *Hub IoT*



Fonte: https://blogs.biztalk360.com/iot-hub-device-to-cloud/.

Figura 8 – Serviços de Dados do Armazenamento do Azure



Fonte: Próprio autor.

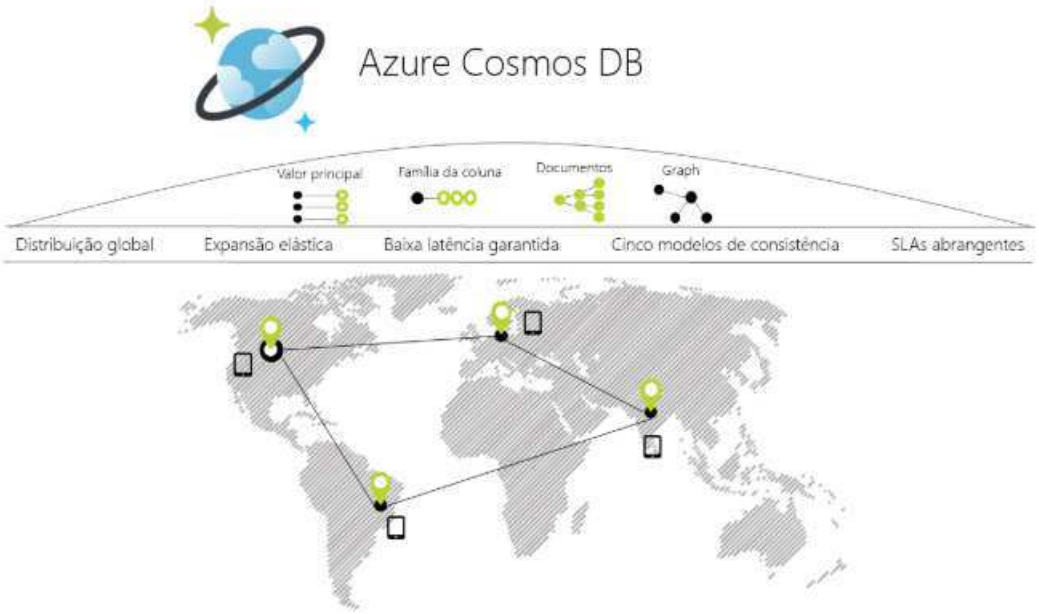
- **Tabelas**: O Armazenamento de Tabelas do Azure é um serviço que armazena dados NoSQL estruturados na nuvem, fornecendo um repositório chave/atributo com um design sem esquema.
- **Blobs**: O Armazenamento de Blobs do Azure é uma solução de armazenamento de objetos da Microsoft para a nuvem. O armazenamento de Blobs é otimizado para armazenar grandes quantidades de dados não estruturados, como texto ou dados binários.
- **Arquivos**: Os Arquivos do Azure permitem a configuração de compartilhamentos de arquivos por redes altamente disponíveis. Isso significa que várias VMs podem compartilhar os mesmos arquivos com acesso de leitura e gravação.
- **Discos**: O Armazenamento do Azure inclui recursos de disco gerenciados e usados por máquinas virtuais. Cada máquina virtual tem um disco de sistema operacional anexado. Ele é registrado como uma unidade SATA e rotulado como a unidade C: por padrão. Este disco tem uma capacidade máxima de 2048 gigabytes (GB).

iv *Azure CosmosDB*

O Azure Cosmos DB é o serviço de banco de dados multimodelo distribuído globalmente da Microsoft e suas principais características são ilustradas na Figura 9. Ele permite que você dimensione a taxa de transferência e o armazenamento de maneira elástica e independente em qualquer número de regiões geográficas do Azure. O acesso a dados no Cosmos DB é rápido, em um valor de milissegundos de dígito único, e permite utilizar a API de favoritos como o SQL, o MongoDB, o Cassandra, as Tabelas ou o Gremlin.

O Cosmos DB permite a criação de aplicativos altamente responsivos, escaláveis e disponíveis em todo o mundo. Ele foi projetado com partição horizontal transparente podendo oferecer alta disponibilidade para leituras e gravações. Outro benefício é que O Cosmos DB indexa automaticamente todos os dados – sem esquema, sem necessidade de nenhum índice – e serve consultas rapidamente.

Figura 9 – Principais Características do Azure Cosmos DB



Fonte:https://docs.microsoft.com/en-us/azure/cosmos-db/introduction.

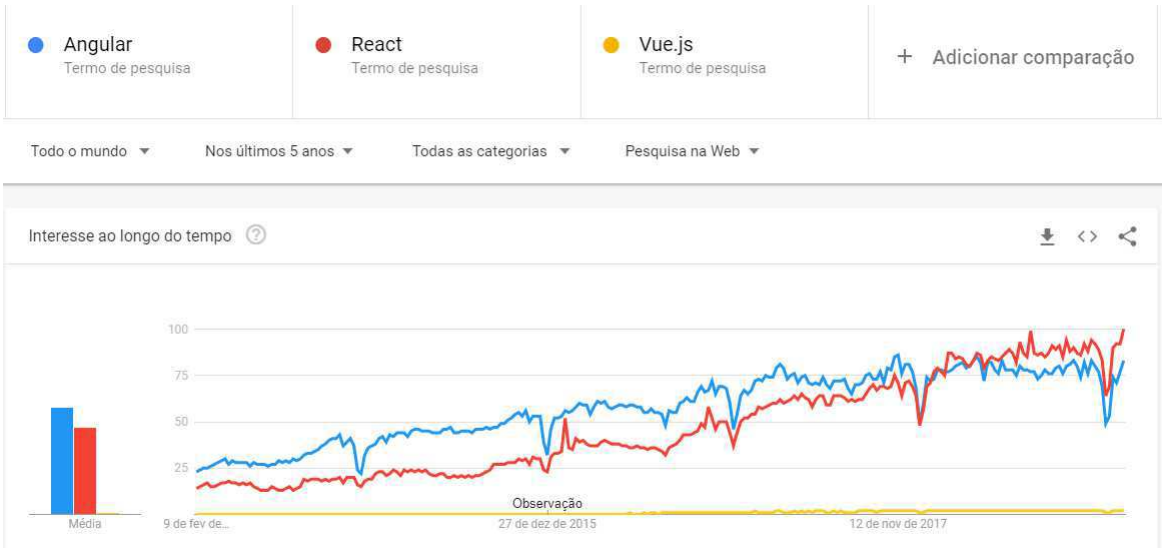
3.1.2 Interface Web

Com passar do tempo o uso de frameworks para o desenvolvimento de aplicações web tem crescido bastante. Isso ocorre porque desenvolver aplicações complexas e robustas utilizando puramente tecnologias como HTML, CSS e Javascript é uma tarefa bastante complexa e que consome muito tempo.

Os gráficos da Figura 10 demonstram a popularidade dos principais *frameworks web* dos últimos 5 anos segundo a quantidade de pesquisas feitas no Google. Do lado esquerdo tem-se um gráfico de barras indicando a preferencia ao longo dos anos, e à direita um gráfico de linhas que mostra a linha do tempo. Nota-se que o Angular tem maior preferência ao longo dos anos, mas atualmente está em segundo lugar, atrás do React.

Para o projeto proposto foi escolhido o Angular por ser uma plataforma que têm se mantido de forma regular entre as mais populares para os desenvolvedores e oferecer diversas funcionalidades que facilitam a construção de *websites*, como responsividade, interface REST, validação de formulários, formulários reativos e associação de dados em 2 sentidos.

Figura 10 – Popularidade dos *Frameworks Web* nos Últimos 5 anos



Fonte:https://trends.google.com/trends/explore?date=today%205-y&q=Angular,React,Vue.js.

3.1.3 Módulo RFID

Para o sistema de controle de acesso foi escolhido um kit do módulo RFID-RC522. Este módulo leitor RFID é baseado no chip MFRC522 da empresa NXP e é altamente utilizado em comunicação sem contato a uma frequência de 13,56MHz. Este chip, de baixo consumo e pequeno tamanho, permite sem contato ler e escrever em cartões que seguem o padrão *Mifare*, muito usado no mercado. A Figura 11 ilustra os componentes do kit utilizado.

Figura 11 – Kit do módulo RFID-RC522

Fonte: Próprio autor.

3.1.4 Placa de Desenvolvimento

O conceito da internet das coisas vem revolucionando o mercado tecnológico no Brasil e no mundo. Cada vez mais são necessários dispositivos conectados de baixo custo e longa bateria. Um microcontrolador (MCU) para IoT deve ser uma solução de prototipagem que apresenta processadores de baixa potência que suportam vários ambientes de programação, coletam dados do sensor usando firmware e os transferem para um servidor local ou baseado na nuvem.

Nesse contexto, a empresa chinesa Espressif Systems tem se destacado lançando chips para dispositivos IoT baseados em WiFi. Os módulos são geralmente de baixo custo, baixo consumo de energia e fáceis de usar. Os chips ESP vêm com muita flexibilidade e podem ser usados como módulos WiFi, conectados a outros microcontroladores ou usados em modos autônomos sem microcontroladores adicionais. A figura 12 ilustra dois dos principais chips lançados pela marca: o ESP8266 e o ESP32.

Figura 12 – Placas de Desenvolvimento ESP8266 e ESP32

Fonte: Próprio autor.

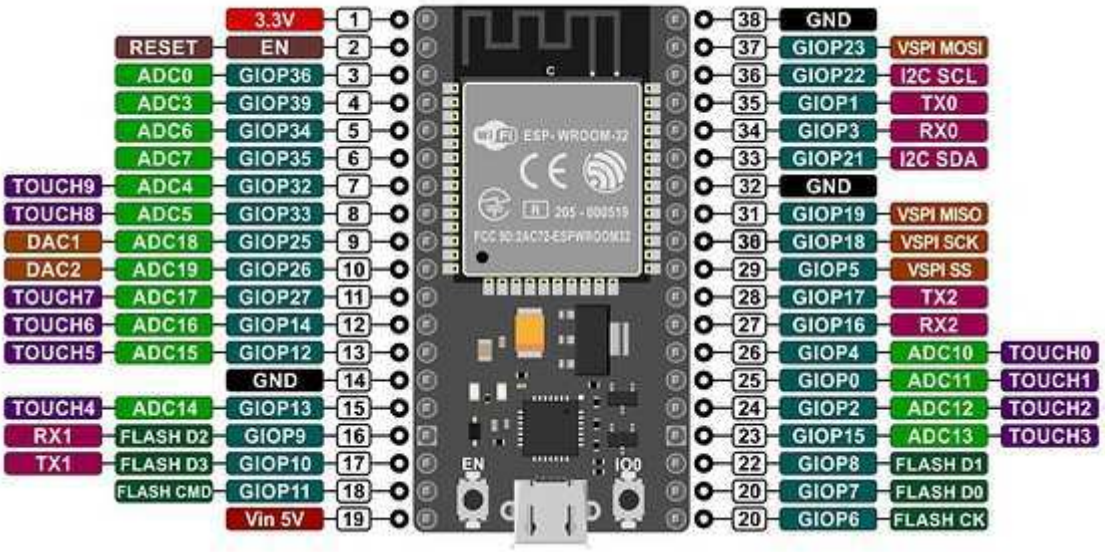
A Tabela 3 apresenta um quadro comparativo entre as duas placas de desenvolvimento. Nota-se que, embora seja um pouco mais cara, a ESP32 apresenta especificações muito melhores tanto em processamento quanto em memória e flexibilidade de uso. Dessa forma, pensando em um projeto que pode escalar com facilidade, foi escolhida a placa ESP-WROOM-32 com chip ESP32 para o projeto proposto. A programação para a placa pode ser feita nas linguagens C ou C++ utilizando o *framework* de desenvolvimento ESP IDF ou A IDE do Arduino. O *pinout* dessa placa é apresentado na Figura 13.

Tabela 3 – Comparação ESP8266 e ESP32

Especificação	**ESP8266**	**ESP32**
MCU	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
802.11 b/g/n Wi-Fi	HT20	HT40
Bluetooth	X	Bluetooth 4.2 and BLE
Frequência típica	80 MHz	160 MHz
SRAM	X	✓
Flash	X	✓
GPIO	17	36
Hardware /Software PWM	Nenhum / 8 canais	Nenhum / 16 canais
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	X	✓
Interface Ethernet MAC	X	✓
Sensor de Toque	X	✓
Sensor de temperatura	X	✓
Sensor de Efeito Hall	X	✓
Temperatura de trabalho	-40°C to 125°C	-40°C to 125°C
Preço	R$ (20 - 40)	R$ (40 - 70)

Fonte: https://makeradvisor.com/esp32-vs-esp8266/(Adaptado)

Figura 13 – *pinout* da placa ESP-WROOM-32



Fonte: http://forum.fritzing.org/t/esp32s-hiletgo-dev-boad-with-pinout-template/5357?u=steelgoose.

3.1.5 Ferramentas para Desenvolvimento

- Emuladores do Azure

  A plataforma de nuvem do Azure oferece opções de desenvolvimento em ambiente local. Isso permite a realização de testes na aplicação sem os custos de processamento e armazenamento da nuvem. Nesta aplicação foram utilizados dois emuladores do Azure, o emulador de armazenamento e o emulador do CosmosDB.

  O Emulador de Armazenamento do Microsoft Azure fornece um ambiente local que emula os serviços de blob, fila e tabela para fins de desenvolvimento. O Emulador do Azure Cosmos DB fornece um ambiente local que emula o serviço Azure Cosmos DB. Usando os emuladores é possível testar sem criar uma assinatura Azure ou incorrer em custos.

- Testes de Web API

  Sistemas desenvolvidos com base em serviços web têm se tornado cada vez mais comuns, principalmente pela flexibilidade que garantem aos clientes. Atualmente a maior parte destes serviços utiliza o modelo REST, com a troca de mensagens sendo realizada através de requisições HTTP.

  Uma das ferramentas que podem ser utilizadas para testar web APIs é o Postman, uma aplicação que permite realizar requisições HTTP a partir de uma interface simples e intuitiva, facilitando o teste e depuração de serviços REST.

  O Postman está disponível como uma aplicação para o browser Google Chrome e para desktop. Possui diversas funcionalidades úteis no desenvolvimento desse tipo de projeto como o envio e recebimento de dados no formato JSON.

3.2 Metodologia

Nesta seção serão detalhadas as metodologias utilizadas no processo de desenvolvimento da aplicação.

3.2.1 Introdução

Este trabalho foi desenvolvido no Laboratório de Instrumentação Eletrônica e Controle (LIEC), localizado na Universidade Federal de Campina Grande (UFCG). O local dispôs os equipamentos e suprimentos necessários ao desenvolvimento do projeto, bem como permitiu a realização de testes dos protótipos resultantes do trabalho realizado.

3.2.2 Metodologia de Desenvolvimento

A gestão e planejamento do projeto foi feita utilizando um processo ágil baseado na metodologia *Scrum* (SCHWABER, SUTHERLAND, 2017). No *Scrum*, os projetos são divididos em ciclos mensais chamados de *Sprints*. O Sprint representa um período de tempo dentro do qual um conjunto de atividades deve ser executado. As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*.

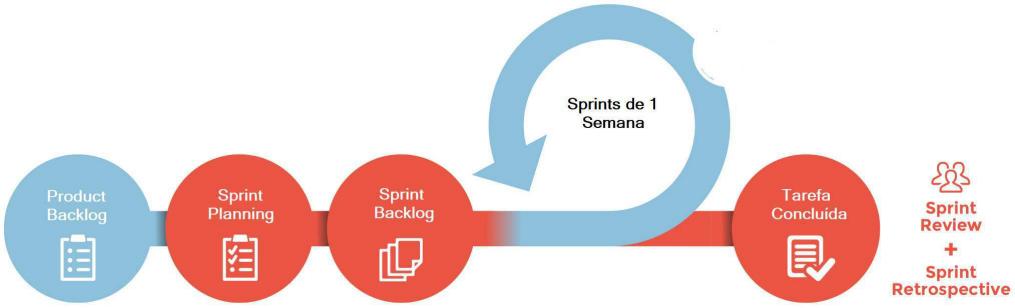
As tarefas alocadas em um Sprint são transferidas do *Product Backlog* para o *Sprint Backlog*. A cada dia de uma *Sprint*, a equipe faz uma breve reunião chamada *Daily Scrum*. No final de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma *Sprint Review Meeting*. Finalmente, faz-se uma *Sprint Retrospective* e a equipe parte para o planejamento do próximo *Sprint*. Assim reinicia-se o ciclo.

A abordagem utilizada neste trabalho é ilustrada na Figura 14. Não foi possível aproveitar ao máximo o *Scrum* para resolver seus problemas e aproveitar todos os benefícios do desenvolvimento de produtos usando o *Scrum*. Nesses casos existem os *ScrumButs*, que são as razões pelas quais não possível seguir a metodologia. Os *ScrumButs* do projeto são listados a seguir.

- Como este trabalho não será desenvolvido todos os dias, não poderão existir reuniões diárias.
- Como este é um trabalho de curta duração, foram utilizadas sprints semanais ao invés de mensais.

A Tabela 4 é o *Product Backlog* do sistema. A primeira coluna representa a lista de atividades a serem realizadas, a 2º coluna o esforço necessário para realizar cada atividade na notação *ScrumPoker* e a última representa a *Sprint* cronológica de cada atividade.

Figura 14 – Metodologia Scrum Utilizada



Fonte: https://pyxis-tech.com/en/agile-approaches/(Adaptado)

Tabela 4 – Product Backlog

Atividade	**Esforço**	**Sprint**
Estudar a plataforma do Azure e o funcionamento das suas aplicações sem servidor	13	1,2,3
Definir a arquitetura de hardware e software que deverá ser utilizada.	5	4
Implementar a arquitetura do software.	8	4
Desenvolver a funcionalidade para leitura dos dados do cartão RFID.	3	5
Desenvolver a funcionalidade de comunicação bidirecional do Microcontrolador com a Nuvem.	5	5
Definir a modelagem do banco de dados.	5	6
Desenvolver a camada de persistência integrada a nuvem.	13	6,7
Desenvolver uma interface web para o usuário integrada ao servidor em nuvem.	13	8,9
Desenvolver as funcionalidades CRUD de usuários.	5	10
Desenvolver as funcionalidades CRUD de locais.	5	10
Desenvolver a funcionalidade de autenticação de usuários.	5	11
Desenvolver a funcionalidade de histórico de acessos.	5	11
Realizar testes de integração e melhorias no sistema	?	12,13,14

Fonte: Próprio autor

4 MODELAGEM

Neste capítulo serão descritas as etapas relativas ao projeto de desenvolvimento da aplicação. Na primeira seção será apresentado o documento de visão, contendo informações gerais sobre a aplicação proposta. Em seguida será apresentado o documento de requisitos e, por fim, a modelagem e arquitetura da aplicação.

4.1 Documento de Visão

O propósito deste documento é coletar, analisar e definir as necessidades e características de alto nível do sistema. Fornece uma descrição da finalidade, do escopo e dos objetivos do projeto. Ele também define os produtos que se espera que o projeto libere.

4.1.1 Visão Geral

Em áreas industriais e empresariais o controle de acesso de pessoas é um fator crítico para o aperfeiçoamento da segurança. Além disso, a análise dos dados de fluxo de pessoas é uma ferramenta bastante útil para o monitoramento do ambiente.

Os sistemas de controle de acesso são dispositivos que visam manter a segurança de determinado local. Eles podem ser físicos ou eletrônicos e são projetados para controlar quem tem acesso a uma rede ou lugar específico, limitando o que os usuários podem ser autorizados a utilizar, como, por exemplo, os cartões magnéticos.

O controle ao acesso de pessoas pode ser integrado com meios eletrônicos, viabilizando a comunicação desses sistemas com a internet e, consequentemente, a troca de informações. Tal conexão permite que o ambiente possa ser controlado e monitorado remotamente, além de facilitar a análise dos dados coletados pelo sistema.

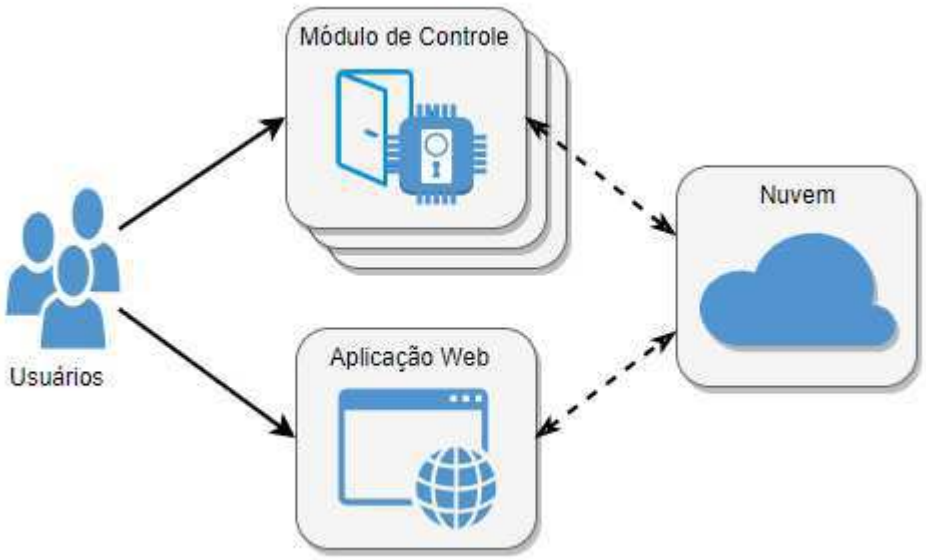
4.1.2 Solução Proposta

Propõe-se o desenvolvimento de um sistema informatizado para o controle de acesso de usuários. Este sistema será composto de três entidades principais: o Módulo de Controle, a Aplicação Web e os Serviços em Nuvem. A Figura 15 ilustra de forma abstrata como estas entidades se relacionam entre si e com os usuários do sistema.

De forma geral, o sistema constituirá de um conjunto de módulos de controle de acesso que serão instalados nas salas do laboratório. Cada módulo deverá ser capaz de receber as informações de entrada do usuário, realizar a comunicação com a nuvem e destravar a porta. As informações de acesso e fluxo de pessoas, bem como de cadastro,

deverão ser armazenadas em um banco de dados na nuvem e exibidas em uma plataforma web/móvel integrada.

Figura 15 – Visão geral do sistema



Fonte: Próprio autor.

4.1.3 Escopo Negativo

A seguir são apresentados alguns casos que não fazem parte do escopo do projeto.

- O sistema não apresenta informações sobre horários de saída dos ambientes, apenas entrada.
- O sistema não tem capacidade de autorizar ou alertar sobre acessos que sejam realizados por outras vias que não sejam a porta.
- O sistema não tem capacidade de autorizar ou alertar sobre acessos em casos nos quais a porta já estava aberta.
- O sistema não tem capacidade de funcionar em casos em que haja falta de energia ou internet no ambiente.

4.2 Especificação de Requisitos do Sistema

Nesta seção serão descritos os requisitos funcionais e não funcionais do sistema.

4.2.1 Requisitos Funcionais

RF1. Níveis de Acesso: As funcionalidades da plataforma web devem depender do nível de acesso do usuário. O sistema possui dois níveis de acesso: usuário administrador e usuário comum.

RF2. Login na Plataforma Web: Os usuários devem ter acesso a interface web via confirmação de E-mail e Senha. Caso o login não seja permitido, o sistema deve informar ao usuário que as informações de nome de usuário e senha estão incorretas.

RF3. Logoff na Plataforma Web: Após realizar o login, o usuário poderá realizar o logoff da aplicação web.

RF4. Cadastro de Usuários: O cadastro de usuários somente poderá ser realizado pela aplicação web. Deverão ser cadastradas as seguintes informações: Nome; E-mail; Endereço; Telefone; Senha;Código de Acesso; Locais permitidos para acesso;

RF5. Cadastro de Locais: O cadastro de locais somente poderá ser realizado pela aplicação web. Deverão ser cadastradas as seguintes informações: Nome fantasia; Endereço completo; Dispositivo instalado;

RF6. Cadastro de Módulos: O cadastro de módulos poderá ser realizado pela aplicação web ou diretamente pelo portal. Deverão ser cadastradas as seguintes informações: Nome do Dispositivo; Tipo de Segurança;

RF7. Armazenamento de Informações: Todas as informações de cadastro e acesso devem ser armazenadas em um Banco de Dados.

RF8. Módulo de Controle: O módulo de controle deve ser instalado na porta de todos os ambientes cadastrados. Deve conter os seguintes itens: Carcaça; Leitor RFID; Bateria; Relé; Controlador; Conectividade Wifi. Cada Módulo de Controle deverá manter um registro interno dos usuários com acesso ao ambiente em que o módulo está instalado.

RF9. Solicitação de acesso a um ambiente: Usuários com a finalidade de entrar em um ambiente cadastrado só poderão fazê-lo por meio de uma solicitação de acesso ao módulo de controle. Essa solicitação se dará mediante a apresentação de cartão RFID cadastrado.

RF10. Verificação dos dados de acesso: Sempre que houver conectividade, a verificação das informações de acesso deve ser feita na nuvem. Caso contrário, deverá ser feita utilizando o registro local do Módulo de Controle. Apenas usuários cadastrados e com permissão de acesso a determinado ambiente devem acessar esse ambiente.

RF11. Acesso a um ambiente: Caso a solicitação de acesso seja aprovada, o Módulo Instalado deve permitir a entrada no ambiente acionando o relé conectado a fechadura. Caso contrário, não deve permitir o acesso.

RF12. Indicador de Status: O visor contido no Módulo de Controle deve apresentar leds que indiquem os status das operações de leitura e validação da referida solicitação de acesso. Sendo LED vermelho para entrada não autorizada e verde para entrada autorizada.

RF13. Visualização de Acessos: Cada usuário Comum poderá visualizar apenas a sua lista de acessos na aplicação web. Os usuários Administradores, por usa vez, poderão além disso visualizar a lista de acesso dos outros usuários, com filtros por usuário, data e localidade.

4.2.2 Requisitos Não-funcionais

RNF1. A aplicação web deverá possui um design responsivo.
RNF2. A aplicação web deverá ser compatível com os diferentes browsers da atualidade. (e..g. Google Chrome, Firefox, Microsoft Edge, Opera e Safari).
RNF3. O módulo instalado deve informar o status da solicitação de acesso em menos de 3 segundos após a inserção da senha pelo usuário.

4.3 Casos de Uso

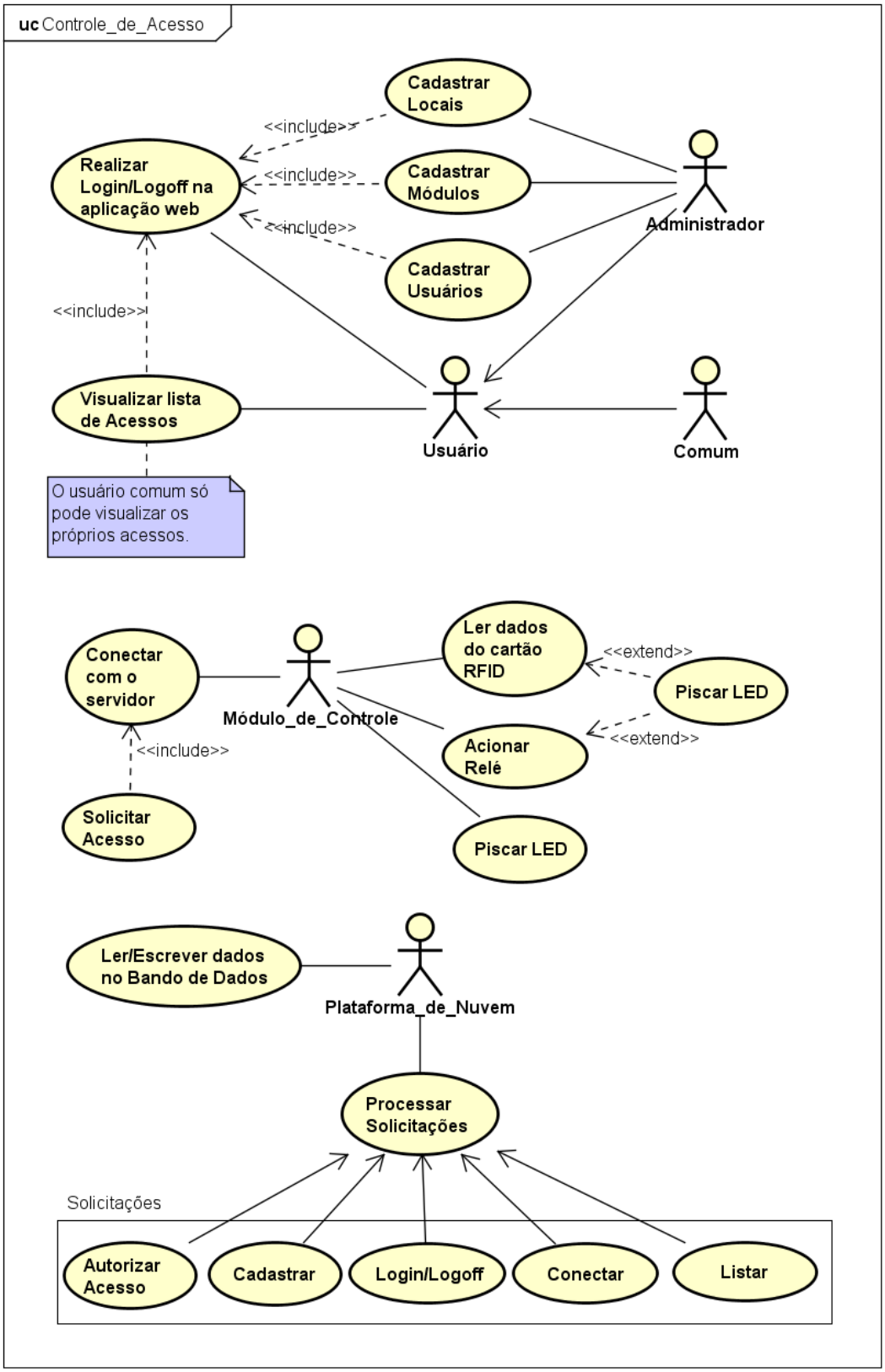
Nesta seção serão descritos de forma detalhada os atores e os casos de uso do sistema. O diagrama de Casos de Uso é apresentado na Figura 16. O sistema apresenta 3 tipos de atores principais: Usuário, Módulo_de_Controle e Nuvem.

- **Usuário**: O Ator Usuário representa as pessoas físicas que irão utilizar a aplicação web. Esse ator é dividido em dois outros atores: Administrador e Comum, que se relacionam com Usuário por uma relação de generalização.
  - **Administrador**: Pode visualizar o histórico de acessos de todos os usuários e cadastrar novos usuários;
  - **Comum**: Pode visualizar apenas o próprio histórico de acessos e editar algumas informações do próprio cadastro;
- **Módulo_de_Controle**: O módulo de controle representa a entidade física dos módulos que serão instalados nas portas dos ambientes e realizarão o controle de acesso.
- **Nuvem**: Ator que representa as ações realizadas pelo servidor em nuvem.

4.4 Projeto de Arquitetura

O projeto de arquitetura representa a estrutura dos dados e os compromitentes necessários para construção de um sistema. Ela considera o estilo de arquitetura do sistema, a estrutura e as propriedades dos seus componentes, bem como as inter-relações que ocorrem entre esses componentes (PRESSMAN, 2011). A Figura 17 ilustra uma visão geral da arquitetura do sistema, nota-se que ele é dividido em três componentes principais. Nesta seção serão detalhadas as estruturas de cada um desses componentes.

Figura 16 – Diagrama de Casos de Uso do Sistema



Fonte: Próprio autor.

Figura 17 – Arquitetura Geral do Sistema

Fonte: Próprio autor.

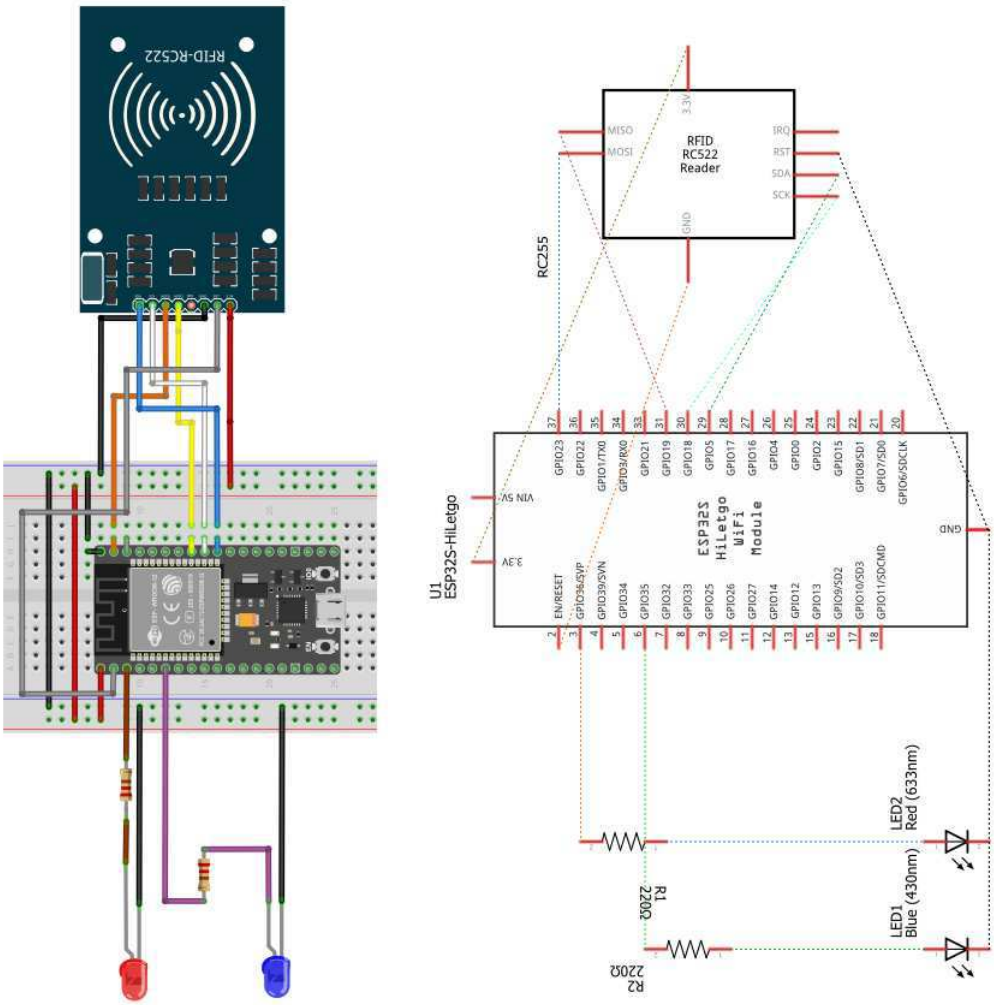
4.4.1 Módulo de Controle

O Módulo de Controle é o componente de *hardware* do sistema. Nele estão contidos os dispositivos com conexão a internet que realizam as tarefas físicas do sistema de controle de acesso, como leitura de dados do cartão RFID, exibição de status e acionamento do relé.

Cada dispositivo contém uma placa de desenvolvimento NodeMCU-32S, um leitor de cartões RFID RC522 e um conjunto de LEDs. O esquema elétrico de conexão entre estes itens pode ser visto na Figura 18. Nota-se que são utilizados os periféricos relativos a conexão SPI para o leitor de RFID e duas portas GPIO para os LEDs, além dos periféricos de alimentação 3.3V, *reset* e terra.

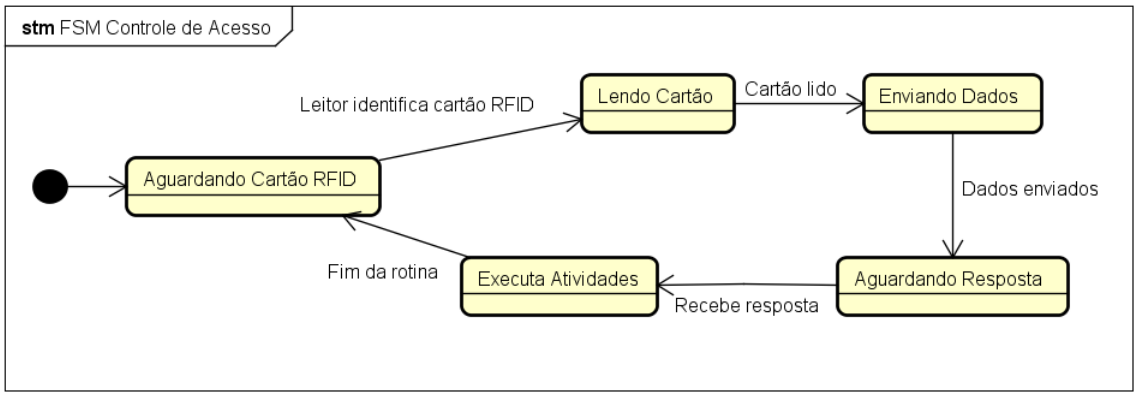
Como este módulo não realiza muitas ações em paralelo, foi escolhida uma arquitetura de software *baremetal* utilizando máquina de estados finitos. O diagrama de estados é representado na Figura 19. O estado inicial e padrão do microcontrolador é esperar por um evento de aproximação de tag RFID. Nesse estado o microcontrolador não pode ficar em *stand-by* pois necessita estar sempre conectado à internet. Assim que esse evento ocorre, o processo continua com a leitura e envio dos dados para a nuvem. Quando recebe uma resposta, processa a informação autorizando ou não a entrada do usuário e volta para o estado inicial. A comunicação do módulo com a nuvem é feita através do protocolo MQTT pelo *hub IoT*.

Figura 18 – Esquemático de Conexão dos Elementos do Dispositivo IoT



Fonte: Próprio autor.

Figura 19 – Máquina de Estados Finitos do *software* embarcado



Fonte: Próprio autor.

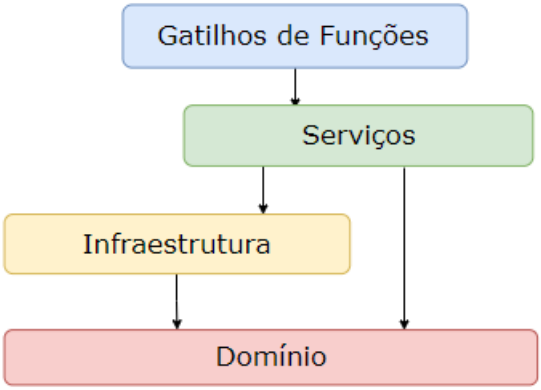
4.4.2 Serviços da Nuvem

O centro desta aplicação é a nuvem, pois nela são processadas todas as informações importantes para o funcionamento do sistema. Nesse contexto, por se tratar de um sistema sem-servidor (*serverless*), os principais componentes do sistema são os Aplicativos de Funções do *Azure*.

Todas as funcionalidades dependem do Aplicativo de Funções. As requisições realizadas pelo módulo de controle são recebidas pelo *Azure IoT Hub* e transferidas paras as funções por evento. O acesso ao banco de dados e as respostas as requisições HTTP são também feitas por chamadas às funções.

Inicialmente, para este módulo foi escolhida uma arquitetura em camadas, como ilustrado na Figura 20. Nesta arquitetura o sistema se comunica com entidades externas pelos gatilhos das Funções. Estes interagem com a camada de serviços, na qual estão implementadas as regras do negócio. A camada de serviços acessa o banco de dados pela camada de infraestrutura e tem acesso aos tipos de dados pela camada de domínio. Deste modo, cada camada tem sua própria responsabilidade.

Figura 20 – Arquitetura de Camadas da Aplicação *Serverless*



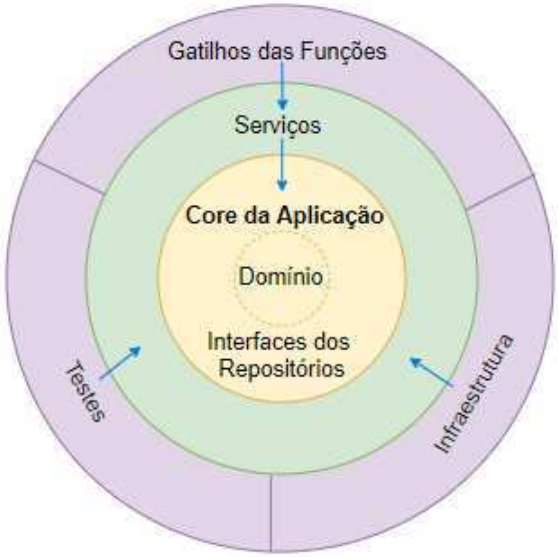
Fonte: Próprio autor.

Uma desvantagem dessa abordagem tradicional de camadas é que as dependências em tempo de compilação são executadas de cima para baixo. Ou seja, a camada de funções depende dos serviços, que depende da infraestrutura e do domínio. Isso significa que os serviços, que geralmente contém a lógica mais importante do aplicativo, depende dos detalhes de implementação de acesso a dados. Testar a lógica de negócios em tal arquitetura é geralmente difícil.

Uma abordagem bastante utilizada para contornar este problema é utilizar arquiteturas baseadas no Princípio da Inversão de Controle. Nele, o fluxo do programa depende das dependências construídas durante a execução do programa. Esse fluxo dinâmico é possibilitado pelas interações de objetos que são definidas através de abstrações. Isso pode ser obtido utilizando padrões como o de injeção de dependência.

Dessa forma, com o objetivo de tornar o software mais modular e possibilitar a realização de testes unitários, foi implementada uma versão utilizando uma Arquitetura Limpa (*Clean*), também conhecida como Arquitetura Cebola (*Onion*). A Figura 21 apresenta uma visão dessa arquitetura para o projeto proposto. Essa arquitetura coloca a lógica de negócios e o modelo no centro da aplicação. Em vez de a lógica de negócios depender do acesso a dados ou de outras preocupações de infraestrutura, essa dependência é invertida.

Figura 21 – Arquitetura Cebola da Aplicação *Serverless*



Fonte: Próprio autor.

4.4.3 Aplicação Web

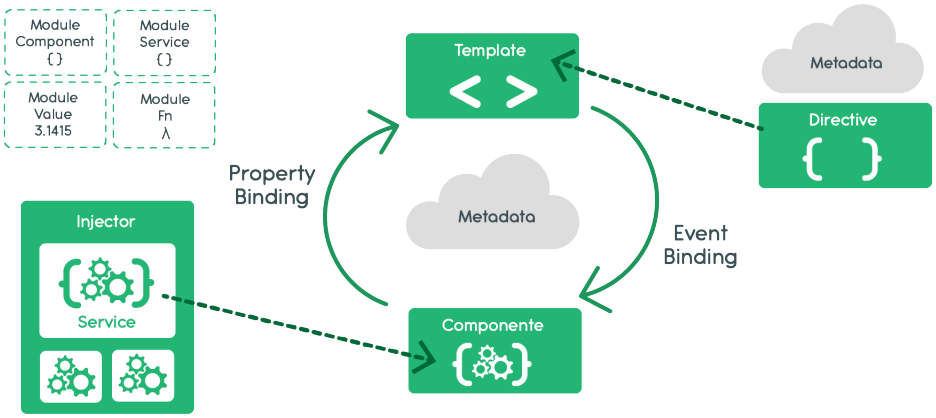
A aplicação web foi feita utilizando o *framework* Angular. Os arquivos estáticos gerados foram armazenados como blobs do serviço de armazenamento do Azure. O Angular é um *framework* para criação de aplicativos clientes em HTML e TypeScript. Os blocos de construção básicos de um aplicativo Angular são *NgModules*, que fornecem um contexto de compilação para os *componentes*. *NgModules* coletam código relacionado em conjuntos funcionais; um aplicativo Angular é definido por um conjunto de *NgModules*.

Os componentes(*components*) definem exibições: conjuntos de elementos de tela que o Angular pode escolher entre e modificar de acordo com a lógica e os dados do programa. Eles também usam serviços(*services*) que fornecem funcionalidade específica. Os provedores de serviços podem ser injetados em componentes como dependências, tornando seu código modular, reutilizável e eficiente (Angular,21–).

Ambos os componentes e serviços são simplesmente classes, com decoradores que marcam seu tipo e fornecem metadados(*metadata*) que informam ao Angular como usá-los. Os metadados de componentes associam a um modelo(*template*) que combina HTML

comum com diretivas(*directives*) angulares e os metadados de serviço fornecem as informações que o Angular precisa para disponibilizá-los aos componentes por meio da injeção de dependência (*injector*). A Figura 22 apresenta um diagrama com as principais partes de uma aplicação Angular.

Figura 22 – Diagrama de Blocos de uma Aplicação Angular



Fonte: https://blog.algaworks.com/o-que-e-angular/.

4.5 Visão Lógica

Nesta seção são apresentados alguns diagramas que representam a visão lógica do sistema. Essa visão se baseia na descrição apresentada por (KRUTCHEN,1995), que divide a arquitetura de software em um sistema de visões chamado "4+1".

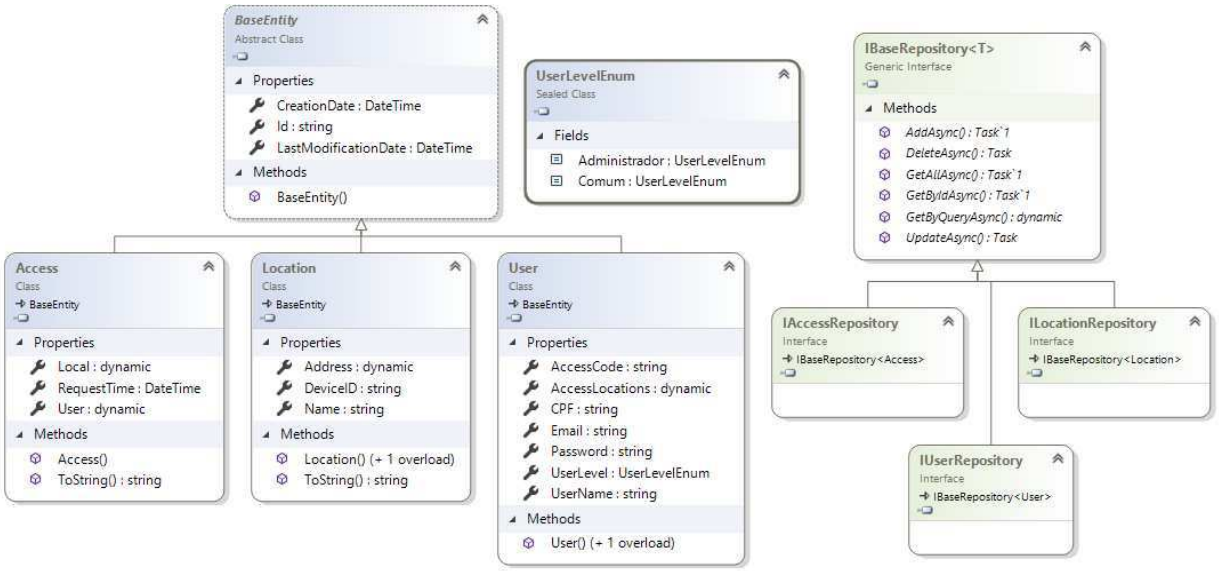
4.5.1 Diagrama de Classes

O diagrama de classes completo do software sem-servidor pode ser encontrado no Anexo A. Nesta seção são apresentados alguns exemplos e conceitos que foram utilizados na implementação. As classes que representam as entidades do domínio da aplicação são apresentadas na Figura 23. As entidades do domínio são as classes *Access*, *Location* e *User*, ambas herdam da classe abstrata *BaseEntity*.

Para realizar a persistência dos dados foi utilizado o padrão Repositório(Repository). Conceitualmente, um repositório encapsula um conjunto de objetos armazenados no banco de dados e operações que podem ser executadas neles, fornecendo uma maneira mais próxima da camada de persistência. Repositórios, também, suportam o propósito de separar, claramente e em uma direção, a dependência entre o domínio de trabalho e a alocação ou mapeamento de dados. (GAMMA,1995)

Para a implementação do repositório foram criadas as interfaces *IAccessRepository*, *ILocationRepository* e *IUserRepository*, ambas implementam a interface *IBasesRepository*.

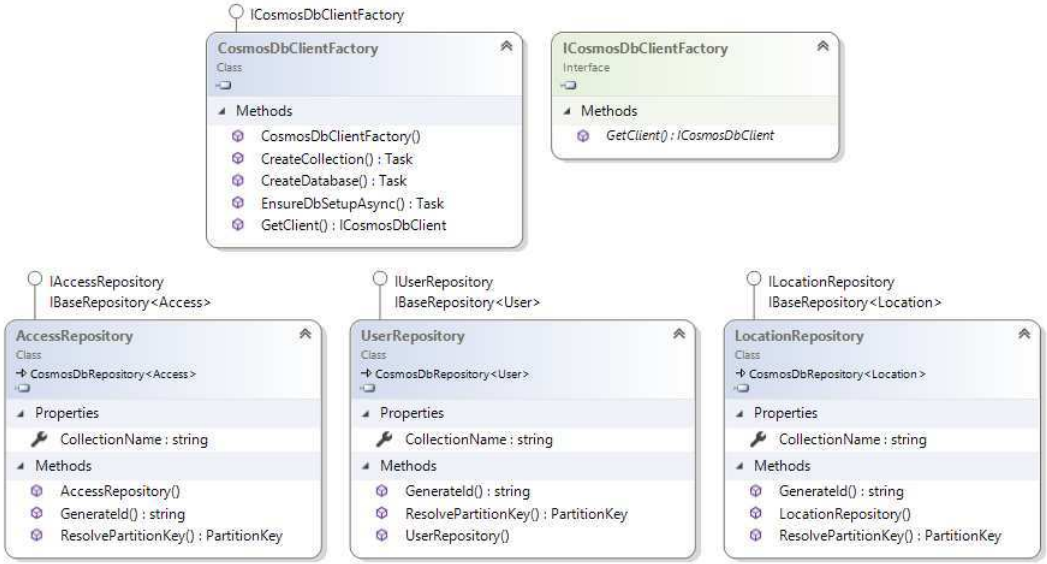
Figura 23 – Diagrama de Classes do Domínio



Fonte: Próprio autor.

Essa forma de projetar o repositório utiliza o padrão de projeto Estratégia(*Strategy*). Neste padrão é definido um conjunto de algoritmos encapsulados e intercambiáveis, permitindo que o algoritmo se diversifique independentemente dos clientes que os utilizam (GAMMA, 1995). A interface *IBaseRepository* representa a estratégia dessa estrutura e descreve os contratos para a implementação de métodos CRUD.

Figura 24 – Classes da Camada de Infraestrutura



Fonte: Próprio autor.

A implementação das interfaces dos repositórios e o acesso ao banco são feitos na camada de Infraestrutura. A Figura 24 ilustra algumas classes dessa camada. Para a criação e os acesso ao banco de dados CosmosDB foi criada a interface *ICosmosDBClient-Factory*, que reflete o padrão Fábrica Abstrata (*Abstract Factory*). Este padrão fornece

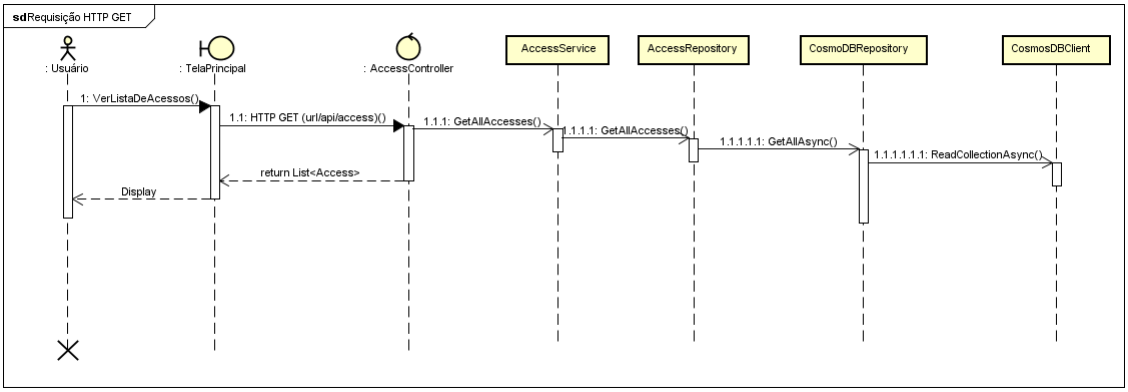
uma interface para criar famílias de objetos dependentes relacionados sem especificar suas classes concretas (GAMMA,1995).

Outro padrão utilizado foi a injeção de dependências. Os construtores das classes que implementam as interfaces de Repositórios são "injetados"com um objeto da classe *CosmosDBClientFactory*. Esse padrão é uma forma de implementar o Princípio da Inversão de Controle citado anteriormente.

4.5.2 Diagramas de Sequência

Os principais tipos de eventos que ativam os gatilhos de funções são as chamadas por interfaces HTTP e *Hub* de eventos IoT. Os diagramas de sequência para alguns casos destes tipos de gatilhos são apresentados nas Figuras 25 e 26.

Figura 25 – Diagrama de Sequência para um Requisição HTTP GET



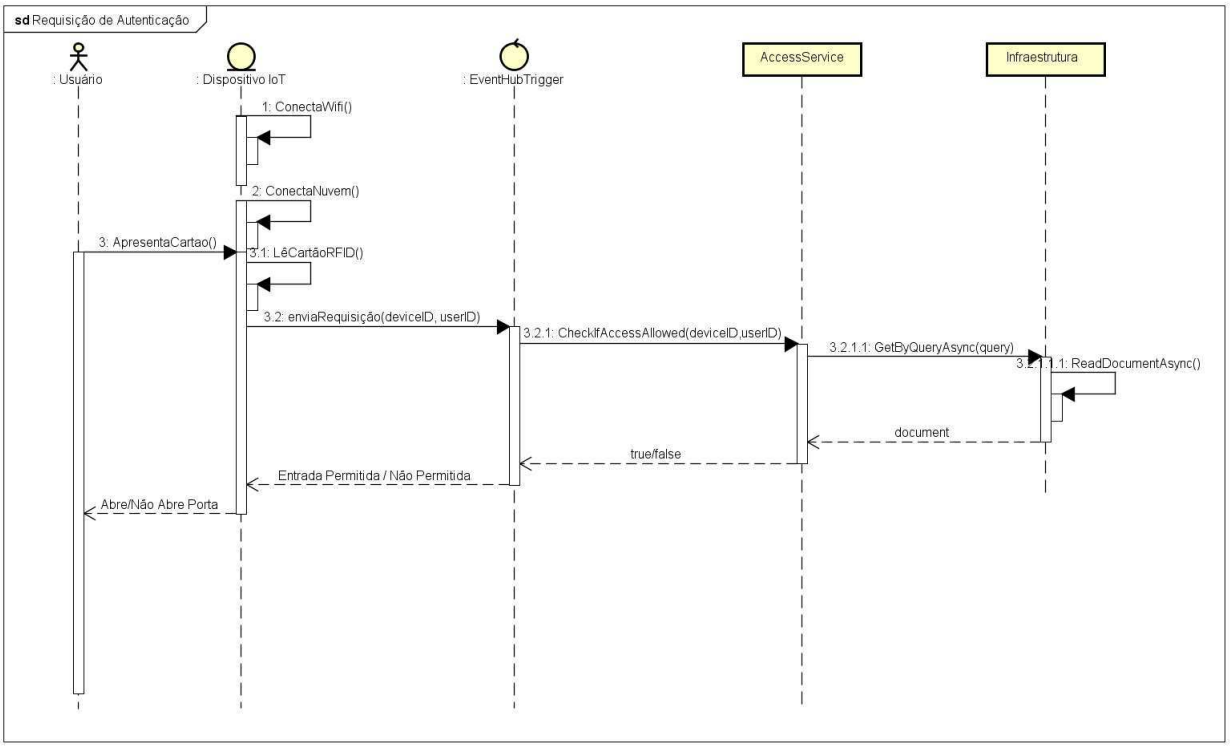
Fonte: Próprio autor.

4.6 Modelagem dos Dados

O *Azure Cosmos DB* é um banco de dados orientado a documentos. Um banco de dados orientado a documentos é um tipo de banco de dados não relacional projetado para armazenar dados semiestruturados como documentos. Esses bancos de dados são intuitivos para desenvolvedores pois os dados no nível do aplicativo normalmente são representados como um documento JSON. Pela não existência de esquemas esses bancos de dados se tornam muito mais flexíveis, mas são desvantajosos em consistência das informações.

Embora bancos de dados sem esquemas facilitem muito a adoção de mudanças no modelo de dados, é importante considerar a melhor forma de armazenar esses dados. Existem duas formas de relacionar dados em um documento: inserção e referência. No primeiro os dados das entidades são armazenados juntos, esse tipo de abordagem é mais usado em relações 1:1 (lê-se: um para um) ou 1:Poucos e provê uma melhor performance para leitura de informações. Já o segundo indica armazenar apenas uma informação, como

Figura 26 – Diagrama de Sequência para um Requisição IoT

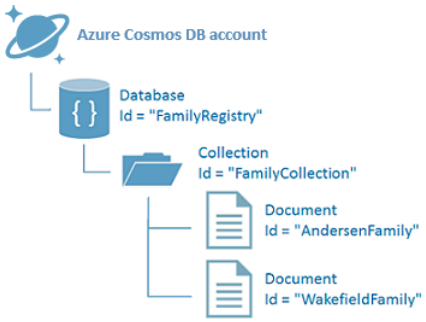


Fonte: Próprio autor.

o id, da entidade relacionada. Referências são indicadas para casos de relações 1:Muitos e Muitos:Muitos e, em geral, otimiza as operações de escrita em documentos.

A Figura 28 ilustra um exemplo de estrutura de documentos em um banco de dados Cosmos DB. O banco de dados(*Database*) é composto de coleções(*Collections*) e essas coleções armazenam os documentos com os dados. Neste projeto foi adotada uma modelagem com três coleções de dados: *UserCollection*, *LocationCollection* e *AccessInfo-Collection*.

Figura 27 – Exemplo de Estrutura de um Banco de Dados Cosmos DB



Fonte: https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-get-started.

Dado o contexto da aplicação, pode-se considerar que as coleções de locais e usuários serão mais utilizadas para leitura, visto que não é sempre que é necessário o cadastro

de tais entidades e que elas representam uma relação 1:Poucos. Já a coleção de acessos deve ser mais utilizada para escrita, pois a cada acesso a um local um novo documento é criado e sua relação com as outras entidades é de Muitos:Muitos. Uma forma de reduzir o gasto de escrita da inserção é inserir apenas dados que serão importantes em consultas. Essas foram as hipóteses utilizadas para fazer a modelagem da aplicação. A Figura **??** apresenta exemplos da modelagem adotada.

Figura 28 – Exemplo de Documentos da Modelagem Adotada

Fonte: Próprio autor.

5 DESENVOLVIMENTO

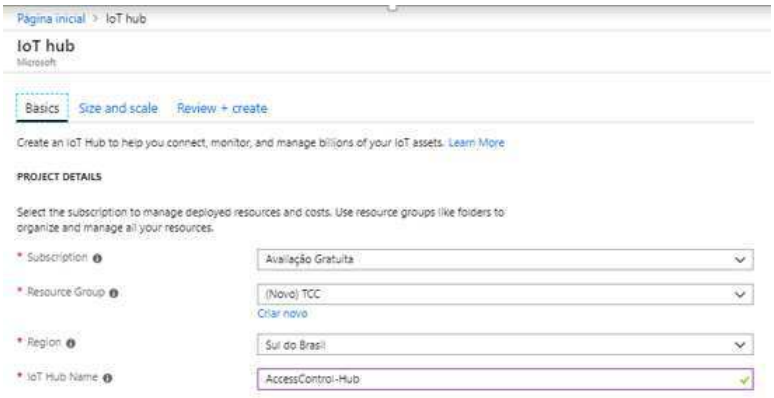
Neste capítulo serão apresentados detalhes do desenvolvimento do software. Ele está dividido em três seções, cada uma correspondendo a um dos módulos descritos na seção Projeto de Arquitetura do capitulo anterior.

5.1 Configuração Inicial

O primeiro passo no desenvolvimento da aplicação foi a criação e uma assinatura no Portal do Azure. Para isso é necessário informar algumas informações pessoais e fornecer dados de um cartão de crédito. Embora esses dados financeiros sejam requisitados, o primeiro mês de uso é gratuito. No portal do Azure é possível criar todos os recursos da nuvem que serão necessários na aplicação.

Um recurso essencial é o Hub Iot do Azure. Utilizando o plano gratuito é possível criar apenas 1 Hub (cada Hub suporta milhares de dispositivos conectados) com limite de 8000 mensagens por dia. A principal informação a ser fornecida na criação é o nome do hub, que deve ser único. A Figura 29 ilustra as informações básicas necessárias para a criação de um hub. Com o Hub criado é possível acessar suas strings de conexão que são utilizadas na comunicação com dispositivos e serviços. Também é possível adicionar dispositivos ao hub pelo portal, mas nessa aplicação isso será feito de forma automatizada.

Figura 29 – Informações Básicas para a Criação de um Hub IoT.

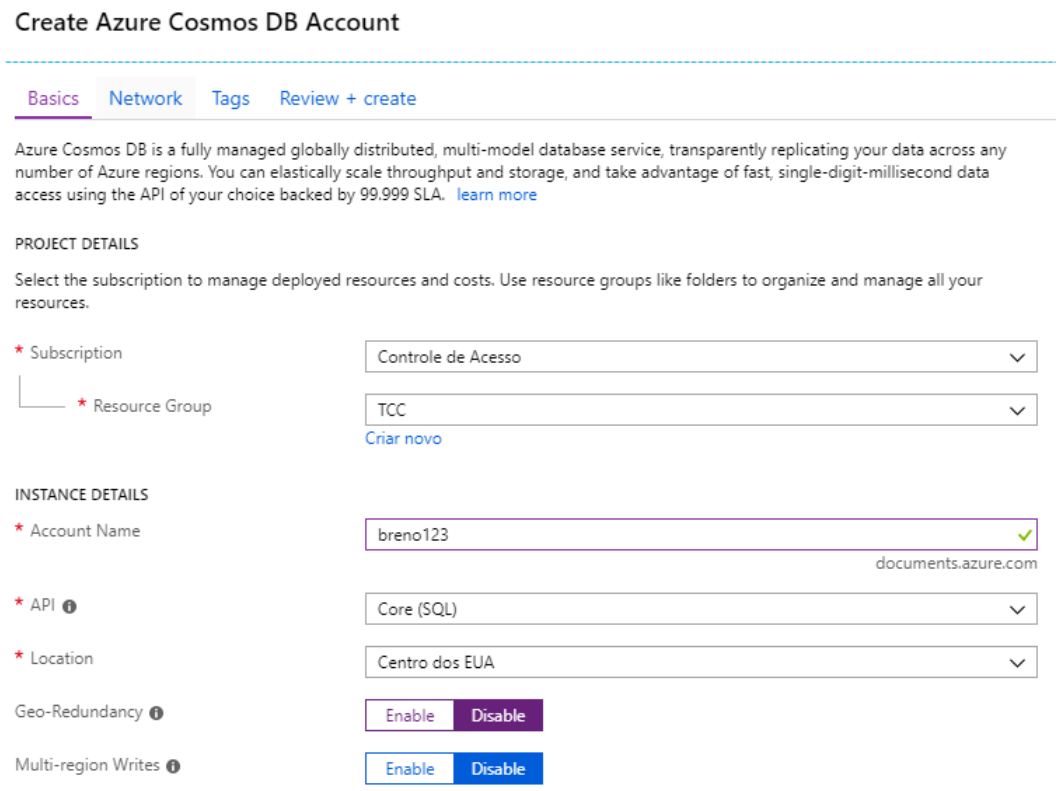


Fonte: Próprio autor.

Outro recurso que foi criado utilizando o Portal do Azure é uma conta no Azure Cosmos DB. Essa conta é necessária para obter as informações como link e string de conexão para que os serviços possam acessar o banco. A Figura 30 apresenta as informações básicas que são necessárias para a criação de uma conta no Azure CosmosDB. Nota-se

que é possível escolher o local de armazenamento global e a API que será utilizada. No desenvolvimento dessa aplicação foi utilizada a API SQL.

Figura 30 – Informações Básicas para a Criação de uma conta no Azure Cosmos DB



Fonte: Próprio autor.

5.2 Módulo de Controle

Como foi dito previamente, o módulo de controle foi implementado utilizando a placa de desenvolvimento ESP-WROOM-32. A programação da placa pode ser feita utilizando a IDE Arduino, entretanto pelo fato desta IDE não apresentar recursos auxiliares como *intellisense*, foi utilizada a extensão do Visual Studio Code para Arduíno, que embora ainda esteja na versão prévia, apresenta recursos que facilitam o desenvolvimento.

Quando um projeto é criado, é necessário adicionar a URL ao arquivo *settings.json* para que o ambiente de desenvolvimento reconheça placas com o chip ESP32. A linguagem C foi utilizada para a programação da placa. O código completo pode ser visto no Anexo B.

Para comunicar o microcontrolador com a nuvem foram inclusas as bibliotecas *AzureIotHub* e *Esp32MQTTClient*. Elas apresentam métodos que permitem a conexão, envio e recepção de dados com o Hub Iot do Azure utilizando o protocolo MQTT. Para a

conexão é chamada a função *Esp32MQTTClient_Init* passando como parâmetro a string de conexão do Hub IoT que se deseja conectar.

Quando uma leitura é realizada com sucesso as informações de ID do cartão e ID do dispositivo são enviadas para a nuvem no formato JSON utilizando a função *Esp32MQTTClient_SendEventInstance*. Por sua vez, quando uma mensagem é enviada da nuvem para o dispositivo o método de *callback MessageCallback* é acionado. Um exemplo de mensagem enviada pode ser visto abaixo.

```
{
    "DeviceID":"123", "UserAccessCode":"q2:4e:2a:12"
}
```

A funcionalidade de leitura de cartões RFIF requer a adição de mais duas bibliotecas, a *MFRC522* para as funções do módulo e a biblioteca *SPI* para a comunicação com o módulo RC522. A criação de uma instanciação da biblioteca MFRC522 requer que sejam configurados os pinos de RST e SS(SPI), que correspondem aos valores 3 e 5 na placa ESP-WROOM-32. O código utilizado para fazer a leitura do ID do cartão RFID pode ser visto a seguir na função *getID()*.

```
uint8_t getID() {
  // Verifica aproximacao de tag RFID
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
  continue
    return 0;
  }
  // Realiza a leitura pela serial
  if ( ! mfrc522.PICC_ReadCardSerial()) {
  continue
    return 0;
  }
  //Le os primeiros 4 Bytes (Representam o UID do cartao)
  for ( uint8_t i = 0; i < 4; i++) {  //
    readCard[i] = mfrc522.uid.uidByte[i];
  }
  // Para de ler a serial
  mfrc522.PICC_HaltA(); // Stop reading
  return 1;
}
```

5.3 Aplicação Sem-Servidor

O desenvolvimento da aplicação sem-servidor foi feito de forma local no Visual Studio 2017 utilizando a plataforma .Net Core 2.2. Embora seja possível codificar as funções diretamente da plataforma do Azure, tal procedimento não permite a criação de aplicações mais robustas, não oferece recursos como *debug* e a depender do uso pode levar a custos desnecessários em fase de desenvolvimento.

Primeiramente, com o intuito de testar a aplicação de forma geral, foi desenvolvida a arquitetura de camadas. A estrutura de pastas pode ser vista na Figura 31. A camada *ApplicationCore* apresenta as classes POCO apresentadas na modelagem. Na camada *Infrastructure* é realizado o acesso ao banco de dados CosmosDB do Azure. Para isso é necessário adicionar o pacote *Microsoft.Azure.DocumentDB.Core* ao projeto.

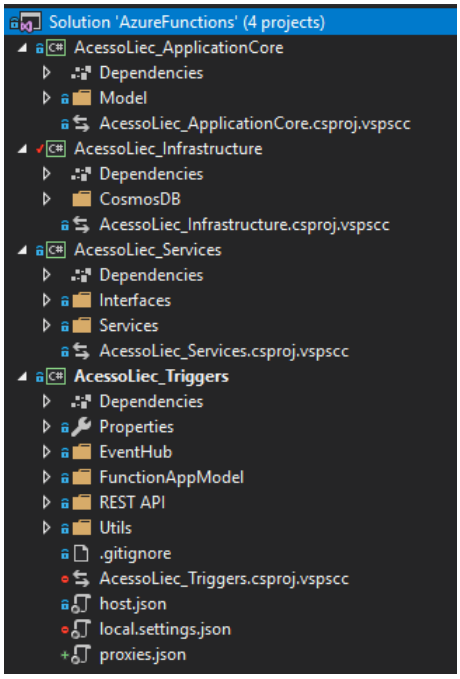
Para realizar a conexão com o banco é necessário informar o link e a string de conexão do Cosmos DB. Para testes foram utilizadas as informações fornecidas pelo Emulador do Azure Cosmos DB. A camada *Services* contém os métodos de cada modelo que se comunicam com o banco de dados. O código abaixo apresenta a interface do serviço relacionado a entidade *location*.

```
namespace AcessoLiec_Services.Interfaces
{
    public interface ILocationService
    {
        void CreateLocation(Location location);
        IEnumerable<Location> GetAllLocations();
        Location GetLocationById(string id);
        Location GetLocationByDeviceId(string id);
        void DeleteLocation(string id);
    }
}
```

Para trabalhar com os Aplicativos de Funções do Azure no VS2017 é necessário instalar a extensão Azure Functions and Web Jobs Tools. Com isso é possível criar um Aplicativo e suas Funções. Quando um aplicativo é criado, dois arquivos JSON de configuração são criados: *host.json* e *local.settings.json*. O primeiro apresenta a informação da versão do *Azure Functions* que está sendo utilizada, no caso a versão 2.0, e o outro com informações de configuração para os gatilhos, como os *enpoints* do hub de eventos e da conta de armazenamento do azure. Para testes foram utilizadas as informações do Emulador de Armazenamento do Azure.

O código da função *CreateLocation* apresentado a seguir ilustra um exemplo de implementação de gatilho HTTP. O primeiro parâmetro é um tipo *HttpRequest* com as-

Figura 31 – Estrutura da Aplicação com Arquitetura em Camadas

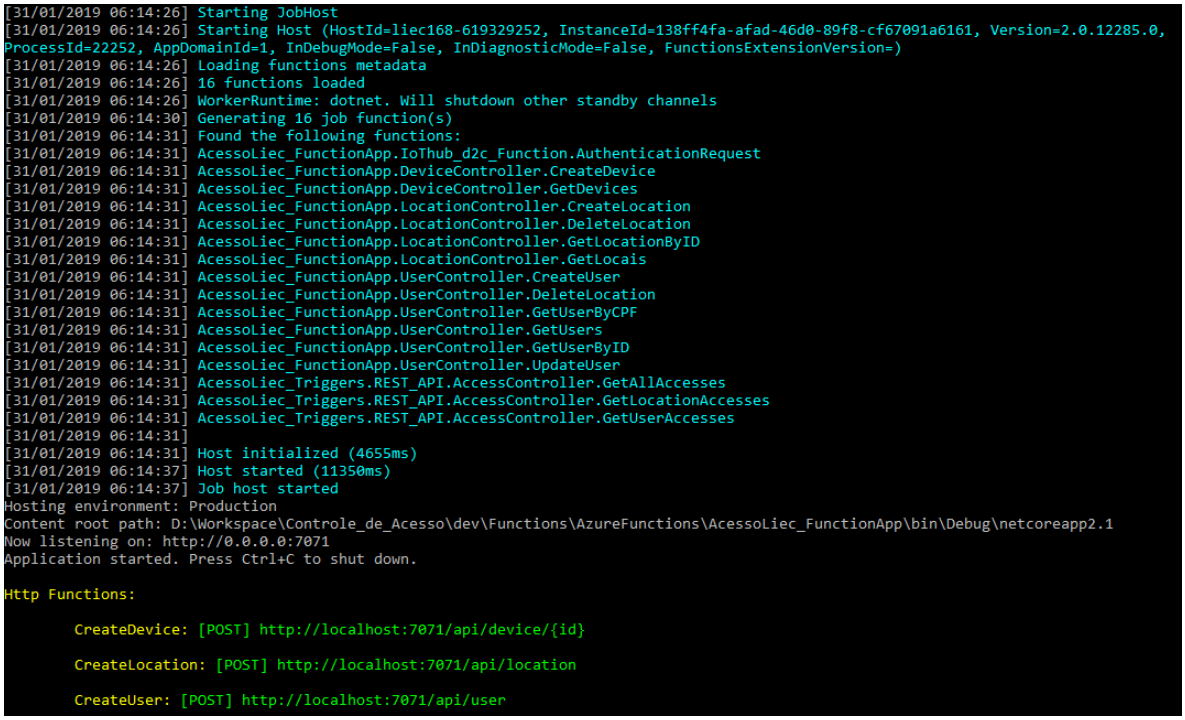


Fonte: Próprio autor.

sociações do gatilho relacionadas ao nível de acesso da função, o tipo de requisição HTTP e o roteamento, e o outro parâmetro contém informações de log. A informação recebida pelo gatilho é convertida de Json para o tipo *location* e tenta-se criar o objeto. Quando um projeto de Funções do Azure é compilado a ferramenta Azure Functions CLI inicializa e hospeda a aplicação em , como apresentado na Figura 32.

```
[FunctionName("CreateLocation")]
 public static async Task<IActionResult> CreateLocation(
     [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "location")]HttpRequest req,
     ILogger log)
 {
     log.LogInformation("C# HTTP trigger CreateLocation request.");

     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

     Location location = new Location();
     location = JsonConvert.DeserializeObject<Location>(requestBody);

     try
     {
         locationService.CreateLocation(location);
     }
     catch (Exception e)
     {

         return new BadRequestObjectResult(e.Message);
     }

     return new OkObjectResult(true);

  }
```

Figura 32 – Informações Apresentadas no Azure Functions CLI



Fonte: Próprio autor.

Após ser feita a aplicação para a arquitetura em camadas e validadas as informações decidiu-se reescrever o código utilizando a arquitetura Limpa. Além da utilização do padrão Repositório, a principal modificação foi a utilização do Princípio da Inversão de Controle. A documentação do aplicativo de funções do Azure não apresenta uma forma oficial de se implementar essa inversão. Dessa forma, foi utilizado o pacote de terceiros *AzureFuntions.Autofac*. A função *AutofacConfig* apresentada a seguir apresenta a forma como são registrados os tipos nessa abordagem.

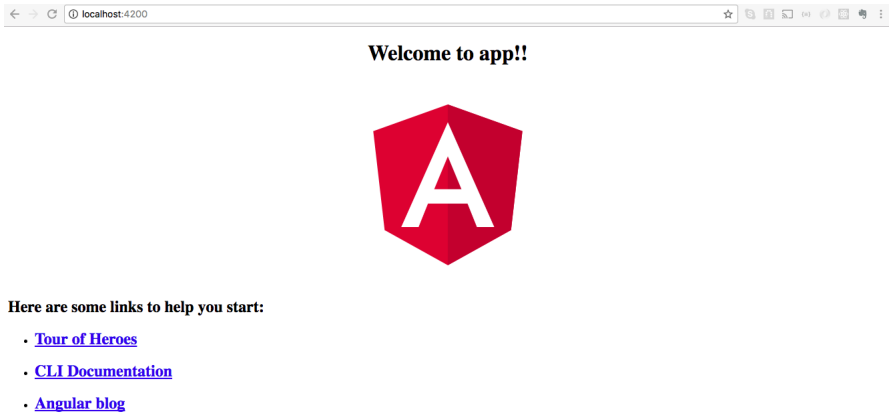
```
public AutofacConfig(string functionName) {
    AzureFunctions.Autofac.Configuration.DependencyInjection.Initialize(builder => {
        builder.RegisterType<CosmosDbClientFactory>()
                .As<ICosmosDbClientFactory>().InstancePerRequest();
        // Repositories
        builder.RegisterAssemblyTypes(typeof(UserRepository).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
        builder.RegisterAssemblyTypes(typeof(LocationRepository).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
        builder.RegisterAssemblyTypes(typeof(AccessRepository).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
        // Services
        builder.RegisterAssemblyTypes(typeof(UserService).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
        builder.RegisterAssemblyTypes(typeof(LocationService).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
        builder.RegisterAssemblyTypes(typeof(AccessControlService).Assembly)
                .AsImplementedInterfaces().InstancePerLifetimeScope();
    }, functionName);

}
```

5.4 Aplicação Web

A aplicação web foi desenvolvida utilizando o *framework* Angular versão 7. A instalação da interface de linha de comandos do Angular requer que as versões mais recentes do Node.js e do NPM estejam instaladas no computador. O código abaixo apresenta a linhas de comando necessárias para criar um projeto Angular com os componentes necessários para a aplicação. A primeira linha cria uma nova aplicação Angular, a segunda muda o diretório e da linha 03 a 09 são criados componentes. A última linha inicializa o *host* do Angular na url , como mostrado na Figura 33.

```
01- ng new myAngularApp
02- cd myAngularApp
03- ng gererate component local/create-local
04- ng gererate component local/read-local
05- ng gererate component local/update-local
06- ng gererate component user/create-user
07- ng gererate component user/read-user
08- ng gererate component user/update-user
09- ng gererate component access/read-access
10- ng serve
```

Figura 33 – Tela Padrão do Angular



Fonte: Próprio autor.

Cada componente gerado é formado por 4 arquivos. Um arquivo de extensão HTML, um CSS e dois Typescript. Nos arquivos HTML e CSS são definidos os *Templates* de aplicações Angular. Um dos arquivos Typescript serve para testes e outro implementa a lógica da lógica da aplicação. Toda aplicação Angular apresenta um componente principal chamado *app* e além disso apresenta um módulo com o mesmo nome onde devem ser declarados e importados todos os outros componentes e módulos da aplicação.

Um desses outros módulos se chama *app-routing.module.ts* e nele são definidas todas as rotas da aplicação. Isso é exemplificado no código abaixo. Nele o caminho da url 'user/create' é redirecionado para o componente *CreateUserComponent*, de forma similar

para o local e o caminho padrão ' ' é redirecionado para o componente *ReadAccessComponent*

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { CreateLocalComponent } from './local/create-local.component';
import { CreateUserComponent } from './user/create-user.component';
import { ReadAccessComponent } from './access/read-access.component'

const routes: Routes = [
  { path : 'user/create', component : CreateUserComponent },

  { path : 'local/create', component : CreateLocalComponent },

  { path: '', component : ReadAccessComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

A forma de se trabalhar com requisições HTTP em angular é através dos serviços. Os serviços do Angular são objetos substituíveis que são conectados usando a injeção de dependência (DI). Os serviços HTTP usam a biblioteca reativa *RxJS* do angular e retornam dados do tipo *Observable*. Para criar um serviço em Angular basta digitar o comando *ng generate service NomedoServiço*. O Código abaixo representa o serviço que implementa uma requisição GET no nosso aplicativo de funções.

```
import { Injectable } from '@angular/core';
import {Observable} from "rxjs/index";
import { HttpClient } from '@angular/common/http';

@Injectable({ providedIn: 'root' })

export class AccessService {

  constructor(private http: HttpClient) { }
  public baseUrl = "https://localhost:7071/api";
  public accessUrl = this.baseUrl + "/access/";

  getAllAccesses() : Observable<any> {
    return this.http.get<any>(this.accessUrl);
  }
}
```

Quanto a manipulação de dados de formulários, o angular dispõe de duas formas de se fazer essa associação de dados: os formulários digeridos ao modelo e os formulários reativos. O primeiro tipo é mais simples de implementar e permite uma associação de dados de 2 vias, entretanto não é fácil de testar. Os formulários reativos são mais flexíveis e lidam com cenários mais complexos. No desenvolvimento da aplicação proposta foram utilizados os dois tipos. Essa abordagem foi escolhida simplesmente para que o conhecimento de ambas fosse estudado e aplicado.

6 RESULTADOS

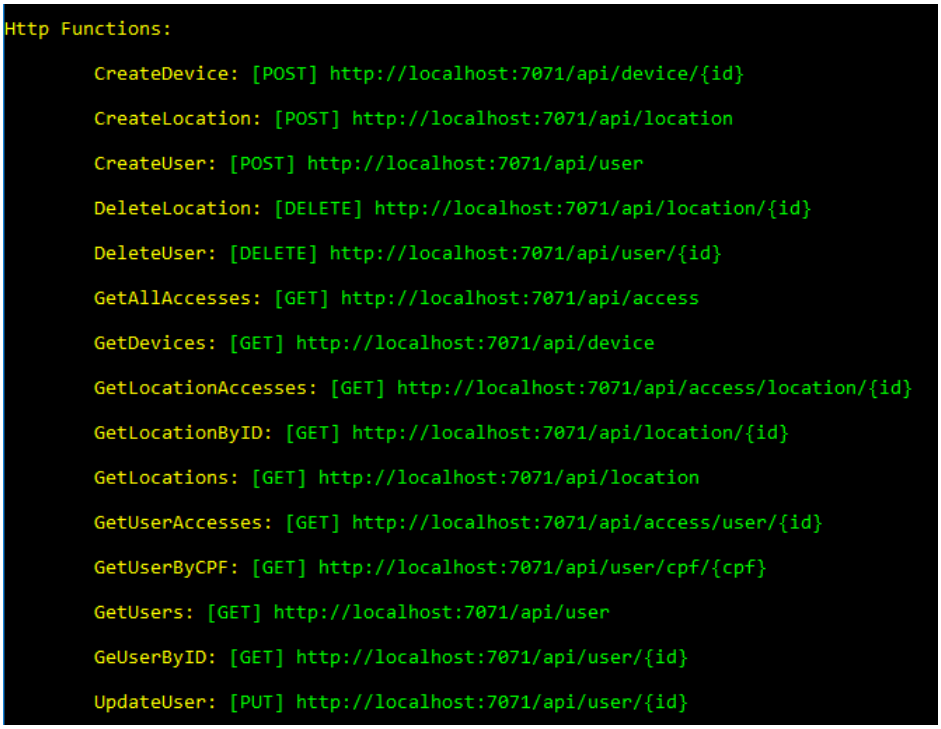
Neste capítulo são apresentados os resultados obtidos até o final do projeto e são discutidos alguns fatores que foram relevantes na obtenção destes resultados. Foram realizados, basicamente, três tipos de testes no projeto resultante. A API REST HTTP da aplicação sem-servidor foi testada isoladamente utilizando o Postman. Logo após foram feitos testes de integração da aplicação sem-servidor com o módulo de controle e com a aplicação web.

6.1 Resultados

- Testes da API REST

  A Figura 34 mostra todas as funções de gatilhos HTTP que foram criadas tantos com a arquitetura em camadas quanto com a arquitetura cebola. Percebe-se que foram implementadas quase todas as funcionalidades da especificação, a exceção do login de usuários.

Figura 34 – Funções de Gatilhos HTTP

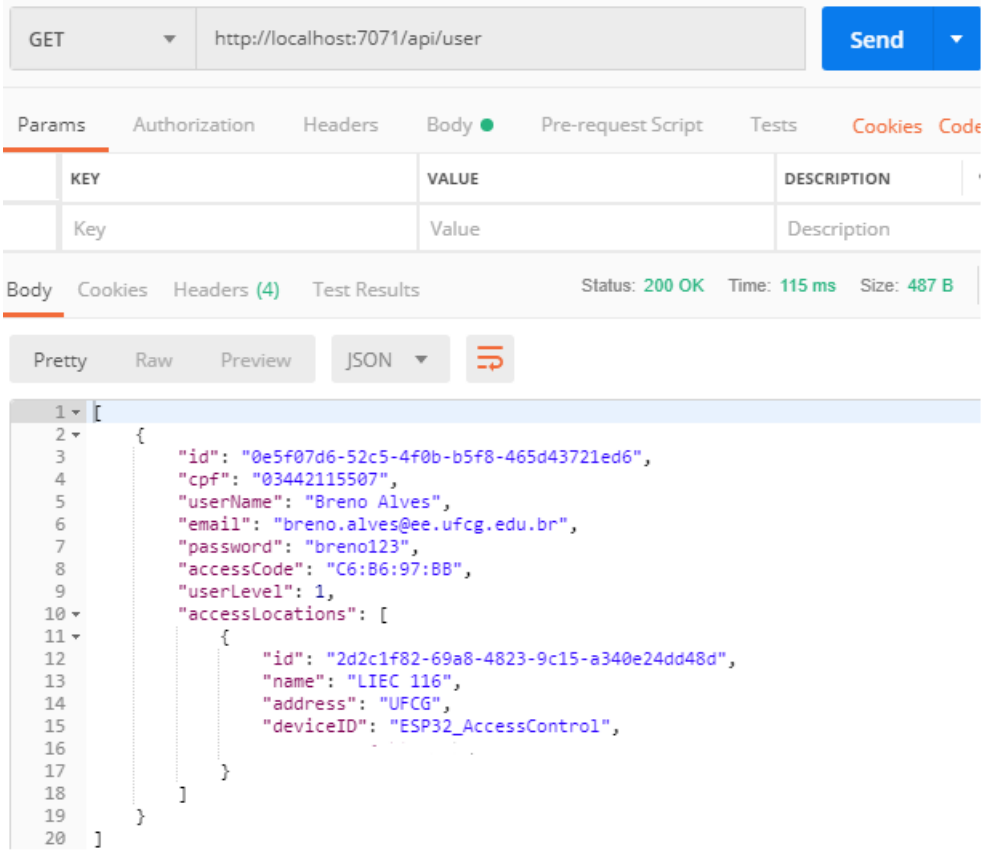


Fonte: Próprio autor.

Foram feitos testes para as duas arquiteturas realizando requisições HTTP em todas as funções utilizando o Postman e todos os testes tiveram resultados positivos. A

Figura 35 apresenta o resultado de uma requisição GET feita utilizando o Postman alguns desses testes.

Figura 35 – Testes da API REST



Fonte: Próprio autor.

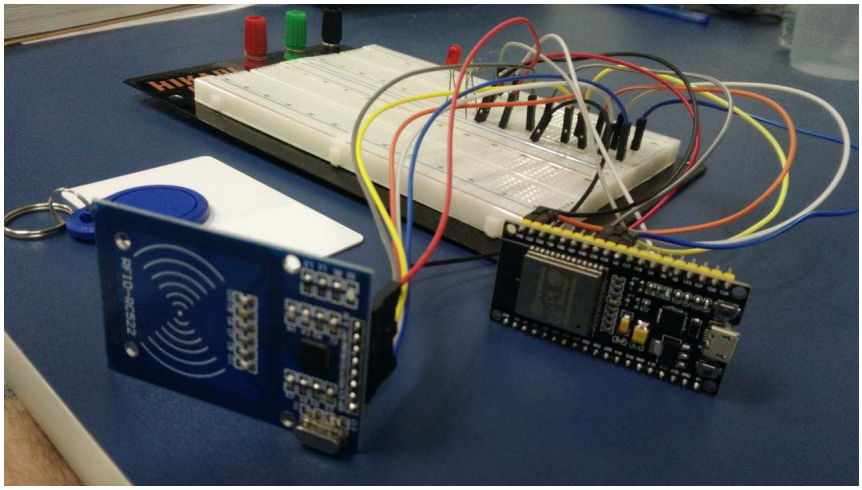
- Testes do Sistema de Autenticação

  Foi realizada a montagem de um módulo de controle em *protoboard*, uma foto dessa montagem encontra-se na Figura 36. Para testar o sistema de autenticação foi adicionado ao banco de dados um local com o dispositivo *ESP32_AccessControl* e um usuário com acesso ao local (tag C6:B6:97:BB). A tag 30:3E:7C:7A foi utilizada para testar o caso de um usuário que não tem acesso ao ambiente.

  Os testes foram realizados para ambas as arquiteturas da aplicação *serverless* e os dados analisados foram a resposta da requisição e o tempo. A Figura 37 ilustra o resultado das requisições para a arquitetura cebola. A arquitetura em camadas obteve o mesmo resultado, o que é comprovado pelo fato do tipo de arquitetura não interferir nesse tipo de teste.

  A Tabela 5 apresenta o resultado dos testes de tempo de resposta. Nesse caso a arquitetura em camada apresentou-se um pouco mais rápida, mas deve-se notar que a diferença é muito pequena quando comparada aos ganhos de se usar a arquitetura cebola.

Figura 36 – Montagem do Módulo de Controle em *Protoboard*



Fonte: Próprio autor.

Figura 37 – Log Resultante dos Testes de Autenticação

```
Scanned PICC's UID:
303E7C7A
Enviando Dados para nuvem
{"DeviceId":"ESP32_AccessControl", "UserAccessCode":"30:3E:7C:7A"}
Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmissio
Info: >>>Confirmation[12] received for message tracking id = 12 with res
Send Confirmation Callback finished.
Aguardando Resposta
Info: >>>Received Message [12], Size=6 Message Negado
Recebeu Resposta
Entrada Não Autorizada
```

```
Scanned PICC's UID:
C6B697BB
Enviando Dados para nuvem
{"DeviceId":"ESP32_AccessControl", "UserAccessCode":"C6:B6:97:BB"}
Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmissio
Info: >>>Confirmation[11] received for message tracking id = 11 with res
Send Confirmation Callback finished.
Aguardando Resposta
Info: >>>Received Message [11], Size=2 Message OK
Recebeu Resposta
Entrada Autorizada
```

Fonte: Próprio autor.

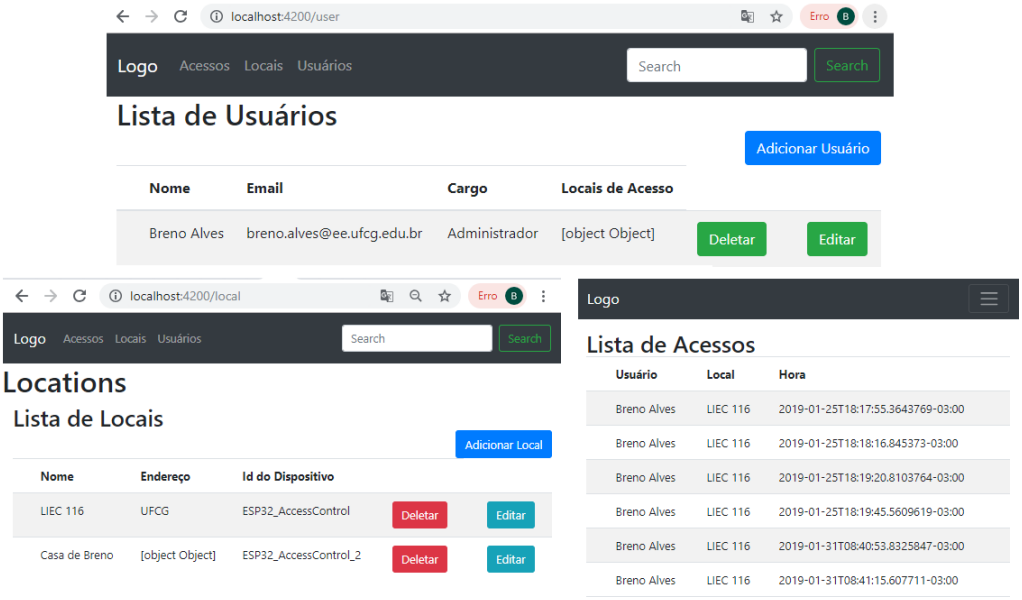
Tabela 5 – Testes de Tempo de Resposta

Arquitetura	**T1**	**T2**	**T3**	**T4**	**T5**	**Média**
Cebola	2,3s	2,3s	2,6s	1,87s	2,0s	**2,214**
Camadas	2,0s	2,2s	2,0s	1,7s	2,3s	**2,072**

- Testes da Integração com a Aplicação Web

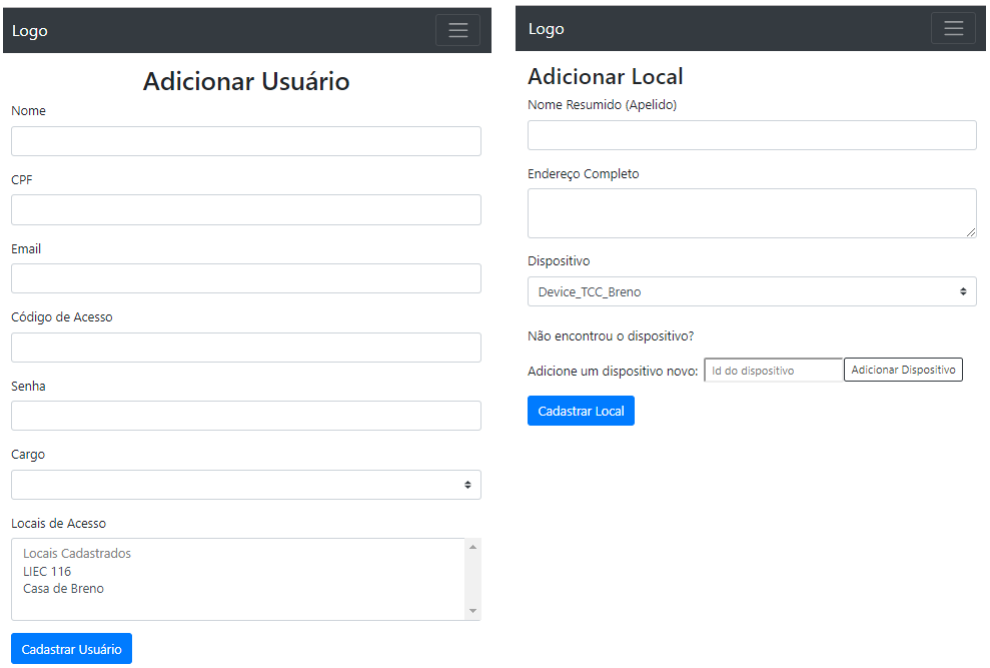
  As Figuras 38 e 39 apresentam as telas criadas para cada as listas e cadastros de informação. Devido ao tempo não foi possível implementar as telas para atualização de dados e a validação dos formulários. Entretanto, pode-se ver que a aplicação apresenta um *design* responsivo e funciona bem para as entradas corretas.

Figura 38 – Telas com Listas de Dados



Fonte: Próprio autor.

Figura 39 – Telas de Cadastro de Dados



Fonte: Próprio autor.

6.2 Discussões

Do ponto de vista de produto, os resultados obtidos deixam a desejar pois a especificação não pôde ser inteiramente cumprida. Entretanto, como um protótipo, o projeto é funcional e capaz de validar os conceitos de integração da internet das coisas com a computação em nuvem.

Ainda assim, alguns problemas foram observados nos serviços utilizados. Por exemplo, o SDK IoT da Azure utilizado para o desenvolvimento do firmware apresenta como falhas para iniciar a comunicação do microcontrolador com o *Azure IoT Hub* e um erro que leva o sistema a reiniciar sempre que uma requisição demora para ser feita.

Dentre os fatores que levaram ao não cumprimento da especificação, pode-se citar que o tempo levado para aprender e usar as tecnologias foi maior que o previsto e que a tentativa de utilizar boas práticas no desenvolvimento, como a mudança de arquitetura, acabou enfraquecendo a busca por mais resultados.

É importante informar que o sistema foi implantado no azure e testado de forma remota. Os resultados desses testes muito similares aos dos testes locais, tanto em assertividade quanto em performance.

7 CONCLUSÃO

Neste trabalho foi desenvolvido um protótipo de um sistema de controle de acesso por RFID com processamento em Nuvem. Os sistemas de controle de acesso são muito importantes e bastante difundidos mundialmente. Por isso, uma aplicação distribuída desse tipo de sistema apresenta um grande potencial de mercado.

O resultado obtido apresenta uma aplicação simples, mas capaz de validar as qualidades do sistema proposto. O projeto acompanha o estado da arte desse tipo de aplicação, utilizando as principais ferramentas e tecnologias presentes no mercado, como os serviços de funções da plataforma de nuvem do Azure, o *framerork* Angular e o chip ESP32.

Em virtude dos que foi apresentado, pode-se concluir que a utilização da Computação em Nuvem e da Internet das Coisas, auxiliados pelas boas práticas do desenvolvimento, resultaram em um sistema seguro, Inter operável, modular e facilmente escalável. Qualidades que são muito importantes nesse contexto globalizado de revolução da indústria.

Bibliografia

[1] KASTNER, W. et al. Communication systems for building automation and control. [S.l.: s.n.], 2005. 1178-1203 p. v. 93.

[2] HARPER R. Inside the Smart House, Springer-Verlag New York, Inc., Secaucus, NJ, 2003

[3] GIUSTO, D. et al. The Internet of Things, 1661 Springer, 2010.

[4] IDC. (Framingham, Massachusetts, EUA). 30 Billion Autonomous Devices By 2020. 2013. Disponível em: <https://securityledger.com/2013/10/idc-30-billion-autonomous-devices-by-2020/>. Acesso em: 20 set. 2018.

[5] BHIDE, Vishwajeet H. A survey on the smart homes using Internet of Things (IoT). International journal of advance research in computer science and management studies, [S.l.], 12 dez. 2014. v. 2.

[6] BOTTA, A. et al. Integration of cloud computing and internet of things: a survey. Future Gener. Comput. Syst., 2016.

[7] STOJKOSKA, Biljana L.Risteska; TRIVODALIEV, Kire V. A review of Internet of Things for smart home: challenges and solutions. Journal of Cleaner Production, [S.l.], 01 jan. 2017. v.140, Part 3, p. 1454-1464.

[8]Lopez Research LLC, "Uma introdução à Internet das Coisas (IoT)". 2013.

[9] Li, S., Xu, L., Wang, X., & Wang, J. Integration of hybrid wireless networks in cloud services oriented enterprise information systems. Enterprise Information Systems, 2012a, 165–187.

[10] Cisco, The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. 2011

[11] CARR, N. Big Switch: Rewiring the World,from Edison to Google. Norton & Company, 2008.

[12] BUYYA, R. YEO, C. VNUGOPAL, S. Market-oriented cloud 41 computing: Vision, hype, and reality for delivering it services as computing utilities. 2008.

[13] BORTOLI, F. Cloud pública e privada: quais as diferenças e como elas afetam o cliente?. 2016. Disponível em <https://computerworld.com.br/2016/08/25/cloud-publica-e-privada-quais-diferencas-e-como-elas-afetam-o-cliente/> Acesso em: 01 Fev. 2019.

[14] HURWITZ, J. KAUFMAN, M. HALPER, F. Cloud Services For Dummies.

IBM Limited Edition. 111 River Street: John Wiley & Sons, Inc, 2012.

[15] DIVAKARLA, U. KUMARI, G. An Overview Of Cloud Computing In Distributed Systems, AIP Conference Proceedings, 1324(1), 2010, pp. 184–186.

[16] BORGES, H. et al. Computação em nuvem. Brasil, 2011. 48 p.

[17] MICROSOFT, Tipos de Computaçãoem Nuvem. 21–. Disponível em <https://azure.microsoft.com/pt-br/overview/types-of-cloud-computing/> Acesso em: 01 Fev. 2019.

[18] AMAZON, Tipos de computação em nuvem. 21–. Disponível em <https://aws.amazon.com/pt/types-of-cloud-computing/> Acesso em: 01 Fev. 2019.

[19] FOWLER, M. Serverless. 2016. Disponível em: <https://www.martinfowler.com/bliki/Serverless.html> Acesso em: 01 Fev. 2018.

[20] MICROSOFT. Functions Overview. 2017 Disponível em: <https://docs.microsoft.com/pt-br/azure/azure-functions/functions-overviewl> Acesso em: 22 Jan. 2019.

[21] MICROSOFT. What is Azure IoT Hub?. 2018 Disponível em: <https://docs.microsoft.com/pt-br/azure/iot-hub/about-iot-hub> Acesso em: 22 Jan. 2018.

[22] MICROSOFT. Microsoft Azure IoT Architecture . 2018.

[23] MICROSOFT. Introduction to Azure Storage. 2019. Disponível em: <https://docs.microsoft.com/pt-br/azure/storage/common/storage-introduction> Acesso em: 22 Jan. 2019.

[24] DENIS, G. Microsoft Azure Storage Types Explained. 2017. Disponível em: <https://www.cloudberrylab.com/resources/blog/microsoft-azure-storage-types-explained/> Acesso em: 22 Jan. 2019.

[25] GAMMA, E. et al. Design patterns: elements of reusable object-oriented software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 1995.

[26] PRESSMAN, Roger S. MAXIM, Bruce R. Engenharia de software . Uma abordagem profissional. 8 ed. São Paulo: AMGH Editora Ltda, 2011.

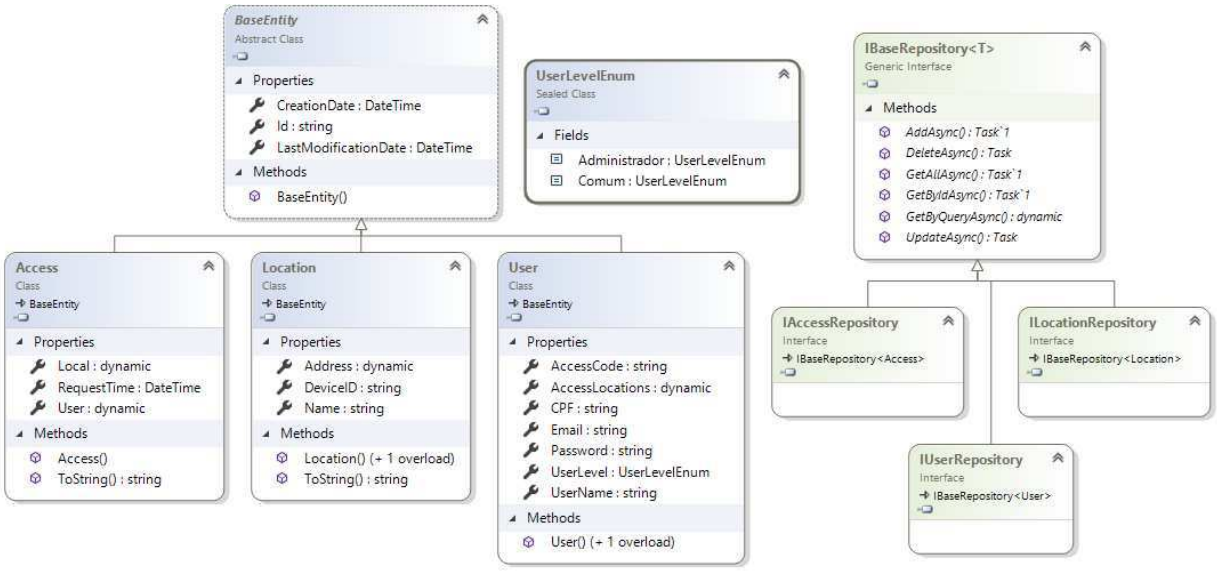
[27] ANGULAR. Architecture overview. 21–. Disponível em: <https://angular.io/guide/architecture> Acesso em: 25 Jan. 2019.

[28] KRUCHTEN, P. Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), 1995.

[29] SCHWABER K.; SUTHERLAND, J. The Scrum Guide, 2017.

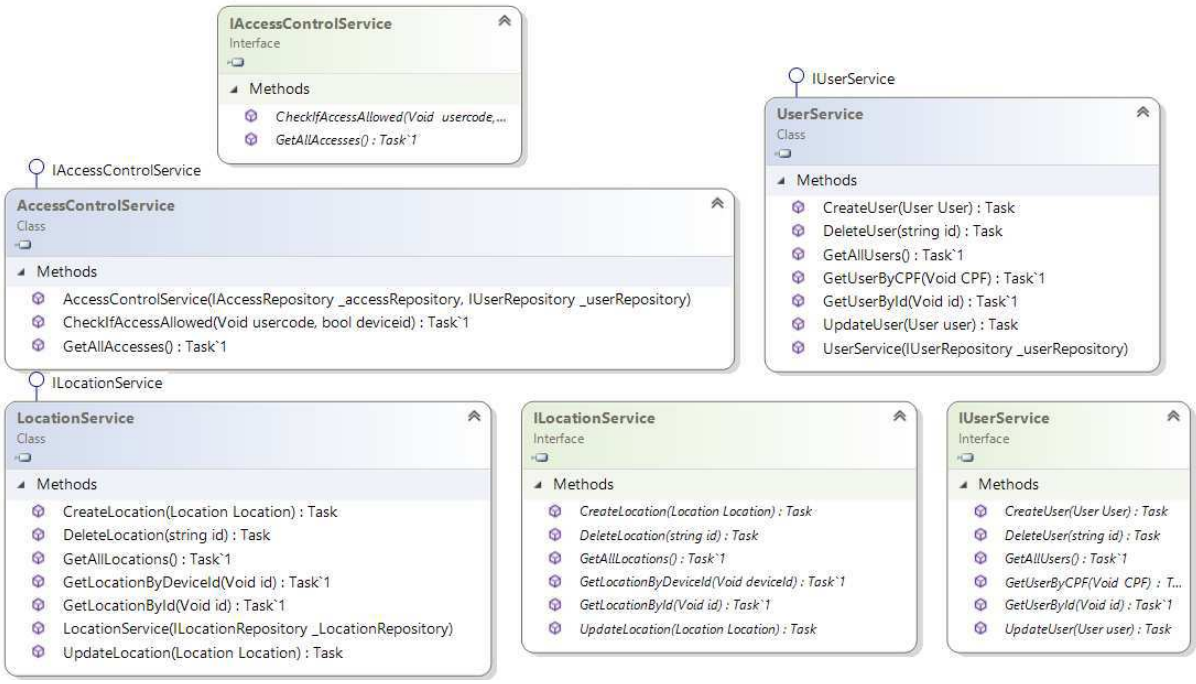
ANEXO A – Diagramas de Classes - Onion

Figura 40 – Domínio



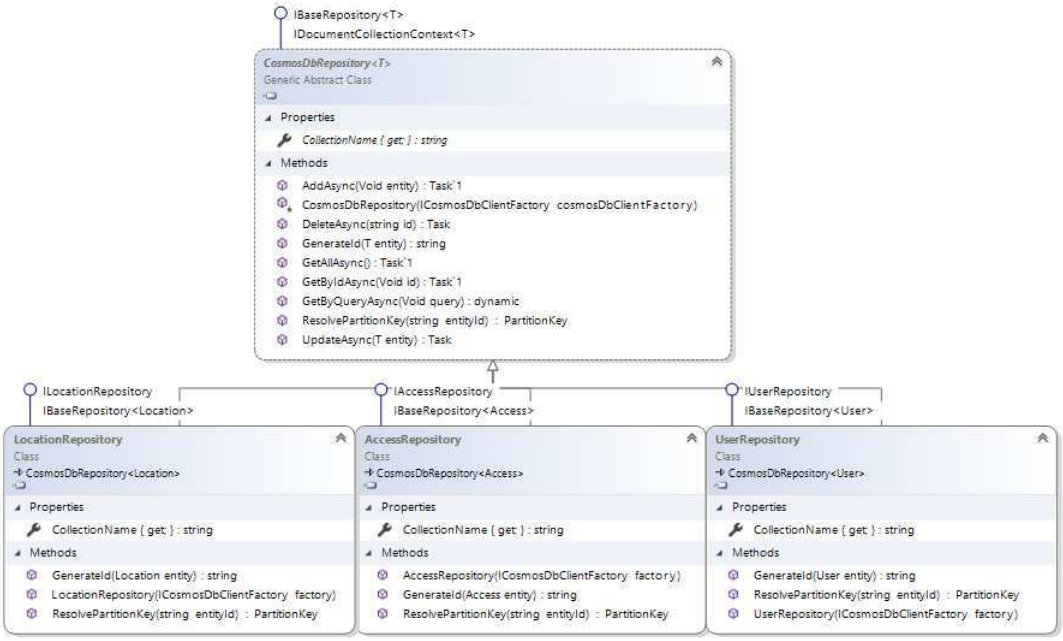
Fonte: Próprio autor.

Figura 41 – Serviços



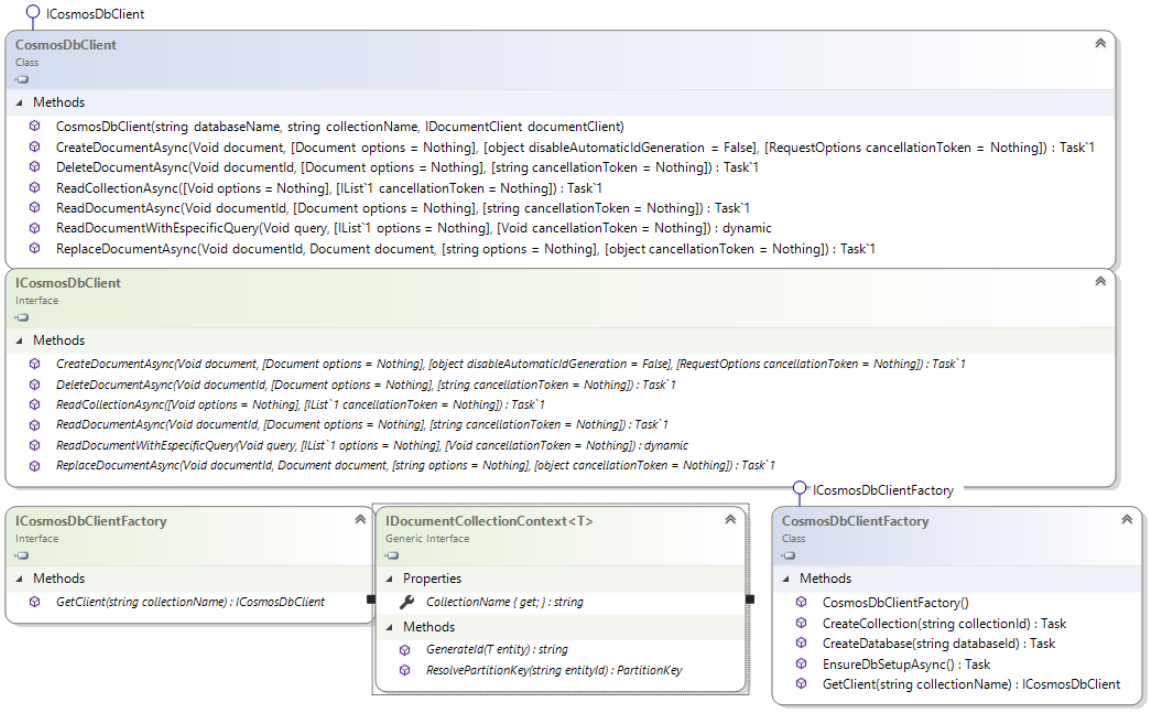
Fonte: Próprio autor.

Figura 42 – Infraestrutura - Repositorios



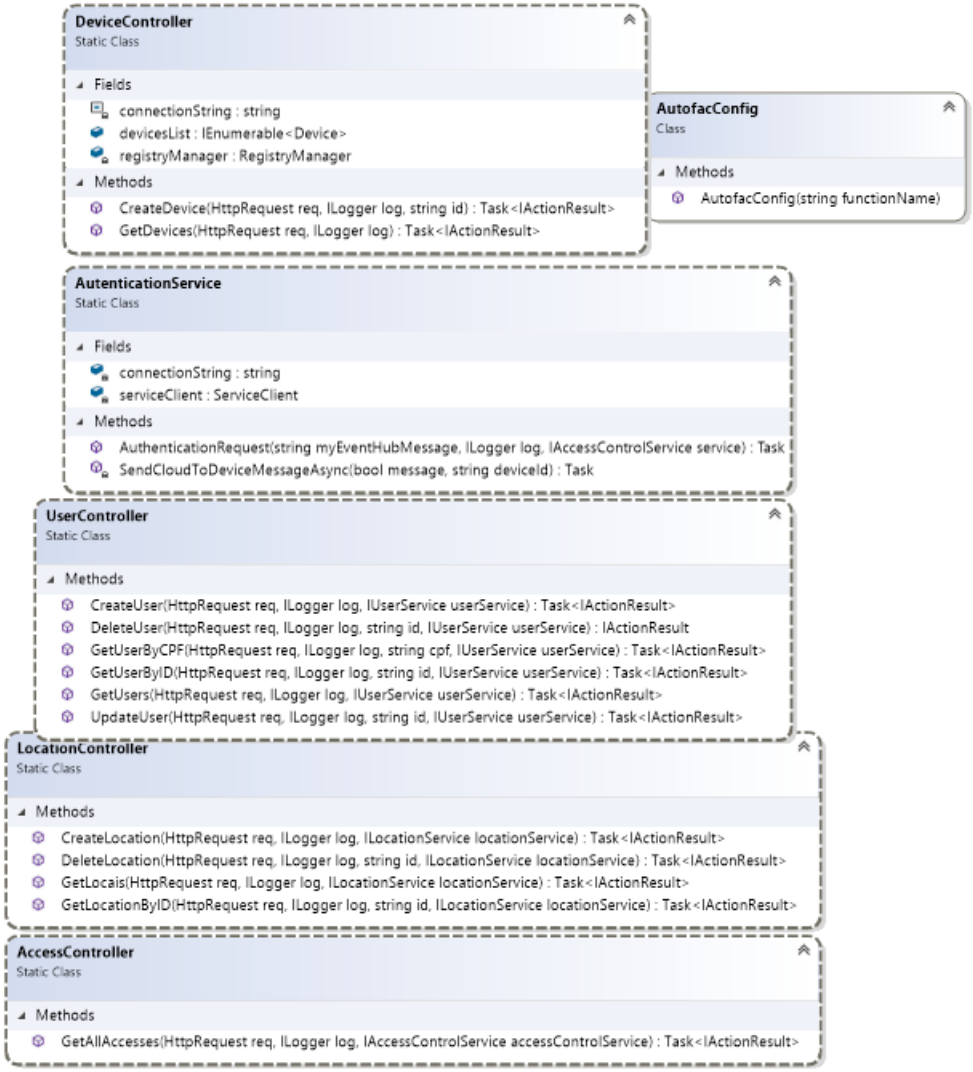
Fonte: Próprio autor.

Figura 43 – Infraestrutura - CosmosDB



Fonte: Próprio autor.

Figura 44 – Triggers



Fonte: Próprio autor.

ANEXO B – Código C do ESP-WROOM-32

```
      // Includes
//*****************************************************************************

#include <MFRC522.h>
#include <WiFi.h>
#include "AzureIotHub.h"
#include "Esp32MQTTClient.h"
#include "string.h"
#include <SPI.h>

// Defines
//*****************************************************************************
//#define RST_PIN         22
//#define SS_PIN          21
#define RST_PIN         3
#define SS_PIN          5
#define INTERVAL 10000
#define DEVICE_ID "ESP32_AccessControl"
#define MESSAGE_MAX_LEN 256

//Variaveis Globais
//*****************************************************************************
// Configuracao da rede
const char* ssid     = "LIEC_Wireless116";
const char* password = "Industrial_IoT";

// Azure ConnectionString - IOT HUB
//static const char* connectionString = "*"; //Secret
static const char* connectionString =  "*"; //Secret

const char *messageData = "{\"DeviceId\":\"%s\",
□□□□□□□□□□□□□\"UserAccessCode\":\"%02X:%02X:%02X:%02X\"}";

// Instancia MFRC522
MFRC522 mfrc522(SS_PIN, RST_PIN);

//flags
int messageCount = 1;
static bool hasWifi = false;
static bool messageSending = true;
char* response = "";
static bool hasIoTHub = false;
volatile uint32_t isrCounter = 0;
volatile uint32_t lastIsrAt = 0;
byte readCard[4];   // Stores scanned ID read from RFID Module

static bool waiting = 1;
static bool dadoEnviado = 0;
static uint8_t successRead = 0;
static bool recebeuResposta = 0;
static bool autenticado = 0;
```

```
//LEDS
const int LED_RED = 13;
const int LED_GREEN = 12;


//Setup
//*****************************************************************************
void setup() {

  //Inicializa a Serial
  Serial.begin(115200);

  //LEDS
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);

  // Setup MFRC522
  SPI.begin();        // Init SPI bus
  mfrc522.PCD_Init();    // Init MFRC522
  mfrc522.PCD_DumpVersionToSerial();  // Show details of PCD - MFRC522 Card Reader details

  // Initialize the WiFi module
  Serial.println("␣>␣WiFi");
  hasWifi = false;
  InitWifi();
  if (!hasWifi)
  {
    return;
  }
  // Initialize the aZURE Connection
  Serial.println("␣>␣IoT␣Hub");
  Esp32MQTTClient_Init((const uint8_t*)connectionString, true);
  Esp32MQTTClient_SetSendConfirmationCallback(SendConfirmationCallback);
  Esp32MQTTClient_SetMessageCallback(MessageCallback);
}

bool teste = 0;

void loop() {
    if(waiting)
    {
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, LOW);
        Serial.println("Aguardando␣RFID");
        // sets successRead to 1 when we get read from reader otherwise 0
        successRead = getID();
    }
    if(successRead)
    {
      Serial.println("Enviando␣Dados␣para␣nuvem");
      waiting = 0;
      bool teste = Send_AuthenticationRequest();
      if( teste ){
        // mensageVisor = (Aguardando resposta da Nuvem)
        dadoEnviado = 1;
      }
      else{
        // mensageVisor = (Erro ao comunicar)
        waiting = 1;
```

```
      }
        successRead = 0;
    }
    else if( dadoEnviado )
    {
        Serial.println("Aguardando Resposta");
        Esp32MQTTClient_Check();
    }
    if(recebeuResposta)
    {
      Serial.println("Recebeu Resposta");
       if(autenticado)
       {
          // visor Entrada Autorizada
          // aciona rele
          Serial.println("Entrada Autorizada");
          digitalWrite(LED_GREEN, HIGH);

      }
      else
      {
           // visor Entrada Nao Autorizada
          Serial.println("Entrada Nao Autorizada");
          digitalWrite(LED_RED, HIGH);
      }
      recebeuResposta = dadoEnviado = 0;
      waiting = 1;
    }

    delay(1000);
}

// ********************************************* FUNCOES ***********************************************

//////////////////////////////////////// Get PICC's UID ///////////////////////////////////
uint8_t getID() {
  // Getting ready for Reading PICCs
  if ( ! mfrc522.PICC_IsNewCardPresent()) { //If a new PICC placed to RFID reader continue
    return 0;
  }
  if ( ! mfrc522.PICC_ReadCardSerial()) {   //Since a PICC placed get Serial and continue
    return 0;
  }
  // There are Mifare PICCs which have 4 byte UID
  // Until we support 7 byte PICCs
  Serial.println(F("Scanned PICC's UID:"));
  for ( uint8_t i = 0; i < 4; i++) {  //
    readCard[i] = mfrc522.uid.uidByte[i];
    Serial.print(readCard[i], HEX);
  }
  Serial.println("");
  mfrc522.PICC_HaltA(); // Stop reading
  return 1;
}


static void InitWifi()
{
  Serial.println("Connecting...");
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  hasWifi = true;
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

static void SendConfirmationCallback(IOTHUB_CLIENT_CONFIRMATION_RESULT result)
{
  if (result == IOTHUB_CLIENT_CONFIRMATION_OK)
  {
    Serial.println("Send Confirmation Callback finished.");
  }
}

static void MessageCallback(const char* payLoad, int size)
{
  recebeuResposta = 1;
  if ( strcmp(payLoad,"OK") == 0 )
    autenticado = 1;
  else
    autenticado = 0;
}


static bool Send_AuthenticationRequest()
{
    char messagePayload[MESSAGE_MAX_LEN];
    snprintf(messagePayload,MESSAGE_MAX_LEN, messageData, DEVICE_ID, readCard[0],
                readCard[1],
                readCard[2],
                readCard[3]) ;
    Serial.println(messagePayload);
    EVENT_INSTANCE* message = Esp32MQTTClient_Event_Generate(messagePayload, MESSAGE);
    return Esp32MQTTClient_SendEventInstance(message);
}
```