UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Diogo Kenji Tsuruda

Proposta de Arquitetura baseada em Internet das Coisas: estudo de contexto, Modelo de Referência e desenvolvimento prático

São Carlos
2019

Diogo Kenji Tsuruda

Proposta de Arquitetura baseada em Internet das Coisas: estudo de contexto, Modelo de Referência e desenvolvimento prático

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Daniel Varela Magalhães

São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Tsuruda, Diogo Kenji
T882p Proposta de arquitetura baseada em internet das coisas: estudo de contexto, modelo de referência e desenvolvimento prático / Diogo Kenji Tsuruda; orientador Daniel Varela Magalhães. São Carlos, 2019.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. internet das coisas. 2. iot. 3. modelo de referência iot. 4. iot-a. 5. arquitetura iot. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Diogo Kenji Tsurudo

Título:
Proposta de arquitetura baseada em Internet das Coisas: estudo de contexto, modelo de referência e desenvolvimento prático

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professor DANIEL VARELA MAGALHÃES
(Orientador)

Nota atribuída: 10,0 (DEZ)

(assinatura)

Professor GLAUCO A. P. CAURIN

Nota atribuída: 10 (Dez)

(assinatura)

PROFESSOR ANDRÉ CANNONA HERNANDES

Nota atribuída: 10 (DEZ)

(assinatura)

Média: 10,0 (DEZ)

Resultado: APROVADO

Data: 18 / 11 / 2019.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador

DEDICATÓRIA

Dedico este trabalho à minha família, pelo imenso apoio que sempre forneceram, independente das escolhas feitas.

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais, Aurora e Mauro, pelo incentivo, encorajamento e suporte durante toda trajetória de minha vida, principalmente nos momentos de maior dúvida.
Às minhas irmãs Alessandra, Andressa e Renata, que sempre serviram como referência em meu desenvolvimento pessoal e profissional.
Ao meu orientador Prof. Dr. Daniel Varela Magalhães, por prover todo o suporte necessário para o desenvolvimento deste trabalho, assim como de outros projetos durante a graduação.
À todos os amigos que fizeram parte de uma jornada intensa, mas muito feliz.
À todo o corpo docente da Escola de Engenharia de São Carlos, por proverem uma educação de qualidade.
À todos os meus professores que fizeram parte da minha vida.

“Quantas chances desperdicei, quando o que eu mais queria era provar pra todo mundo que eu não precisava provar nada pra ninguém.”

Renato Russo (1986)

RESUMO

TSURUDA, D. K. **Proposta de Arquitetura baseada em Internet das Coisas:** estudo de contexto, Modelo de Referência e desenvolvimento prático. 2019. 116 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

Um dos conceitos de destaque no cenário atual é da Internet das Coisas, ou também conhecido como IoT. Existem grandes expectativas nas tecnologias que pertencem a esse domínio, como forma de impactar pessoas, indústrias, processos e mercados inteiros. Neste trabalho, busca-se entender o que esse conceito significa, quais as expectativas que se tem para o futuro próximo, como o atual momento está sendo impactado por isso. Em uma segunda parte, é realizado o estudo de um Modelos de Referência, denominado IoT-A, que apresenta conceitos e ferramentas para o desenho de soluções com arquitetura IoT. Esse modelo também auxilia na padronização de um linguajar comum para o entendimento de pessoas com perfis distintos. Em seguida, o modelo de IoT-A é utilizado no desenvolvimento de uma solução para envio, recebimento, armazenamento e visualização de dados proposto pelo autor como forma de aplicação dos conceitos. Nesta etapa, foram identificados as entidades envolvidas e todo o contexto de aplicação. A partir disso, foram definidos um Modelo de Domínio IoT, Visão Funcional com fluxo de informações e Visão de Comunicação. Com esse material, o funcionamento de cada parte do sistema foi especificado, assim como as tecnologias necessárias. Para comprovar o funcionamento como um todo em um tempo hábil, o escopo ganhou o caráter de PoC (*Proof of Concept*), cujo objetivo é testar apenas as funcionalidades da solução. Foi utilizado um dispositivo real, composto por um microcontrolador e um receptor GPS, que envia dados para um sistema *web*, por intermédio de um *Broker*, para efetuar a sua armazenagem segura em Banco de dados relacional. Como resultado, os objetivos foram atingidos com sucesso e a validade do Modelo de Referência foi comprovada.

Palavras-chave: Internet das Coisas. IoT. Modelo de Referência IoT. IoT-A. Arquitetura IoT.

ABSTRACT

TSURUDA, D. K. **IoT-based architecture proposal:** study of context, Reference Model and practical development. 2019. 116 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

One of the key concepts in the current scenario is the Internet of Things, or also known as IoT. There are high expectations in the technologies that pertain this domain as a way to impact people, industries, processes and entire markets. In this paper, we seek to understand what this concept means, what expectations we have for the near future, how the current moment is being impacted by it. In a second part, the study of a Reference Model, called IoT-A, is presented, which presents concepts and tools for the design of solutions with IoT architecture. This model also assists in standardizing a common language for better comprehension among people with different background. Then, the IoT-A model is used in the development of a solution for sending, receiving, storing and visualizing data proposed by the author as a way of applying the concepts. At this stage, the entities involved, and the entire application context were identified. From this point, an IoT Domain Model, a Functional View with information flow and Communication View were defined. With this material, the operation of each part of the system was specified as well as the required technologies. To prove the whole operation in a timely manner, the scope has gained the character of PoC (Proof of Concept), whose goal is to test only the functionality of the solution. A real device, consisting of a microcontroller and a GPS receiver, was used to send data to a web system via a Broker to store it safely in a relational database. As a result, the objectives were successfully met, and the validity of the Reference Model has been proven.

Keywords: Internet of Things. IoT. IoT Reference Model. IoT-A. IoT Architecture.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	-	*Application Programming Interface*
B2B	-	*Business to Business*
B2C	-	*Business to Consumer*
CRUD	-	*Create, Read, Update, Delete*
CSS	-	*Cascading Style Sheets*
E/S	-	Entrada/Saída
EF	-	Entidade Física
EV	-	Entidade Virtual
GF	-	Grupo Funcional
GIS	-	*Geographic Information System*
GPS	-	*Global Positioning System*
HTML	-	*Hypertext Markup Language*
HTTP	-	*Hypertext Transfer Protocol*
IIRA	-	*Industrial Internet Reference Architecture*
IoT	-	*Internet of Things*
IoT-A	-	*Internet of Things - Architecture*
IP	-	*Internet Protocol*
IPSO	-	*IP for Smart Objects*
ISO	-	*International Organization for Standardization*
M2M	-	*Machine to machine*
MQTT	-	*Message Queuing Telemetry Transport*
MVC	-	*Model-View-Controller*
NMEA	-	*National Marine Electronics Association*
PoC	-	*Proof of Concept*
QoS	-	*Quality of Service*
RAMI 4.0	-	*Reference Architecture Model Industrie 4.0*
RFID	-	*Radio-Frequency IDentification*
SQL	-	*Structured Query Language*
UART	-	*Universal Asynchrounous Receiver/Transmiter*
URL	-	*Uniform Resource Locator*

SUMÁRIO

Capítulo 1

Introdução

Com o avanço gradual da tecnologia e o advento da Internet, cada vez mais o mundo real se mistura com o digital para agregar mais valor no trabalho e modo de vida das pessoas, aumentando a produtividade, diminuindo custos, melhorando a experiência de uso e qualidade de vida das pessoas. A Internet das Coisas (Internet of Things ou IoT) começa a apresentar uma evolução, saindo do estágio inicial de desenvolvimento e primeiros casos de uso, para começar a mostrar resultados positivos em sua implantação, principalmente no mercado em que está inserido.

Entretanto, há ainda um vasto caminho que os envolvidos precisam desenvolver de modo que essa tecnologia alcance um cenário maduro, exercendo todo o potencial visualizado por seus idealizadores.

Segundo estudo realizado pela McKinsey Global Institute, o mercado de IoT tem potencial de gerar, até o ano de 2025, U$ 11 Trilhões somados em diferentes seguimentos do mercado, considerando relações B2B e B2C em nove categorias distintas (MANYIKA, 2015).

Desde então, o mercado tem mostrado um esforço crescente de empresas e negócios que utilizam dispositivos para compor soluções inovadoras, possibilitando novos serviços e modelos de negócio que, em última análise, sempre beneficia os consumidores finais da cadeia. É o que diz o estudo publicado pela Boston Consulting Group (BHATIA et al., 2019) acerca dos direcionamentos atuais das empresas que utilizam soluções IoT para inovar em produtos e serviços.

1.1 Motivação

Neste trabalho, o autor tem como motivação o estudo e implementação de uma arquitetura no contexto de Internet das Coisas. O tema foi escolhido pelo interesse nas novas

tecnologias, assim como na abrangência de situações e mercados onde o conceito de IoT tem aplicação.

Acredita-se que essa tecnologia terá participação fundamental na evolução dos processos e da qualidade de vida em geral e, portanto, é desejável ter domínio sobre os principais conceitos e atores envolvidos.

O autor considera que a associação dos conceitos estudados com a atividade prática de desenvolvimento da arquitetura é fundamental. Dessa maneira, a aprendizagem sobre o tema é ampliada, na medida que possibilita entender as dificuldades e tecnologias envolvidas no processo como um todo.

1.2 Objetivos

O presente trabalho se propõe a:

- Realizar um estudo bibliográfico do conceito de Internet das coisas, sobre as áreas envolvidas com o desenvolvimento, impactos e aplicações no mercado atual;
- Realizar um estudo de um Modelo de Referência IoT, utilizado para guiar projetos de desenho de arquiteturas;
- Propor e desenvolver um modelo de arquitetura de uma solução IoT aplicada num contexto determinado;
- Expor algumas tecnologias e protocolos utilizados na prática.

1.3 Organização do trabalho

Este trabalho se organiza da seguinte forma:

- Capítulo 2 – Revisão Bibliográfica - apresenta um contexto teórico e abrangente sobre o ecossistema de Internet das Coisas. Também é realizado um estudo do Modelo de Referência IoT-A para o projeto de uma arquitetura IoT;

- Capítulo 3 – Projeto de Arquitetura na prática – apresenta uma situação de aplicação das tecnologias estudadas, gerando material para o desenvolvimento prático da solução proposta;
- Capítulo 4 – Desenvolvimento da Solução – apresenta as tecnologias utilizadas, assim como a lógica dos sistemas e programas desenvolvidos;
- Capítulo 5 – Resultados e Discussões – apresenta os resultados e discute os pontos observados em cada parte do trabalho;
- Capítulo 6 – Conclusão – conclui o trabalho e indica os próximos passos para evolução;
- Capítulos 7 – Referências Bibliográficas;
- Capítulos 8 a 10 – Apêndices - transcreve os códigos utilizados em cada componente.

Capítulo 2

Revisão Bibliográfica

2.1 O conceito de Internet das Coisas

Para se definir o conceito de “Internet das Coisas”, é necessário levar em conta a construção de várias visões que diversos atores e instituições influenciaram no decorrer do tempo. As diferenças entre essas visões, às vezes significativas, ocorrem devido à interesses, finalidades e contextos distintos que os interessados, companhias, pesquisadores e órgãos de padronização possuíam ao construírem novas tecnologias, modelos e padrões assumidos por seus interessados.

O termo Internet das Coisas, atualmente bem difundido, foi cunhado em um momento de divulgação da tecnologia de identificação por Radiofrequência (RFID) pela Auto-ID Labs. O seu conteúdo dá margem à diversas interpretações de definição (ATZORI et al., 2010).

Segundo Atzori (2010), a primeira advém do viés da conectividade, orientado pela palavra “Internet”, que é interpretada não somente como a conectividade entre dispositivos e sistemas, mas entre qualquer coisa, em qualquer lugar, a qualquer momento. Nesse segmento, há um grande esforço que promove a adoção de padrões definidos pela IPSO (IP for Smart Objects) Alliance, atualmente com o nome de OMASpecWorks, que trabalha para a adoção do padrão IP no contexto de IoT.

A segunda visão, orientada pelo viés das Coisas, vai além da tecnologia inicial de RFID que possibilita identificar e endereçar cada dispositivo da rede individualmente ao longo do tempo, considera que será possível criar soluções inteligentes, caracterizadas pelo alto grau de automação e autonomia na interoperabilidade entre coisas e sistemas. Representam a interface entre os mundos físico e digital de objetos que terão mais capacidade de tomar decisões pró-ativas, ou seja, sem interferência humana.

Por fim, a terceira, que leva em consideração a interpretação dos dados gerados pelas coisas, que, segundo estudos e expectativas do mercado, serão da ordem de dezenas de bilhões de dispositivos conectados até 2025 (MANYIKA, 2015).

Para extrair o real valor da conectividade, é necessária uma infraestrutura capaz de armazenar e processar altas cargas de dados em um período de tempo aceitável. Era considerado a adoção do conceito de "*Semantic Web"* (ATZORI et al., 2010), que determina um padrão de comunicação em que máquinas conseguem interpretar, de forma cognitiva, o conteúdo da internet através de uma complexa rede de metadados. Este conceito não obteve a adoção esperada, dando espaço para a integração da área de *"Big Data Analytics"* (AHMED et al., 2017), tornando possível realizar a coleta de dados de diversas fontes e variados formatos de forma integrada.

2.2 Aplicação

Segundo Manyika (2015) as possibilidades de desenvolvimento do ecossistema IoT pode ser dividido em 9 categorias de aplicação, que são brevemente comentadas abaixo:

- **Humano:** duas configurações compõem a categoria "Humano", uma é a linha de aplicações para saúde e condição física, a outra linha é relacionado à produtividade humana em ambientes de trabalho. A utilização de dispositivos vestíveis ou inseridos no corpo humano podem monitorar constantemente as condições físicas, auxiliando no tratamento de doenças crônicas, aumentando a aderência às terapias prescritas. Na segunda linha, um exemplo de aplicação seria a implantação de realidade aumentada em ambientes produtivos que auxiliem o trabalhador na execução de atividades, consulta de informações e procedimentos corretos, entre outras possibilidades;
- **Casa:** representam uma maior interação nos ambientes domésticos, com a aplicação de sensores para monitoramento, soluções inteligentes que possibilitem ações automáticas, como exemplo a limpeza automática, ou ações de segurança do lar e questões ergonômicas que podem evitar lesões em atividades cotidianas;
- **Varejo:** aplicações que auxiliam no processo de compra do consumidor no Varejo, como sistemas de "self-checkout", ofertas personalizadas, otimização do estoque e catálogo;

- **Escritórios:** representa mudanças em ambientes cuja produção de conhecimento é a atividade primária. As aplicações ocorrem em sistemas para gerenciamento de energia, sistemas de segurança inteligentes, soluções para monitoramento ativo via câmeras, realidade aumentada aplicada em treinamentos;
- **Fábricas:** uma das maiores áreas de impacto das tecnologias IoT, considera ambientes de produção repetitiva, incluindo hospitais e fazendas, que representam uma alta repetitividade nas atividades, além das atividades de manufatura. As aplicações resultam em aumento de produtividade, eficiência e otimização de processos, manutenção preditiva de equipamentos, melhor gerenciamento de energia, segurança do trabalho e saúde;
- **Sítios de Trabalhos de Risco:** Ambientes de trabalho que apresentam grau de periculosidade, inclui indústrias de mineração, óleo e gás, construções, entre outros. As aplicações incluem manutenção preditiva de equipamentos, aumento de segurança e qualidade do trabalho;
- **Veículos:** são considerados todos tipos de locomoção como automóveis, caminhões, navios, aviões, trens, entre outros. As aplicações incluem manutenções baseadas no uso, projetos baseados em informações coletadas de usuários, análise para pré-vendas, ofertas personalizadas de serviços;
- **Cidades:** aplicações em ambientes urbanos que melhorem os tráfegos, monitoramento do meio ambiente, gerenciamento de recursos, soluções inteligentes de monitoramento;
- **Outros:** inclui aplicações que se relacionam com outras categorias, mas não se encaixam exclusivamente em uma delas. As aplicações são em ferrovias, veículos autônomos, rastreamento de carga, navegação de voos, roteamento em tempo real.

No atual cenário de implementação das tecnologias IoT, as empresas pioneiras já adquiriram alguma experiência com a implantação de projetos IoT e passaram da fase de

“euforia” causado no começo da década. Essas empresas entenderam que para o sucesso dos projetos, apenas a adoção das novas tecnologias não é suficiente, é essencial que haja a definição de estratégias e processos de negócio bem elaborados. (BHATIA et al., 2019)

A tecnologia IoT está cada vez mais relacionada com outras tecnologias emergentes e promissoras como algoritmos de aprendizado de máquina (*Machine Learning*) para interpretar os dados gerados, Realidade Virtual e/ou Aumentada para projetar visualizações dos dados em sobreposição aos ambientes físicos, e *Blockchain*, que é utilizado para a segurança na transmissão de dados. (BHATIA et al., 2019)

As experiências no mercado possibilitaram o surgimento de novos modelos de negócio que se utilizam das soluções desenvolvidas, conecta setores distintos de forma inovadora e ofertam soluções com maior valor agregado para o consumidor. De forma crescente, o desenvolvimento deixa de estar com foco em soluções técnicas particulares e passa a atuar com maior intensidade em soluções de geração e fluxo de informações, como demonstrado nos casos de uso abaixo. (BHATIA et al., 2019)

Entre os exemplos, alguns citados:

- **Medtronic**: utiliza dispositivos para medição de glicose de forma contínua juntamente com bombas de insulina. A solução possibilita o monitoramento em tempo real do paciente. A solução possibilitou à empresa ofertar um serviço de envio de dados diretamente para o médico do paciente, auxiliando o tratamento de mais de 95.000 pessoas.
- **Volkswagen**: na Europa, a empresa desenvolve uma solução de aquisição de dados meteorológicos dos veículos operantes. Esses dados serão compartilhados com a TenneT, empresa de transmissão de energia elétrica, desta forma, possivelmente haverá redução de custos e otimização de operação devido à melhor predição de condições e oferta de energias solar e eólica.
- **Metromile**: oferta um programa de pagamento de seguro por milha para motoristas que não dirigem com frequência.

Figura 1 - Dispositivo IoT que mede a distância percorrida pelo veículo

Fonte: https://www.metromile.com/

2.3 Modelos de Referência e Arquiteturas IoT

Como apontado por Weyrich (2015), Aplicações IoT tem sido baseados em softwares fragmentados e sistemas distintos, para casos de usos também específicos. Isso se evidencia com a quantidade de áreas que aplicam projetos de IoT e exigem escopos que variam em grande proporção entre si, utilizam sistemas totalmente distintos para finalidades diversas. Ocorre, então, uma dificuldade de comunicação entre cada solução, de forma que cada uma seja limitada pela arquitetura que foi desenvolvida para atuar. Fica difícil, se não impossível, que haja uma interoperabilidade entre áreas citadas anteriormente. Um exemplo seria a integração de soluções de áreas que envolvam as categorias Humano e Cidades.

Desta maneira, torna-se necessário a elaboração de modelos que apresentam, até certo modo, uma padronização de arquitetura de forma abrangente, um consenso de conceitos e termos que facilitam à comunicação, e por extensão promovem maior agilidade na implantação de projetos relacionados.

Uma arquitetura de referência contém alguns requisitos básicos de funcionamento (WEYRICH, 2015):

- Conexão e Comunicação que podem ocorrer entre dispositivos ou numa rede de múltiplos agentes;

- Gerenciamento de Dispositivos deve ser aplicado a todos elementos da rede, desde inclusão de um novo agente até mudanças de configuração, as informações devem ser propagadas paras os dispositivos;
- Extração, análise e atuação em dados, de forma a gerar informações e conhecimentos relevantes nos serviços;
- Escalabilidade é importante para tratar volumes crescentes de informações, em diferentes ambientes de instalação;
- Segurança são necessários para prover privacidade e confiabilidade em todos os aspectos da estrutura IoT.

Ainda neste estudo, são apresentados três Arquiteturas de Referência de maior importância no cenário atual, numa visão superficial sobre suas características. A primeira, *Reference Architecture Model Industrie 4.0* (RAMI 4.0) estabelece guias para a Indústria 4.0 e vai além de aspectos sobre IoT, envolve também conceitos de manufatura e logística em suas definições. As próximas duas estabelecem, com uma visão mais prática, os guias focados somente em IoT, que são a *Industrial Internet Reference Architecture* (IIRA) e *Internet of Things – Architecture* (IoT-A), com aquela apresentando menos tempo de detalhamento pela comunidade e, portanto, menor maturidade quanto ao Modelo IoT-A.

Assim, neste trabalho será apresentado uma análise mais específica sobre o Modelo IoT-A para um entendimento generalizado sobre o assunto.

2.4 IoT-A

Esta seção se propõe a apresentar os principais conceitos sobre o projeto IoT-A (BAUER et al., 2011), detalhando alguns modelos apresentados no documento que serão utilizados em capítulos seguintes para o desenho de uma arquitetura IoT.

O IoT-A foi um projeto realizado no período de 2010 a 2013, financiado pela União Europeia, que envolveu mais de 50 pesquisadores e cientistas para desenvolver um Modelo de Referência de Arquitetura para a Internet das Coisas.

Como Modelo de Referência, o objetivo é de unificar o entendimento em um *framework* abstrato acerca de conceitos, axiomas e relações entre as entidades envolvidas no contexto de IoT. Desta maneira, o material auxiliará no projeto de novas arquiteturas na medida que possibilita uma integração entre outras arquiteturas e soluções existentes, saindo de um modelo de "INTRAnet das Coisas" para "INTERnet das Coisas" (BAUER et al., 2011), que ocorre na grande maioria de soluções empregadas pelo fato de atenderem somente problemas específicos e não as conectar. Uma das intenções é de que seja possível aplicar os mesmos conceitos em ambientes distintos, facilitando a interoperabilidade entre soluções, independentemente dos sistemas e tecnologias utilizados. Desta forma, o verdadeiro valor da aplicação de Internet das Coisas seria alcançado.

É importante citar que os modelos utilizados e a arquitetura elaborada não partiram de um ponto zero de desenvolvimento. Foram utilizados diversos modelos de arquiteturas considerados "estado da arte" para formar a base do Modelo de Referência de Arquitetura IoT.

A finalidade do trabalho, entretanto, não é de detalhar toda a estrutura formulada pela comissão europeia, uma vez que é bem extensa e complexa, mas de mostrar os principais conceitos. Será aplicado alguns modelos descritos para o desenvolvimento de uma arquitetura básica própria para exemplificação e, posteriormente, desenvolvimento.

2.4.1 Conceitos Gerais

O material estudado compõe um guia de referência baseado em diversos modelos advindos de outras áreas do conhecimento, aproveitando de conceitos já estabelecidos e melhores práticas observadas nos seus respectivos campos. Uma delas é de gerar a arquitetura como uma composição de "Visões" separadas em várias finalidades específicas. Por exemplo, trata-se da Visão de Contexto, Visão Funcional, Visão da Informação, Visão de Lançamento (do termo *Deployment*, amplamente utilizado no contexto de desenvolvimento de sistemas), Visão de Operação, entre outras que podem ser relevantes para o projeto.

O conceito de "Visões" facilita o entendimento dos envolvidos, principalmente os interessados no projeto (*stakeholders*), na medida que não excede a quantidade de informações mostradas em uma única ferramenta de visualização. No caso de demonstração em um bloco monolítico, que concentra todas as informações em um só lugar, além de gerar confusão no entendimento, poderia gerar complicações adicionais no fato de possivelmente apresentar informações desnecessárias ou sigilosas para pessoas que tem apenas uma ação parcial no projeto.

Além das visões, há também especificações para análises qualitativas de requisitos do projeto, definidas como "Perspectivas", que avaliam características não-funcionais como Segurança, Desempenho e Adaptabilidade. Essas especificações também detalham etapas de definição de requisitos e restrições, análise de ameaças e riscos, e por fim, oferecem guias para auxiliar na decisão de escolhas da arquitetura.

O Modelo de Referência IoT não especifica uma metodologia particular para ser empregada, pois essa atividade depende muito do contexto em que é aplicada, das estruturas organizacionais do time envolvido, de padrões internacionais adotados ou acordos que precisam se conformar. A sua aplicação é abrangente e faz parte de qualquer processo de construção de uma arquitetura.

Neste trabalho, será apresentado alguns modelos utilizados para gerar as "Visões" propostas por IoT-A. A partir destas definições, esses conceitos serão aplicados para gerar uma arquitetura básica que servirá de base para o desenvolvimento de um sistema.

2.4.2 Modelo de Domínio

Uma importante parte do Modelo de Referência proposto pela Comissão do projeto IoT-A é o Modelo de Domínio. Esse modelo estabelece uma estrutura de descrição dos conceitos principais, além de apresentar as suas inter-relações e responsabilidades.

O propósito principal desse modelo é gerar uma compreensão comum do domínio em questão, desta forma, existe uma maior facilidade no momento de discutir ideias de soluções e suas respectivas arquiteturas. Uma das características desse modelo é a utilização de conceitos abstratos, pois há alguns que existem na maioria dos projetos e podem ser reutilizados, mesmo que as áreas de aplicação sejam bem distintas. A definição de termos comuns também facilita o entendimento, pois estabelece uma linguagem que pode ser mais bem compreendida pelos interessados ao projeto. Há também o fato de que com a utilização da abstração, a arquitetura independe das tecnologias empregadas e, assim, não há um engessamento da estrutura logo no início do desenvolvimento.

Para demonstrar a estrutura de um Modelo de Domínio IoT é utilizado o diagrama em linguagem UML. Os seus conceitos poderão ser consultados em https://www.tutorialspoint.com/uml/.

A estruturação de um modelo de domínio começa com a identificação das “Entidades Físicas” (EF), que são qualquer objeto ou corpo físico que é relevante do ponto de vista do usuário ou aplicação. Para cada EF existe uma “Entidade Virtual” (EV) correspondente, que representa a EF no ambiente digital, pode ser representado como apenas um registro de dados, como também pode haver modelos 3D, objetos (ou instâncias de uma classe, em uma linguagem de programação orientada a objetos), avatares, até uma conta de usuário em uma rede social, pois representa um usuário humano. Outro conceito usual é o de “Artefatos Digitais” que é empregado para representar software, serviços, aplicações, entre outros casos. Vale expor que EV também são artefatos digitais.

Um dispositivo se encontra na intersecção entre EF e EV. É a interface que atua diretamente sobre o ambiente físico, podendo ser por meio de atuadores ou capturar informações deste ambiente através de sensores e transformar em dados digitais.

Outros conceitos utilizados são de “Recursos” e “Serviços”, segundo Bauer et al. (2013), o primeiro se trata de componentes de software que oferecem alguma funcionalidade

no sistema e podem funcionar por meio de Serviços, o último está relacionado com a função realizada, no mesmo conceito de um trabalho prestado por um consultor de negócios, por exemplo, mas neste texto “Serviços” assume um papel de serviço de software que abstrai a operação de uma funcionalidade específica e entrega um resultado, sem que seja necessário o conhecimento específico do componente operante.

Também faz parte do Modelo de domínio IoT a representação das relações, dependências e interações entre o sistema e entidades externas com as quais há interação. (ROZANSKI; WOODS, 2012).

2.4.3 Modelo de Informação

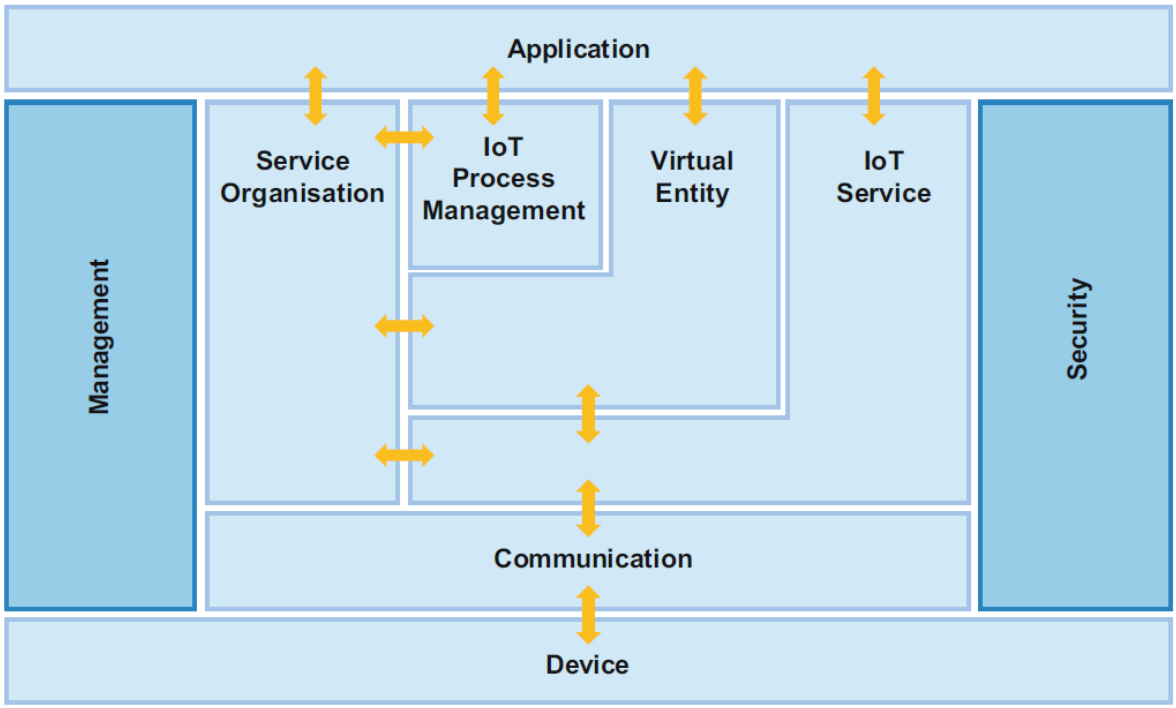
O modelo de informação IoT define a estrutura (Ex: relações, atributos, serviços) de toda a informação para as Entidades Virtuais (EV) em um nível conceitual. Deste modo, as informações-chave de todo o Sistema IoT podem ser representadas e gerenciadas.

A relação existente com o Modelo de Domínio estabelece que o Modelo de Informação represente todos os conceitos do primeiro que permeiam o meio digital. Essa estrutura fornece as bases de como o sistema lidará com as atividades de representar, reunir, processar, armazenar e devolver as informações. Além disso, é utilizado para definir as interfaces entre componentes da Arquitetura.

2.4.4 Modelo Funcional

A estrutura de um projeto IoT pode ser decomposta em diversos Grupos Funcionais (GF), essa atividade separa por meio de blocos semânticos, as funcionalidades que são necessários na operação do sistema. O Modelo Funcional auxilia com a utilização de um *framework* para o desenvolvimento da “Visão Funcional” e, além da diferenciação das funções, estabelece também a relação entre os blocos.

Figura 2 - Diagrama do Modelo Funcional



Fonte: Bauer et al. (2013).

A Figura 2 mostra o *framework* do Modelo Funcional. Nele, existem 7 blocos em azul claro dispostos na horizontal, e dois blocos em azul escuro dispostos na vertical. Cada bloco representa um GF e as setas representam as relações que possuem entre si.

Os GFs de "Gerenciamento" e "Segurança" proveem funcionalidades que são necessárias para todos os outros GFs em azul claro. O primeiro estabelece funcionalidades relacionados à governança do sistema IoT, provendo aspectos de redução de custos, flexibilidade, gerenciamento de falhas e atendimento na ocorrência de eventos inesperados.

O GF "Segurança" é responsável pela segurança e privacidade envolvidos em todo o sistema IoT. Atende aspectos de autenticação e autorização de usuários, proteção de parâmetros privados, troca de informações confidenciais protegidos por criptografias, entre outros aspectos.

Como o próprio nome sugere, o GF "Dispositivo" representa a ponta que está interagindo diretamente com o meio físico. Assim como "Comunicação", que abstrai as várias formas possíveis de interação e fluxo de informações. Há também o GF "Entidade Virtual", já comentado anteriormente, que modela as Entidades Físicas no meio virtual, e o

GF “Aplicação” relacionado com a finalidade de aplicação do Sistema IoT, envolvendo interações com o usuário ou processo de negócio integrado com o sistema.

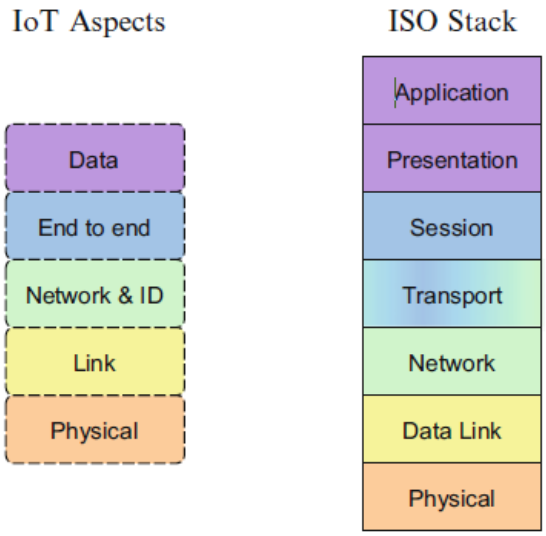
O GF “Serviços IoT” contém os Serviços disponíveis pelos Recursos do Sistema, mais relacionados com baixo nível de abstração (Ex: leitura de um sensor XYZ), gerenciados pelo GF “Organização de Serviços”, que faz uma composição e orquestração desses Serviços para agregar uma solução de alto nível. Um exemplo seria a existência de um modelo que depende da leitura de sensores de temperatura e humidade para determinação de funcionamento de um ar-condicionado, neste caso, há uma necessidade de composição da informação provinda de dois Serviços distintos para o controle efetivo do aparelho. Além desse aspecto, “Organização de Serviços” também está fortemente relacionado com o GF “Gerenciamento de Processos IoT”, este estabelece uma integração com processos externos ao sistema IoT e utilizam seus serviços para compor alguma finalidade específica. Desta maneira, o sistema IoT pode ser integrado com sistemas externos (Ex: integração com um ERP de uma empresa) dentro de etapas em processos de negócio definidos.

2.4.5 Modelo de Comunicação

O Modelo de Comunicação fornece os principais paradigmas para conexão entre elementos da arquitetura IoT. Como base, é utilizado o Modelo OSI de 7 camadas, estabelecido como padrão ISO (*International Organisation for Standardization*) para comunicação em estruturas de redes.

Com a adaptação das definições do Modelo OSI, a comissão do IoT-A propôs um modelo que considera os pontos necessários para projetos IoT, divididos em cinco (5) camadas.

Figura 3 - Modelo de Comunicação IoT (à esquerda) em comparação com Modelo OSI (à direita)



Fonte: Bauer et al. (2013).

Cada camada é relacionada com diferentes níveis da tecnologia e protocolos utilizados, cada qual com a sua finalidade e contexto de aplicação, que será descrito abaixo:

- **Física:** representa a comunicação dos elementos de mais baixo nível de abstração nos sistemas. Define as características físicas e elétricas da rede, operando no nível de troca de bits.
- **Link:** representa a comunicação entre componentes de um dispositivo ou sistema, numa camada governada pelo endereçamento via endereço MAC, definido pelo fabricante dos dispositivos.
- **Rede e ID:** esta camada combina os aspectos da camada *Network* do modelo OSI, adicionando aspectos de Identificação para Artefatos Digitais. Neste nível, a comunicação e interoperabilidade entre dispositivos deve ocorrer de forma padronizada e independente da solução de Rede utilizada.
- **Fim-a-Fim:** esta camada está relacionada com a confiabilidade, aspectos de transporte, funcionalidades de tradução, suporte de *gateways/proxies* e parâmetros de configuração quando a comunicação ocorre entre Redes diferentes.
- **Dados:** está é a camada de maior alto nível de abstração, tem relação com a transferência de dados entre dois atores na arquitetura IoT. Tem estrutura de dados mais complexa, provendo uma estrutura de atributos para a descrição dos dados.

Pode ser traduzida de um protocolo para outro (Ex: CoAP para HTTP, IPv4 para IPv6, entre outros).

Capítulo 3

Projeto de Arquitetura na prática

3.1 Contexto do Projeto

Para aplicar os conceitos estudados, deseja-se criar um modelo genérico de arquitetura que possibilite o envio de informações via internet, principalmente envolvendo dados de geolocalização, mas não de forma exclusiva, podendo enviar também outros dados relevantes dependendo da necessidade do projeto.

Considera-se casos em que o dispositivo se move com frequência, funciona em ambientes remotos, podendo ocorrer de forma autônoma, com uma conexão limitada e/ou intermitente.

A solução deve receber os dados de cada dispositivo e prover um armazenamento seguro dessas informações.

Para ilustrar um caso de uso, um usuário acessa uma plataforma *web*, realiza o *log in* na aplicação através de credenciais, ativa um painel para visualizar as informações enviadas por um dispositivo específico em tempo real, de forma dinâmica, podendo ver dados de temperatura, pressão atmosférica, localização e rota realizada no momento exato que o dispositivo os enviou.

Além disso, poderá realizar consultas histórica dos dados de um dispositivo específico.

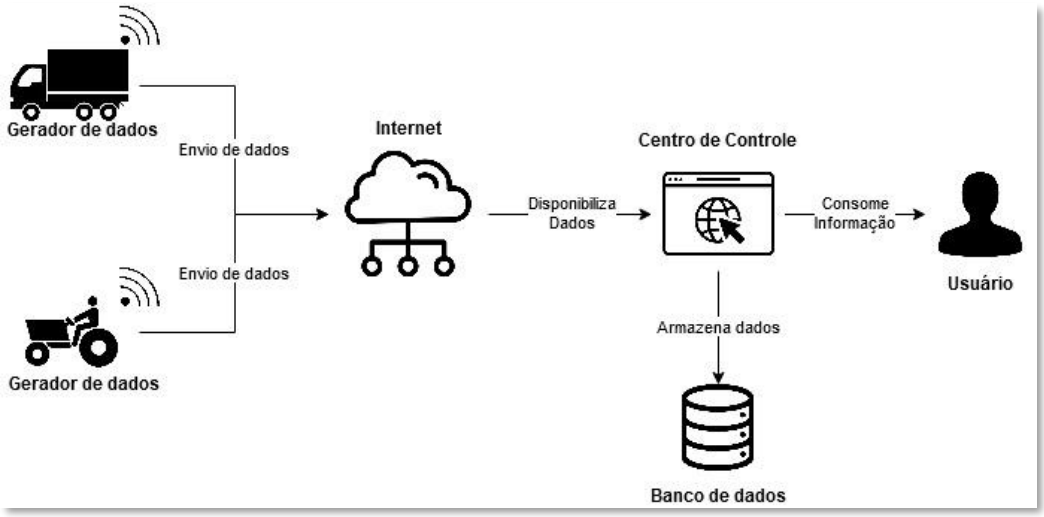
A aplicação web deverá conter também formas de gerenciar os dispositivos e usuários que acessam os serviços.

É importante ressaltar que, em um primeiro momento, a arquitetura gerada será guiada pela finalidade de provar o conceito e funcionamento (*Proof of Concept ou PoC*) da solução. Dessa forma, essa arquitetura não será especificada nos mínimos detalhes, com análises qualitativas das soluções propostas, mas sim o estritamente necessário para

comprovar o envio, recebimento e visualização dos dados gerados pelo dispositivo. O projeto de arquitetura deverá então, em próximas fases de desenvolvimento, passar por iterações de acordo com os modelos de referência apresentados para melhorar aspectos fundamentais de um sistema completo.

A partir da descrição do escopo, um diagrama básico foi gerado para demonstrar o funcionamento de uma forma macro, que consta na **Figura 4** abaixo:

Figura 4 - Diagrama de contexto básico do Sistema IoT proposto



A partir deste diagrama básico, é possível identificar as principais entidades envolvidas. Com a finalidade de definir em maiores detalhes as relações entre elas e elaborar o Modelo de Domínio, cada elemento será analisado.

3.1.1.1 Gerador de dados

Pretende-se descrever a entidade física através do uso de sensores acoplados. A definição da entidade física depende de cada projeto, o escopo proposto não determina uma aplicação específica para tornar a estrutura mais genérica. Dessa forma, será considerado como entidade física a plataforma móvel em que o dispositivo IoT será embarcado,

considerando que para qualquer projeto aplicado será necessário a geolocalização constante dessa plataforma.

Uma importante consideração, para facilitar o desenvolvimento da PoC, é definir que em cada plataforma será acoplado a apenas um dispositivo e este enviará todos os dados relativos àquela. Assim, a plataforma deverá ser representada por uma classe de objeto que contém uma identificação, denominada “DeviceID”, definida pelo usuário.

Esta decisão, no entanto, deverá ser reavaliada em uma etapa posterior para considerar situações em que pode haver mais de um dispositivo enviando dados acoplados à mesma plataforma.

No caso do dispositivo acoplado, deverá conter um sensor para captar a geolocalização da plataforma, um transmissor para enviar os dados gerados e um módulo central para conectar esses componentes e controlar a operação.

3.1.1.2 Internet

No caso do componente “Internet”, alguns aspectos de conexão do dispositivo ao Centro de Controle devem ser analisados. Os principais são os protocolos de comunicação utilizados por cada componente.

Devido à limitação do ambiente, que possui conexão limitada e/ou intermitente, uma atenção especial deve ser voltada para definir os melhores meios de comunicação dos dispositivos, de forma a garantir um nível de Qualidade de Serviço (QoS) adequado para as necessidades do projeto.

3.1.1.3 Centro de Controle

O componente definido como “Centro de controle” é o componente responsável por gerenciar toda a aplicação para o recebimento de dados dos dispositivos, além de disponibilizar essas informações para consumo do usuário.

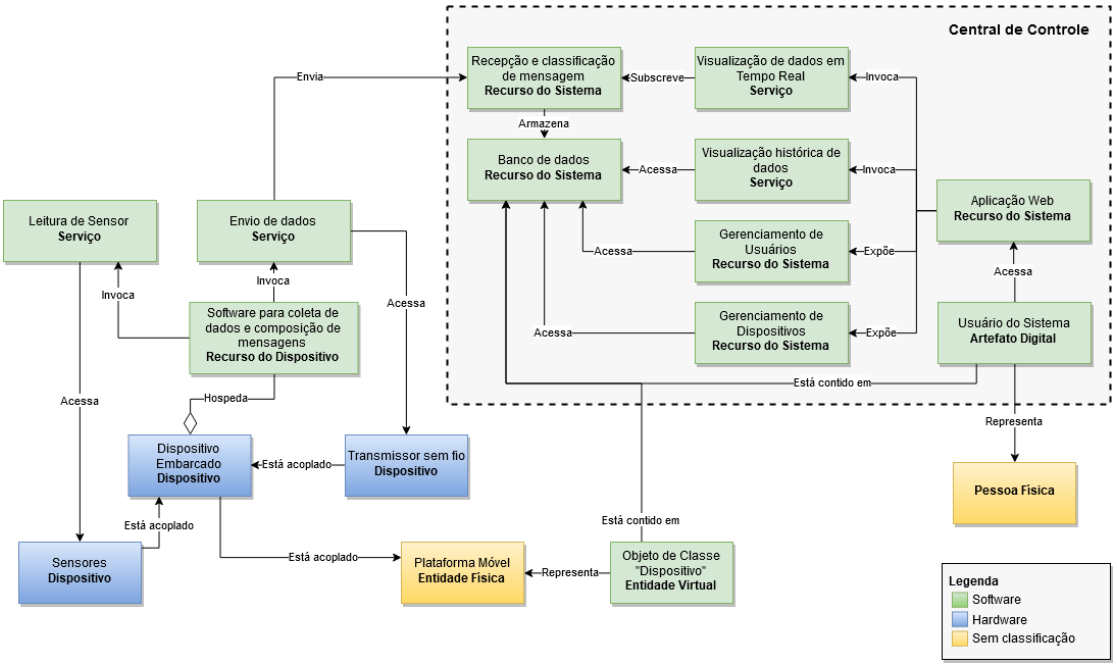
Pela descrição do contexto e funcionalidades necessárias, é possível decompor esse componente em Serviços e Recursos que irão fazer parte do “Centro de Controle”:

- Deve-se recepcionar as mensagens recebidas, de forma a tratá-las e as enviar para o banco de dados de forma estruturada;
- Há dois módulos, um para gerenciamento de usuários e outro para dispositivos;
- Há dois serviços para consumo das informações, um para visualização em tempo real e outra para consulta de histórico.

3.1.1.4 Usuário

Apesar de não ter sido especificado, é razoável supor que haverá níveis de usuários bem definidos, cada qual com permissão de acesso a Serviços e Recursos de forma limitada. Assim, é desejável que o gerenciamento de usuários leve isso em consideração no desenvolvimento.

Figura 5 – Diagrama estrutural do Modelo de Domínio IoT



Com esses conceitos definidos, foi possível gerar o Modelo de Domínio IoT desse caso, ilustrado no diagrama da **Figura 5** acima.

Neste diagrama, é possível identificar a Plataforma Móvel como Entidade Física, que possui um objeto da Classe “Dispositivo” como Entidade Virtual. Para cada dispositivo, será considerado uma instância desse objeto, armazenado no Banco de dados e gerenciado pelo Recurso responsável. De forma semelhante, a pessoa que utiliza o sistema para consumir as informações é representado por um artefato digital, denominado “Usuário”.

Os serviços de visualização das informações são acessíveis por meio de uma Aplicação Web, uma acessando diretamente o Banco de dados para extrair as informações de histórico, enquanto a outra utiliza o conceito de subscrição para receber os dados quando estiverem disponíveis.

Para simplificação, foi considerado que esses dados serão enviados de forma ativa, sem a necessidade de solicitação por parte do usuário. Assim, se estabelece apenas uma via de comunicação entre o centro de controle e dispositivo.

O dispositivo é formado, em sua estrutura mais básica, por sensores, tecnologia de envio de mensagens com conexão remota e um sistema embarcado que irá integrar esses componentes, efetuando o controle das rotinas e lógicas programadas.

3.1.2 Padrão Publicação/Subscrição

Para a estrutura de envio e recebimento de dados, uma prática bem aceita em projetos IoT é a utilização do modelo de Publicação/Subscrição. A arquitetura desse modelo possui três componentes principais: um Serviço, responsável por publicar mensagens, um *Broker* de mensagens, que irá receber e encaminhar as mensagens de forma gerenciada, e um Cliente que irá consumir as mensagens.

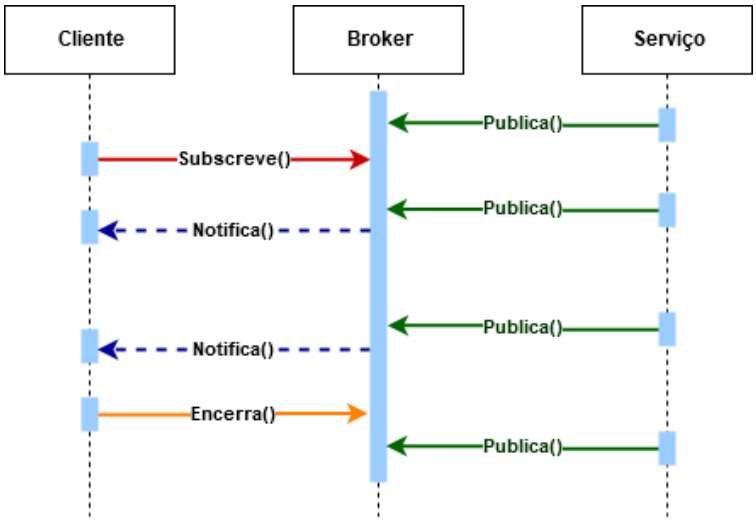
Esse tipo de arquitetura facilita a comunicação de diversos dispositivos operando simultaneamente, com acesso de um ou mais clientes também simultâneos. Essa estrutura

oferece um acoplamento “solto”, no sentido de possibilitar mudanças no sistema sem impactar em grande escala os atuais processos.

Pode-se observar o funcionamento desses componentes na **Figura 6**, o Cliente subscreve à um tópico de interesse no *Broker*, esse mesmo tópico é utilizado pelo Serviço para publicar mensagens. Quando uma sessão é iniciada, toda vez que uma mensagem for recebida será notificada ao Cliente. Isso seguirá até que este encerre a sessão.

O *Broker* permite que vários Serviços publiquem em tópicos distintos e vários Clientes se conectem nos respectivos tópicos de interesse. Assim, um cliente só receberá as mensagens destinadas no tópico ao qual está conectado.

Figura 6 – Diagrama de Sequência do padrão Publicação/Subscrição



Fonte: Bauer et al. (2013).

3.2 Visão Funcional e fluxo de informação

A partir do Modelo de Domínio, identifica-se a necessidade de especificação de funcionamento dos Serviços de envio, armazenamento, consulta e visualização em tempo

real. Para isso, será utilizado o Modelo Funcional, que agrega o fluxo de informações na arquitetura.

Será apresentado uma Visão Funcional e de fluxo de informações utilizando Componentes Funcionais (CF). Esses componentes auxiliam na representação e detalhamento dos Serviços e Recursos identificados no Modelo de Domínio IoT.

Para facilitar a ilustração, os blocos do Modelo Funcional que são utilizados estão em cor de destaque.

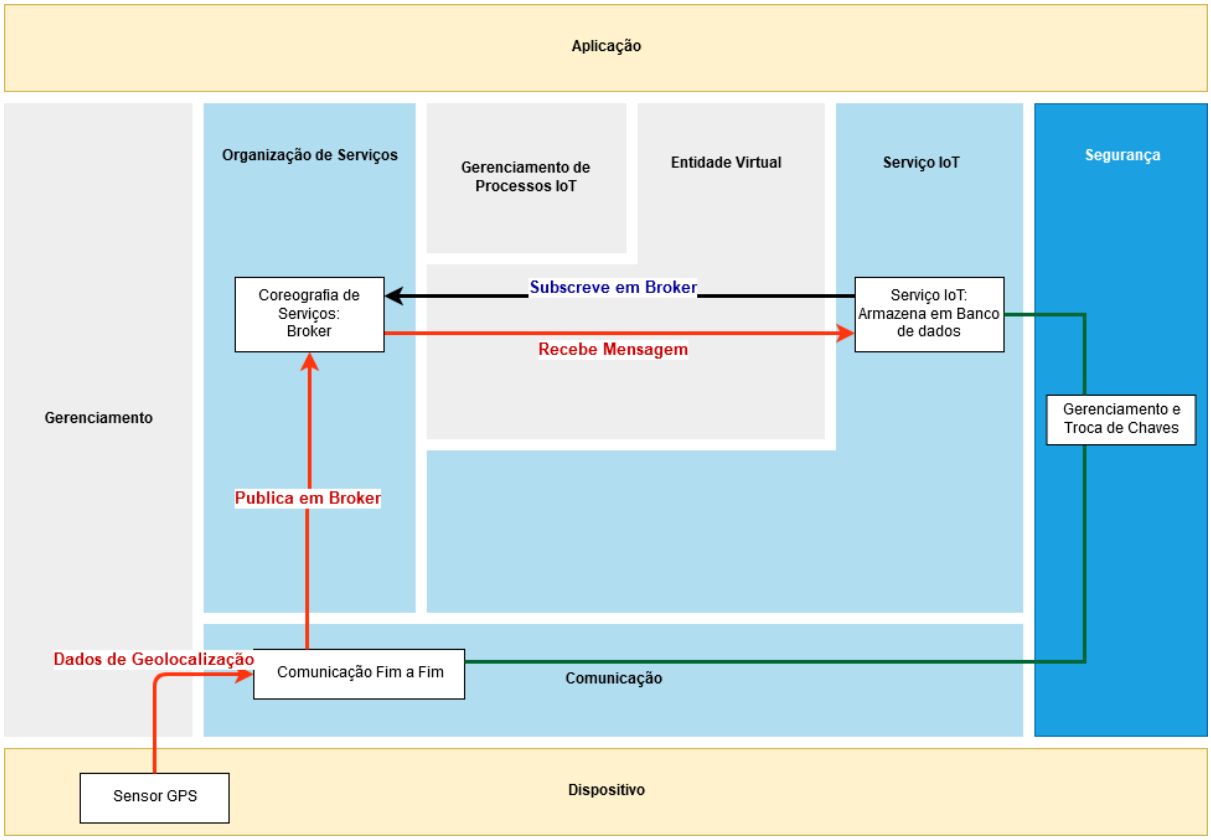
3.2.1 Envio e armazenagem de dados

O dado de geolocalização é capturado pelo sensor GPS, e após processamento pelo dispositivo, é publicado no *Broker* via comunicação Fim-a-Fim. Essa comunicação compreende a ligação entre redes distintas e nesse contexto representará a Internet.

Um serviço ficará responsável por realizar a subscrição no *Broker* e armazenar os dados no Banco de forma estruturada.

Para acessar o *Broker*, uma chave de criptografia será utilizada. Trata-se de uma prática comum adotado em sistemas comerciais e, em uma consulta prévia, se mostrou necessário para utilizar a solução comercial desejada.

Figura 7 - Visão Funcional do serviço de armazenagem de dados



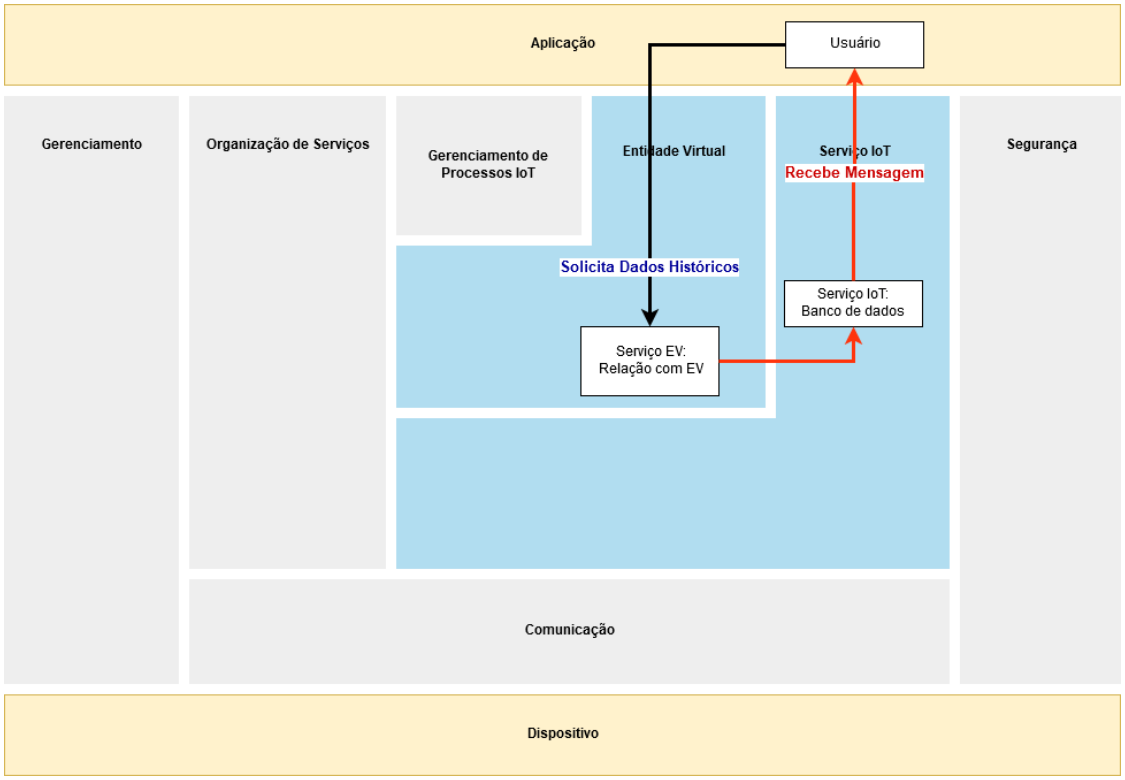
Fonte: Adaptado de Bauer et al. (2013).

3.2.2 Consulta de dados históricos

Para realizar a consulta de dados armazenados, um serviço acessará o Banco e retornará ao usuário os dados encontrados.

Para realizar a consulta, o usuário deverá selecionar qual dispositivo os dados serão retornados, essa função está representada no Bloco “Relação com EV”, pois o serviço é referente à identificação da Entidade Virtual.

Figura 8 - Visão Funcional do serviço de Visualização histórica dos dados

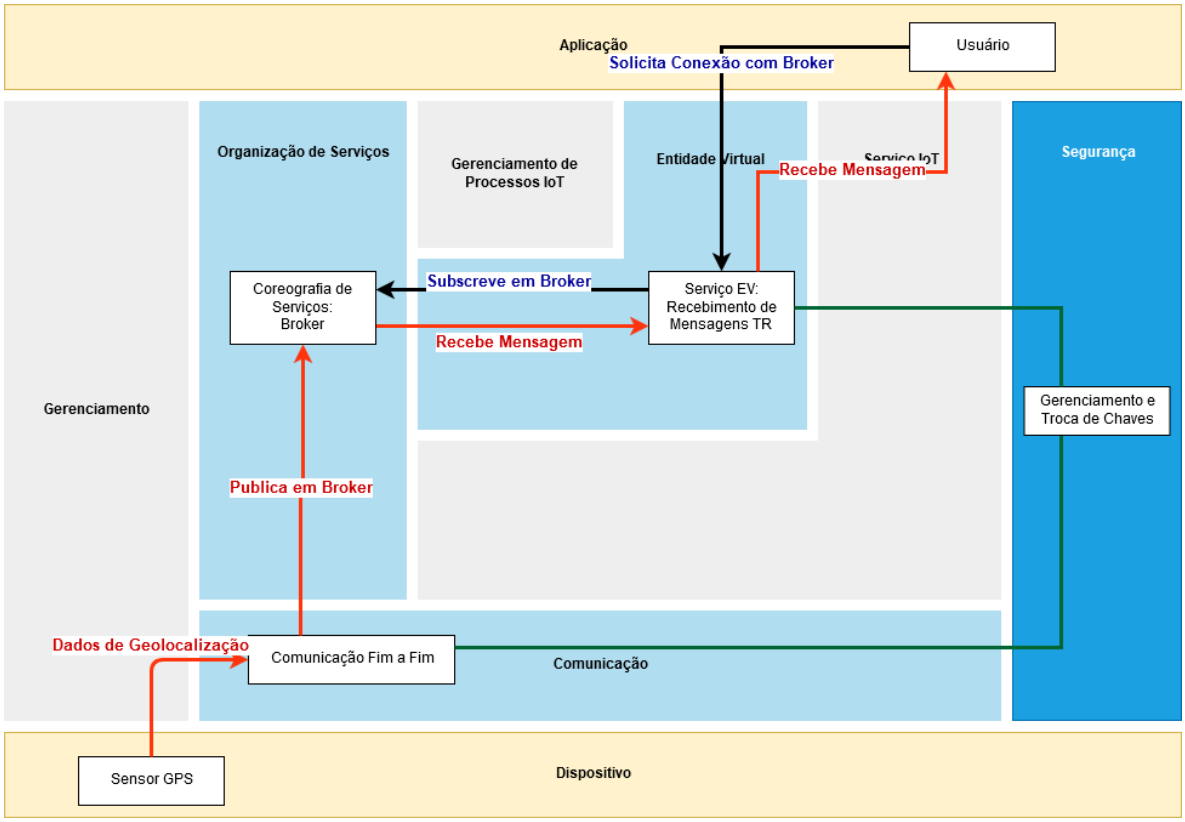


Fonte: Adaptado de Bauer et al. (2013).

3.2.3 Visualização em Tempo Real

O Serviço de "Visualização em Tempo Real" dos dados recebidos segue um processo semelhante ao processo de armazenamento. As informações geradas no dispositivo seguirão via internet ao *Broker*. Nesta etapa, a Aplicação Web assume uma conexão Cliente com o *Broker*, através da seleção da Entidade Virtual desejada e utilização de chaves criptográficas, e obtém as notificações das mensagens recebidas. Com essas informações, a Aplicação Web retorna para o usuário, em formato visual, as informações que o sensor capturou.

Figura 9 - Visão Funcional do serviço de Visualização em Tempo Real



Fonte: Adaptado de Bauer et al. (2013).

Com esses conceitos bem definidos e um certo conhecimento prévio das ferramentas utilizadas, que serão descritas em outro capítulo, algumas considerações práticas começam a tomar corpo no projeto. Com a finalidade de agilizar no processo de desenvolvimento e consolidar a PoC, a comunicação do dispositivo à Internet será provida por um celular.

Esse aparelho irá prover uma comunicação ao dispositivo IoT via rede WiFi, e será utilizado a rede de telefonia móvel para permitir o envio de dados ao *Broker*.

3.3 Visão de Comunicação

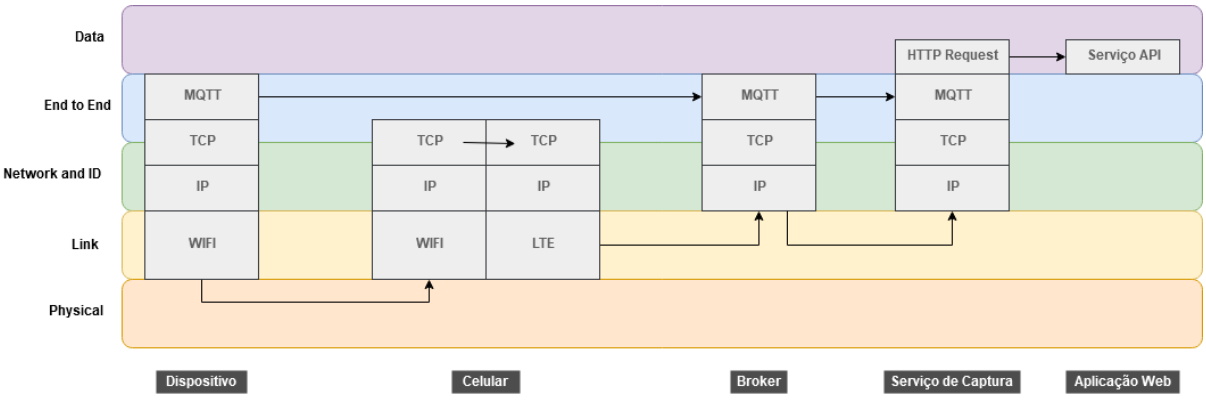
A Visão de Comunicação observa principalmente os protocolos utilizados em cada camada de comunicação definida pelo Modelo de Comunicação. Essa visão abrange todos os atores no processo descrito.

A **Figura 10** Abaixo apresenta o esquema utilizado.

Na pilha definida para o dispositivo, a mensagem gerada é encapsulada com o protocolo MQTT (seção 4.1.4), através de uma conexão WiFi com o aparelho celular, a mensagem é roteada, utilizando a rede de telefonia LTE disponível. A mensagem chega ao *Broker*, que já possui uma conexão de subscrição por parte do serviço de recebimento dos dados, é transmitida via MQTT para o Serviço e, por fim, a mensagem é repassada para a Aplicação Web utilizando uma API disponível no protocolo HTTP.

Na Figura, os elementos de comunicação de outras camadas que não são relevantes não foram representados.

Figura 10 - Visão de Comunicação da arquitetura construída



Fonte: Adaptado de Bauer et al. (2013).

Capítulo 4

Desenvolvimento da Solução

Com o desenho da Arquitetura bem formulado, foi realizado um processo de escolha das ferramentas para aplicar o desenvolvimento prático da solução proposta. As tecnologias escolhidas foram definidas com base no conhecimento prévio do autor, na disponibilidade de determinados dispositivos, em pesquisas de soluções com amplo amparo documental e da comunidade e, considerando também, os custos envolvidos. Grande parte de tecnologias tem licença *Open Source* e possuem livre permissão de uso.

4.1 Tecnologias utilizadas

4.1.1 Raspberry Pi

Raspberry Pi é um computador de pequeno porte e baixo custo, que possui acesso à pinos de Entrada e Saída de dados e funciona com um sistema operacional baseado em Linux.

Foi desenvolvido pela Raspberry Pi Foundation para promover a educação básica de ciência da computação em escolas e acabou tendo uma grande adoção em comunidades de desenvolvimento de projetos robóticos.

Esse aparelho foi utilizado como o “Dispositivo Embarcado” apresentado no Modelo de Domínio IoT.

O modelo utilizado foi o Raspberry Pi 3 B. Segue suas principais especificações técnicas:

- Processador Quad Core 1.2 GHz Broadcom BCM2837 de 64 bits
- 1 GB de memória RAM
- Conectividade Wireless/Lan/Bluetooth Low Energy (BLE) integrados na placa

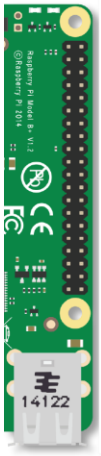
- Ethernet de padrão 100BASE
- 40 pinos de Entrada/Saída (E/S) digitais para uso geral
- 4 portas USB 2
- Conector HDMI para saída de vídeo
- Porta Micro SD para carregar Sistema Operacional e Dados
- Conector Micro USB para fonte de energia de 5V e utilização de até 2.5A

Figura 11 - Raspberry Pi 3 modelo B

Fonte: https://www.raspberrypi.org/

Para utilização das portas de E/S, esse modelo conta com o seguinte diagrama de pinos

Figura 12 - Diagrama de pinos de E/S



Peripherals	GPIO	Particle	Pin #			Pin #	Particle	GPIO	Peripherals
3.3V			1	X	X	2	5V		
I2C	GPIO2	SDA	3	X	X	4	5V		
	GPIO3	SCL	5	X	X	6	GND		
Digital I/O	GPIO4	D0	7	X	X	8	TX	GPIO14	UART
GND			9	X	X	10	RX	GPIO15	Serial 1
Digital I/O	GPIO17	D1	11	X	X	12	D9/A0	GPIO18	PWM 1
Digital I/O	GPIO27	D2	13	X	X	14	GND		
Digital I/O	GPIO22	D3	15	X	X	16	D10/A1	GPIO23	Digital I/O
3.3V			17	X	X	18	D11/A2	GPIO24	Digital I/O
SPI	GPIO10	MOSI	19	X	X	20	GND		
	GPIO9	MISO	21	X	X	22	D12/A3	GPIO25	Digital I/O
	GPIO11	SCK	23	X	X	24	CE0	GPIO8	SPI (chip enable)
GND			25	X	X	26	CE1	GPIO7	
DO NOT USE	ID_SD	DO NOT USE	27	X	X	28	DO NOT USE	ID_SC	DO NOT USE
Digital I/O	GPIO5	D4	29	X	X	30	GND		
Digital I/O	GPIO6	D5	31	X	X	32	D13/A4	GPIO12	Digital I/O
PWM 2	GPIO13	D6	33	X	X	34	GND		
PWM 2	GPIO19	D7	35	X	X	36	D14/A5	GPIO16	PWM 1
Digital I/O	GPIO26	D8	37	X	X	38	D15/A6	GPIO20	Digital I/O
GND			39	X	X	40	D16/A7	GPIO21	Digital I/O

Fonte: http://docs.particle.io

O Sistema Operacional utilizado é o Raspbian Jessie Lite, versão simplificada que dispensa aplicativos pré-instalados na versão tradicional do sistema. A operação do dispositivo é através de uma interface de comando via código.

4.1.2 Módulo GPS

Para a captura de dados de geolocalização, foi utilizado um módulo receptor de GPS que contém o Chip NEO-6M, produzido pela u-blox. A arquitetura reduzida, combinado com o baixo consumo de energia faz desse chip propício para dispositivos móveis alimentados por bateria e pequenos espaços.

Especificações técnicas:

- Tensão de operação: 2.7 - 3.6 V;
- Comunicação: UART (9600 Baud rate);
- Protocolo de mensagens: NMEA 0183 v2.3
- Limites de temperatura de operação: -40 a 85 °C

- Corrente máxima consumida: 67 mA

Figura 13 - Módulo GPS com chip u-blox NEO-6M e Antena

Fonte: http://qqtrading.com.my/

4.1.2.1 Protocolo NMEA 0183

O protocolo NMEA 0183, segundo Langley (1995), foi desenvolvido na década de 80 pela *National Marine Electronics Association*. Nasceu como uma forma de padronizar a comunicação entre dispositivos eletrônicos marítimos e se estabeleceu como o principal padrão de interface de receptores GPS.

A transmissão de dados utiliza apenas texto com caracteres no padrão ASCII de 7 bits. As mensagens recebidas possuem uma estrutura denominada “sentença”, composta por uma ordem sequencial de informações e separadas por um caractere delimitador. Atualmente, existem diversos tipos de sentenças registrados, cada qual está relacionada com um tipo de informação específico. Algumas são padrões abertos, mas há também padrões próprios desenvolvido por empresas para seus aparelhos.

A estrutura da sentença segue o seguinte padrão:

$ determina o início da sentença

GP determina o tipo de dispositivo que origina o dado

GGA determina o tipo de mensagem

,... descrição dos dados da sentença delimitados por vírgula

*76 apresenta a soma de verificação (*checksum*) opcional

<CR><LF> <*Carriage Return*><*Line feed*> determinam o fim da sentença

O módulo receptor GPS NEO-6M utilizado comunica os seguintes tipos de sentenças: GSV, RMC, GSA, GGA, GLL e VTG. Neste trabalho será utilizado apenas o tipo GGA.

A estrutura da sentença GGA está detalhada na **Tabela 1** abaixo.

Tabela 1 - Relação de dados e seus respectivos significados na Sentença GGA

Mensagem	Significado
GGA	Tipo de sentença
011919.00	Hora em UTC da medição
2337.00815,S	Latitude
04638.27288,W	Longitude
1	Qualidade da medição
04	Número de Satélites rastreados
1.81	Diluição Horizontal de Precisão
849.7,M	Altitude
-5.3,M	Altura Geoidal sobre o elipsóide WGS84
****Nenhum dado recebido**	Tempo em segundos desde última DGPS
****Nenhum dado recebido**	Número de Identificação da Estação DGPS
***40**	Soma de Verificação (checksum)

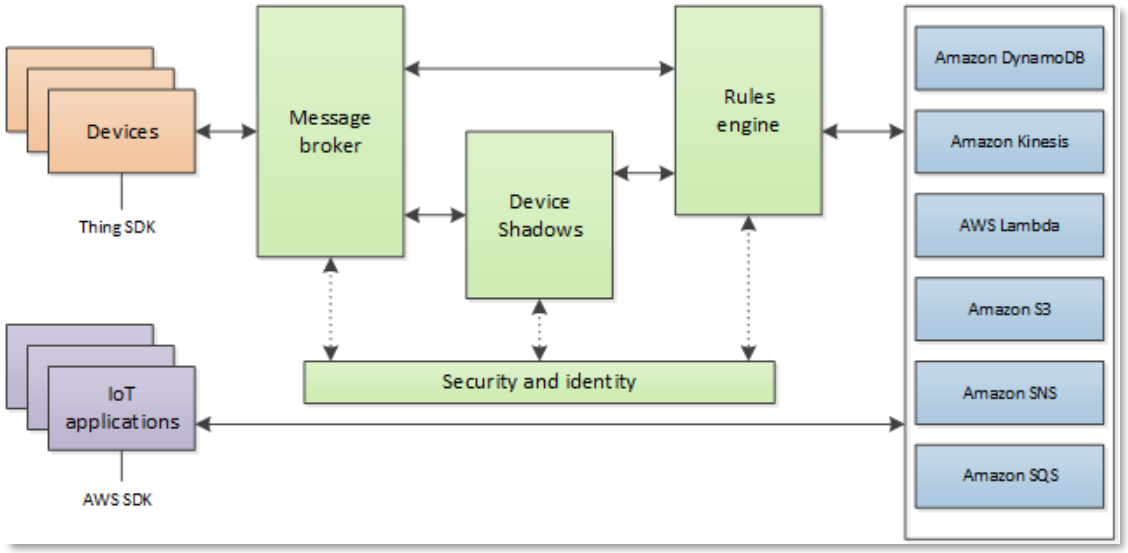
4.1.3 AWS IoT

O AWS IoT é parte de uma plataforma de serviços de computação em nuvem, ofertada pela Amazon Web Services (AWS), Inc. O seu funcionamento provê a conexão de dispositivos na Internet com aplicações de uso geral ou outros serviços da mesma plataforma, como Banco de dados.

O serviço oferece uma conexão bidirecional segura entre dispositivos conectados à internet, como sensores, atuadores e microcontroladores para o recebimento de dados e

gerenciamento dos mesmos. Há também, a possibilidade de utilização dos dados em aplicações diversas com o uso de APIs ou SDK fornecido pela AWS.

Figura 14 - Arquitetura de funcionamento do AWS IoT



Fonte: Amazon Web Services documentation.

Entre os principais componentes do AWS IoT, segue:

- ***Gateway* de Dispositivos:** possibilita a comunicação segura entre dispositivos e o serviço AWS IoT;
- ***Broker* de Mensagens:** oferece um mecanismo de comunicação que segue o padrão de publicação/subscrição em tópicos e utiliza os protocolos MQTT e MQTT sobre *Websockets*. Há também a possibilidade de publicar uma mensagem utilizando o padrão HTTP REST;
- **Motor de Regras:** possibilita o processamento das mensagens antes de enviar para o seu destino, podendo conectar com outros serviços da AWS ou aplicações próprias;
- ***Devices Shadows*:** provê informações de estado dos dispositivos, podendo ser atualizados frequentemente de forma sincronizada;

- **Identidade e Segurança:** o envio e recebimento de mensagens funciona com a utilização de certificados de segurança no padrão X.509, previamente configurados para receber ou acessar os dados apenas por dispositivos e agentes autenticados e autorizados;
- **AWS SDK:** o serviço fornece uma biblioteca SDK (Software Development Kit) que possibilita o acesso de aplicações próprias aos serviços AWS. Outra forma de acesso é via API, utilizando requisições HTTP/HTTPS.

4.1.4 Protocolo MQTT

O protocolo MQTT é uma extensão do padrão publicação/subscrição e foi desenvolvido especificamente para aplicações de telemetria em ambientes com recursos limitados (HUNKELER e TRUONG, 2008). Aplica-se o mesmo conceito de Cliente/*Broker*/Serviço apresentado anteriormente.

A escolha desse protocolo leva em consideração a menor quantidade de bytes e consumo de banda em comparação com o protocolo HTTP, que apresenta uma diferença crescente e considerável com o aumento de dispositivos na rede (YOKOTANI; SASAKI, 2016).

Uma característica desse protocolo é a utilização de tópicos para a publicação de mensagens, assim, é possível separar níveis de hierarquia, delimitados pelo caractere “/”, para agrupar as mensagens de dispositivos num mesmo contexto. Por exemplo, um dispositivo envia uma medição de Temperatura no tópico “Zona3/Ambiente1/SensorXYZ”, dessa forma, a tarefa de identificação do sensor, ambiente e zona ao qual os dados se referem é bem simples. Além desse aspecto, é possível se subscrever no tópico “Zona3/Ambiente1/” e receber os dados de todos os dispositivos alocados nessa região.

Por fim, o MQTT apresenta suporte ao conceito QoS (Qualidade de Serviço) para a comunicação Fim-a-Fim. O QoS estabelece níveis distintos de controle do envio de mensagens entre as redes, que variam conforme a necessidade de confiabilidade da comunicação. O QoS é definido na aplicação.

Existem três níveis de envio:

- **Nível 0:** a mensagem será enviada uma ou nenhuma vez ao destinatário. É o nível mais simples, não possui reconhecimento do recebimento (*aknowledgement*) ou retransmissão de dados;
- **Nível 1:** a mensagem recebida no destinatário será reconhecida por um *acknowledgment*. Caso não seja reconhecida, a mensagem é retransmitida pelo emissor. Assim, será enviada uma ou mais vezes ao destinatário;
- **Nível 2:** garante o envio de uma, e apenas uma, mensagem do emissor para o destinatário.

4.1.5 Django/Python

O desenvolvimento das aplicações que rodam tanto no dispositivo quanto nos sistemas e suas integrações utiliza a linguagem Python, que é nativa do Sistema Operacional Raspbian, operando na versão 3.5.

A linguagem possui uma estrutura de alto nível, orientada a objetos, e é de fácil aprendizado, com uma sintaxe relativamente simples, sem muita verbosidade característica de outras linguagens.

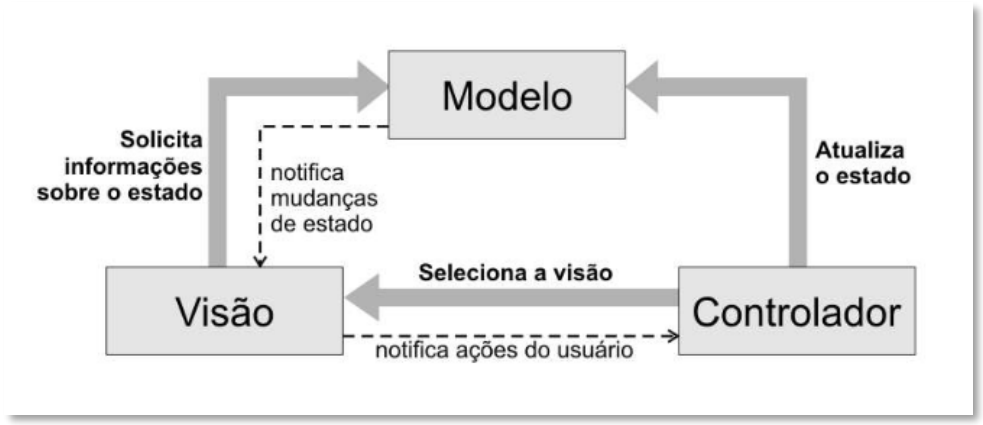
Além disso, possui uma ampla comunidade que desenvolve uma infinidade de pacotes, módulos, *frameworks* e bibliotecas que agilizam a aplicação prática em projetos.

O desenvolvimento da aplicação web em específico utiliza o *framework* Django, também desenvolvido em Python, para facilitar a criação de uma estrutura organizada. O Django já possui alguns módulos prontos, como o de segurança e gerenciamento de usuários. Além disso, possui também um sistema de diretórios estruturados sob o conceito de MVC (*Model-View- Controller*).

O MVC separa a aplicação em três categorias. O “Modelo” é uma camada de abstração para manipulação de dados. Desta forma, facilita a criação de tabelas e

manipulação de seus dados. A “Visão” apresenta a camada de visualização do sistema, mostrando o conteúdo e dados coletados do Modelo. Por fim, o Controlador conduz a aplicação, respondendo às requisições de usuários, interagindo com o modelo e apresentando as visões pertinentes.

Figura 15 - Relacionamento entre as camadas da arquitetura MVC



Fonte: SUN (2011).

Além disso, Django também facilita o desenvolvimento de API’s (*Application Programming Interface*), conexão com Banco de dados e possui a possibilidade de utilização de aplicações desenvolvidos por terceiros.

Esses recursos foram utilizados para compor o *Back-end*, ou estrutura interna, da aplicação web desenvolvida.

4.1.6 WebSocket

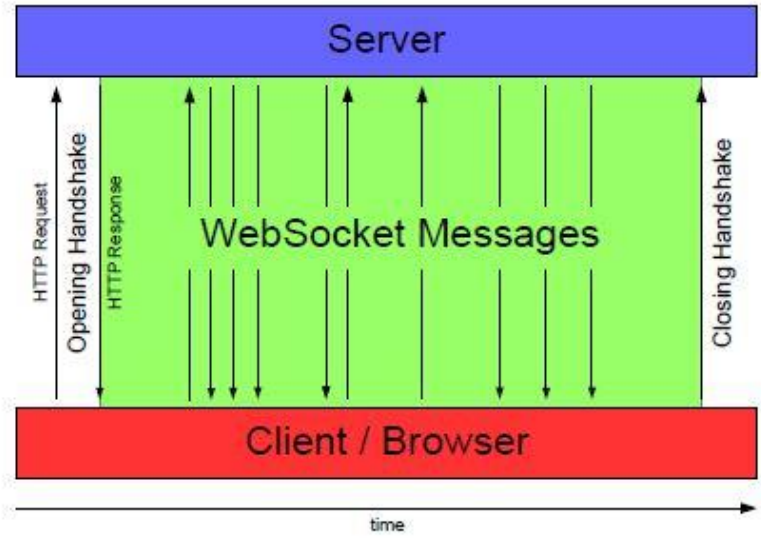
Apesar das facilidades proporcionadas pelo modelo MVC do Django, desenvolver uma funcionalidade de tempo real não é tão simples e desejável com o uso de técnicas HTTP, como *Polling* e *Long Polling*, por possuírem um escopo limitado de uso (Pimentel, 2012). Para uso em tempo real, podem apresentar altas latências, sobrecarga nos *HEADERS* da

requisição (Srinivasan, 2013). Isso se traduziu no uso do padrão de *WebSockets*, que possui comunicação bidirecional, full-duplex e opera sobre um único *socket*.

Há duas etapas de funcionamento, primeiramente o Cliente requisita uma conexão via *WebSocket* e, obtendo uma resposta positiva do Servidor (*Opening handshake*), passa para uma segunda etapa que abre um canal de comunicação e troca de dados entre as duas partes até que outra requisição solicite a finalização (*Closing Handshake*).

Para desenvolver utilizando esse padrão, foi instalado o *Channels*, módulo que expande as habilidades de desenvolvimento do Django além do padrão HTTP com o uso de códigos assíncronos. Esse módulo faz uso de um Banco de dados em estrutura *in-memory*, chamado Redis, para efetuar as transações de dados entre Cliente e Servidor.

Figura 16 - Esquema de funcionamento do Websocket



Fonte: Srinivasan (2013).

4.1.7 HTML/CSS/Javascript

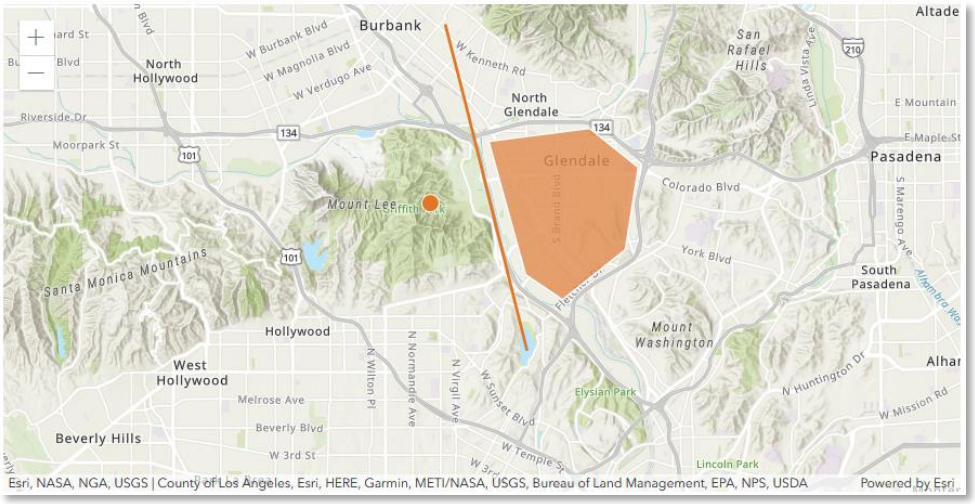
Na parte de *Front-end*, ou seja, a interface de utilização do usuário, a estrutura básica das páginas exibidas foi desenvolvida utilizando HTML/CSS, juntamente com funções em Javascript que viabilizaram a utilização de componentes dinâmicos em aplicações de visualização de mapas e gráficos interativos em tempo real, que serão detalhados abaixo:

4.1.7.1 Arcgis API for Javascript

Trata-se de uma API desenvolvida pela ESRI, empresa que oferta produtos e serviços de software de GIS (*Geographic Information System*).

Através dessa Integração, que é livre para uso, mas com funcionalidade limitadas em comparação às alternativas pagas, é possível criar mapas interativos de alta qualidade, aplicar camadas de visualização, desenhar pontos, linhas e polígonos em coordenadas específicas.

Figura 17 - Exemplo de utilização da API Arcgis



Fonte: https://developer.arcgis.com

4.1.7.2 Charts.js

Charts.js é um módulo em Javascript para reprodução de gráficos diretamente no navegador. É de fácil implementação em não necessita de muitas configurações.

4.1.7.3 PostgreSQL

O PostgreSQL é um Banco de dados relacional, de licença *Open Source*, originado do projeto POSTGRES, que foi criado em 1986 na Universidade da Califórnia em Berkeley. Atualmente, é desenvolvido e sustentado por um time global e contribuições particulares de uma comunidade extensa.

Entre suas qualidades, é um sistema muito utilizado em soluções comerciais por suas características de confiabilidade, robustez e desempenho. Também suporta uma grande parte do padrão SQL para realizar operações e consultas no sistema.

A arquitetura segue um modelo Cliente/Servidor:

- Servidor: responsável por gerenciar os arquivos do sistema, autenticar conexões de clientes, e realizar operações comandados por aplicações clientes.
- Cliente: realizado pelo próprio usuário ou sistema intermediário que deseja realizar operações e consultas no Banco de dados. A aplicação pode ser de caráter diverso, variando desde uma aplicação gráfica, um servidor web que deseja acessar as suas páginas hospedadas, a uma ferramenta especializada de gerenciamento ou uma aplicação desenvolvida pelo próprio usuário.

Nesta configuração, aplicações cliente e servidor podem ser hospedados em máquinas distintas, se comunicando através do padrão TCP/IP. Além disso, o sistema é capaz de criar várias conexões de Clientes simultâneas através de um processo de paralelização do processo.

4.1.7.4 pgAdmin

pgAdmin é uma ferramenta, também *Open Source*, para o gerenciamento do PostgreSQL. Possui uma interface gráfica que simplifica a criação, manutenção e uso do Banco de dados.

Através dessa aplicação, é possível visualizar as tabelas criadas, consultar e baixar os dados parametrizados pela linguagem SQL, realizar backups, entre outras funções que são indispensáveis no gerenciamento de um Banco de dados.

4.2 Lógica de Funcionamento

Dentro da solução proposta, foi desenvolvido o código de operação dos componentes, que serão detalhados abaixo.

4.2.1 Dispositivo

O programa que roda no dispositivo, representado na **Figura 18**, funciona em duas etapas. Primeiro, há uma Configuração Inicial que define o ID do Dispositivo, o endereço dos certificados exigidos na conexão com o *Broker* e, por fim, uma configuração da comunicação serial realizada entre o módulo receptor GPS e o microcontrolador.

Para mais detalhes, a cópia do código utilizado está exposta no **Capítulo 8 - Apêndice A.**

Figura 18 - Configurações iniciais do programa que roda no Dispositivo

```
# ------------------- AWS IoT Config ----------------------------------

DeviceID = 'RASP02'

# MQTT Server Endpoint
endpoint = "a9s9ibiau5ux0-ats.iot.sa-east-1.amazonaws.com"

port = 8883

# Defining relative path to certificates and keys
rootca_path = './Certificates/Rasp02/AmazonRootCA1.pem'
key_path = './Certificates/Rasp02/ba596f331b-private.pem.key'
ca_path = './Certificates/Rasp02/ba596f331b-certificate.pem.crt'

DeviceMQTT = mqtt.Client('Device-' + DeviceID)

# ----------------- Device Serial Port Config -----------------------------
# Device using UART to read GPS Module data

SerialPort = serial.Serial("/dev/ttyS0", 9600)
SerialPort.reset_input_buffer()
print('Serial port activated')
```

Após essa etapa, o programa entra em uma rotina de funcionamento repetindo as ações indefinidamente até que o programa seja finalizado. A primeira ação é de “Esperar 3 segundos”, esse procedimento permitirá um tempo suficiente para reunir informações geradas no receptor GPS. As mensagens ficam armazenadas numa memória *buffer* e, ao acionar o comando de leitura, essas mensagens são coletadas e processadas.

A mensagem extraída é do tipo *string* e compreende todas as sentenças recebidas nos últimos três segundos, essas mensagens são separadas pelo seu delimitador, definidos como “\r\n” (<CR><LF> do protocolo NMEA 0183), resultando em um vetor contendo todas as mensagens. Em seguida, o primeiro e último item são excluídos, pois há uma grande probabilidade de estarem incompletos no momento que a ordem de extração do *buffer* é executada. Assim, é realizado uma seleção dos tipos de sentenças, restando apenas as sentenças GGA. Deste vetor final, apenas o último item é coletado para representar as informações de posição.

Após este procedimento, é ativada uma função para coletar a leitura de data/hora do momento presente (*Timestamp*). Com todas informações, a mensagem é composta e enviada ao *Broker*, que contém:

ID do Dispositivo + Timestamp + Sentença GGA

A publicação é feita no tópico “*FieldData*/ID do Dispositivo”, ficando disponível para as aplicações cliente captarem a mensagem.

Após a publicação, o programa entra em modo de espera pelo tempo determinado no código. Por fim, o *buffer* de mensagens acumuladas nesse período é esvaziado, iniciando um novo ciclo de rotina.

Figura 19 - Dispositivo durante funcionamento

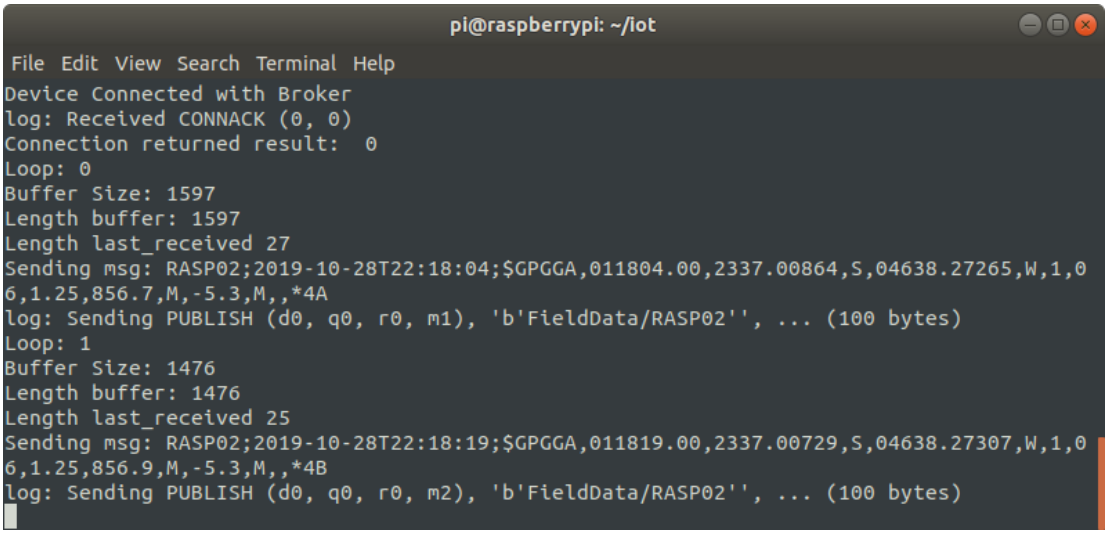
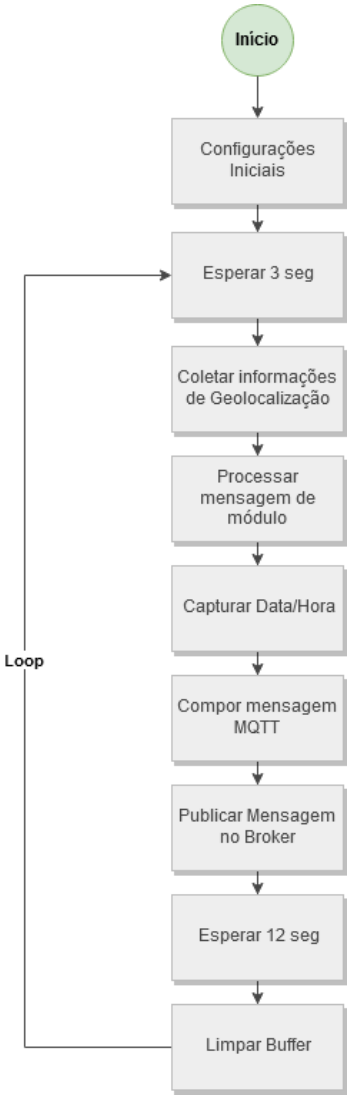


Figura 20 - Diagrama de funcionamento do programa no Dispositivo



4.2.2 Serviço de armazenamento

No Servidor, há um serviço responsável por identificar o recebimento de novas mensagens no *Broker* e os enviar, via API, para a Aplicação *Web* prover o armazenamento no Banco de dados.

O código inicia com a configuração da URL de Requisição POST, disponibilizada pela API da Aplicação Web, e do endereço de certificados de segurança para conexão com o *Broker*. Ao se conectar com o *Broker*, o programa subscreve no tópico "*FieldData/#*" e recebe todas as mensagens publicadas por qualquer dispositivo. Nesta etapa, nenhuma ação será tomada até que uma mensagem seja recebida.

No evento de uma mensagem ser recebida, o *Timestamp* é obtido, a mensagem é decodificada utilizando o padrão "UTF-8", e uma biblioteca desenvolvida por terceiros, chamada "pynmea2", é aplicada para interpretar os dados da sentença GGA. Em seguida, uma mensagem é montada reunindo todas as informações e a requisição POST é enviado à aplicação Web.

Para mais detalhes, a cópia do código utilizado está exposta no **Capítulo 9 - Apêndice B.**

Figura 21 - Implementação prática de Tarefas: obtenção de *Timestamp*, decodificação, interpretação e envio de dados pelo Servidor

```
ServerTimestamp = datetime_now()

message = msg.payload.decode('utf-8')
DeviceID, DeviceTimestamp, payload = message.split(';')

payload = pynmea2.parse(payload)

PostPayload = {
    'DeviceID': DeviceID,
    'ServerTimestamp': ServerTimestamp,
    'DeviceTimestamp': DeviceTimestamp,
    'UTCTime': payload.timestamp,
    'Lat': payload.latitude,
    'LatDirection': payload.lat_dir,
    'Lon': payload.longitude,
    'LonDirection': payload.lon_dir,
    'GPSQual': payload.gps_qual,
    'NumSats': payload.num_sats,
    'Hdop': payload.horizontal_dil,
    'MslAlt': payload.altitude,
    'MslAltUnit': payload.altitude_units,
    'GeoidSeparation': payload.geo_sep,
    'GeoidSeparationUnit': payload.geo_sep_units,
    'AgeLastDgpsUpdate': payload.age_gps_data,
    'DgpsStationID': payload.ref_station_id
}


print('\nEnviando mensagem para WebApp:')
PostRequest = requests.post(url, PostPayload)
print('Status Code: {} | Reason: {} \n'.format(PostRequest.status_code, PostRequest.reason))
```

Figura 22 - Serviço que recebe e armazena dados na Aplicação Web via API

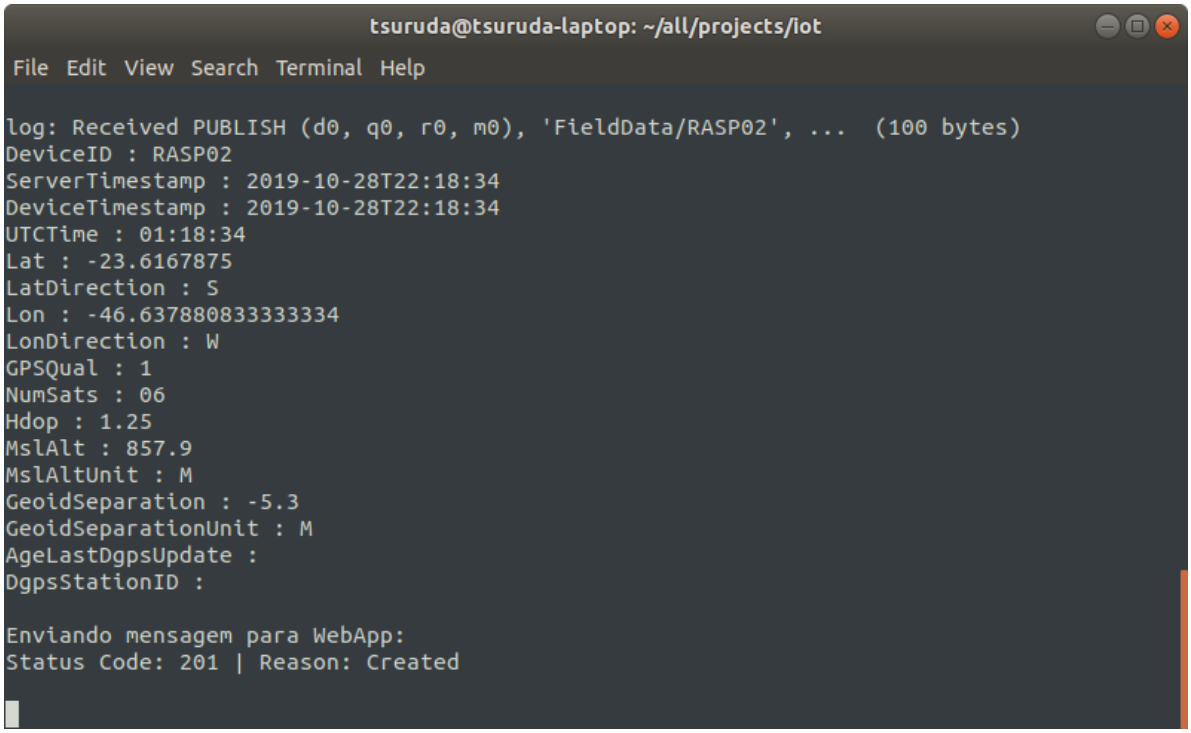
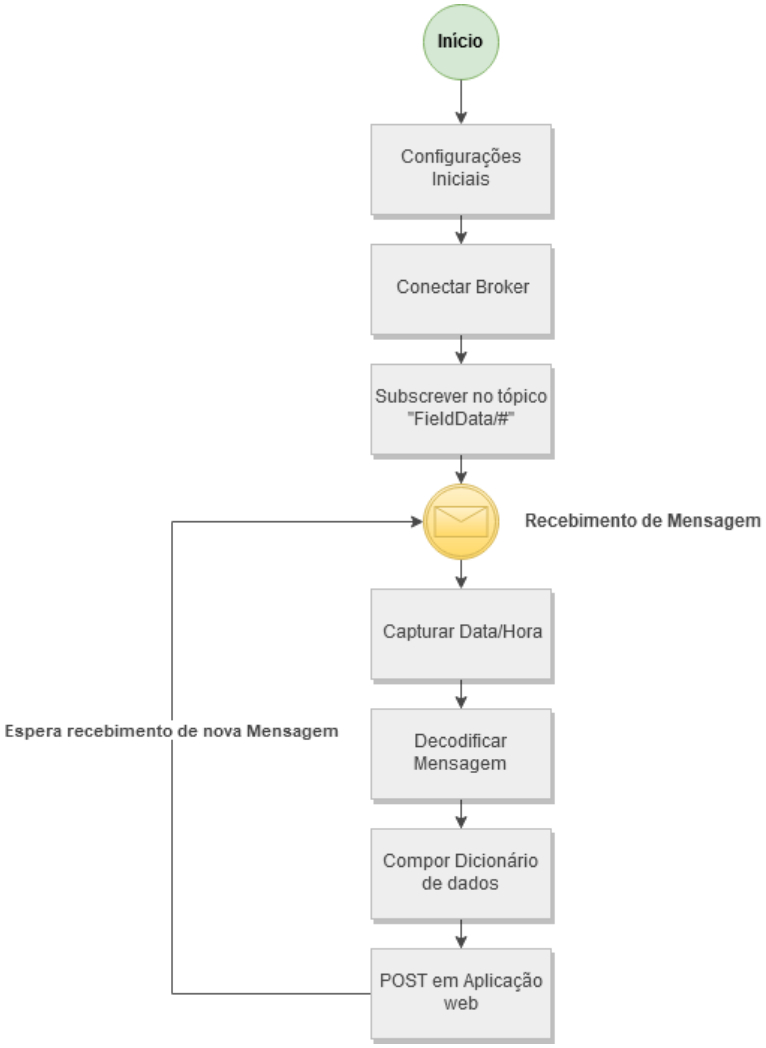


Figura 23 - Lógica do Programa de Recebimento das mensagens



4.2.3 Aplicação Web

A Aplicação Web foi desenvolvida utilizando o *framework web* Django, já citado anteriormente. Esse *framework* possui um sistema de arquivos e diretórios pré-definidos. A estrutura utilizada neste projeto está descrita no **Capítulo 10 – Apêndice C**.

Dentro dos moldes de uso, foi desenvolvido dois aplicativos que implementam as funcionalidades desejadas na Aplicação Web. O primeiro é responsável pela funcionalidade de consulta aos dados históricos armazenados no Banco de dados. Já o segundo, implementa um serviço de acesso em tempo real ao *Broker*, gerando assim uma visualização direta das mensagens atualizadas de um dispositivo específico.

Além dos aplicativos, é importante explicitar também o desenvolvimento de uma API para acesso externo, utilizada pelo serviço de recebimento de mensagens, com a função de fazer uma interface segura com o Banco de dados. Há também outros módulos, responsáveis por cadastrar e gerenciar os dispositivos e usuários da aplicação.

4.2.3.1 API

Através de uma requisição POST dentro dos padrões estabelecidos, no endereço disponibilizado, os dados podem ser armazenados diretamente, sem que o requisitante obtenha acesso direto ao ambiente do Banco de dados. Para implementar essa funcionalidade, uma aplicação chamada Django REST *Framework* foi empregada, sendo necessário especificar um modelo de dados que rege a criação de uma tabela no Banco, um "serializador" para especificar quais campos serão utilizados, e uma estrutura de *URL* para efetuar a requisição. Como demonstrado abaixo:

Figura 24 - Estrutura no Banco de dados para armazenar as informações do receptor GPS

```
class GpsGGA(models.Model):

    DeviceID = models.CharField(max_length=20)
    ServerTimestamp = models.DateTimeField()
    DeviceTimestamp = models.DateTimeField()
    UTCTime = models.TimeField(default=None, null=True, blank=True)
    Lat = models.DecimalField(max_digits=20, decimal_places=17, default=None, null=True, blank=True)
    LatDirection = models.CharField(max_length=1, default=None, null=True, blank=True)
    Lon = models.DecimalField(max_digits=20, decimal_places=17, default=None, null=True, blank=True)
    LonDirection = models.CharField(max_length=1, default=None, null=True, blank=True)
    GPSQual = models.IntegerField(default=None, null=True, blank=True)
    NumSats = models.IntegerField(default=None, null=True, blank=True)
    Hdop = models.DecimalField(max_digits=4, decimal_places=2, default=None, null=True, blank=True)
    MslAlt = models.DecimalField(max_digits=7, decimal_places=1, default=None, null=True, blank=True)
    MslAltUnit = models.CharField(max_length=2, default=None, null=True, blank=True)
    GeoidSeparation = models.DecimalField(max_digits=4, decimal_places=1, default=None, null=True, blank=True)
    GeoidSeparationUnit = models.CharField(max_length=1, default=None, null=True, blank=True)
    AgeLastDgpsUpdate = models.TextField(default=None, null=True, blank=True)
    DgpsStationID = models.TextField(default=None, null=True, blank=True)
```

Figura 25 - Serializador

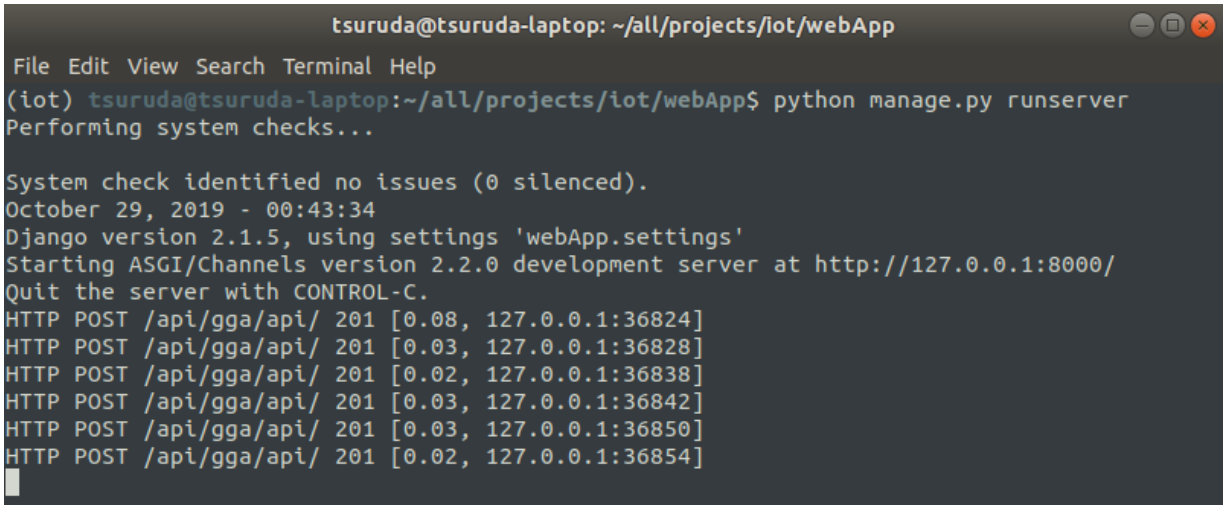
```
class GGASentence(serializers.ModelSerializer):
  class Meta:
    model = GpsGGA
    fields = (
      'DeviceID',
      'ServerTimestamp',
      'DeviceTimestamp',
      'UTCTime',
      'Lat',
      'LatDirection',
      'Lon',
      'LonDirection',
      'GPSQual',
      'NumSats',
      'Hdop',
      'MslAlt',
      'MslAltUnit',
      'GeoidSeparation',
      'GeoidSeparationUnit',
      'AgeLastDgpsUpdate',
      'DgpsStationID'
    )
```

Figura 26 – Implementação de URL da API

```
ggaRouter = routers.DefaultRouter()
ggaRouter.register('api', views.GGAView)

urlpatterns = [
  path('gga/', include(ggaRouter.urls)),
```

Ao enviar dados com sucesso, uma mensagem de log é retornada com a situação da operação. É possível verificar o resultado da solicitação demonstrado na **Figura 27**, a mensagem “HTTP POST /api/gga/api/ 201” indica sucesso na operação.

Figura 27 - Funcionamento da Aplicação *Web*



4.2.3.2 Real Time

Para desenvolver a funcionalidade de visualização de dados em tempo real, foi necessário empregar o uso de *WebSockets*, descrito anteriormente, pois a estrutura principal do Django apenas lida com situações de requisição, processamento e resposta ao requisitante, fechando a conexão e não possibilitando estabelecer uma comunicação duradoura com o *Broker*. Há a possibilidade de uso de técnicas como *Polling*, *Long Polling* ou *Streaming* (SRINIVASAN, 2013), mas implica em diversas desvantagens em termos de tempo de resposta, aplicabilidade e tamanho de dados trocados.

Dessa forma, a Aplicação *Web* foi configurada de forma a aceitar conexões via *Websocket* solicitadas pelo Cliente. Para isso, foi necessário desenvolver alguns componentes utilizando o Módulo *Channels*, que expande as capacidades de resposta do Django para modos assíncronos.

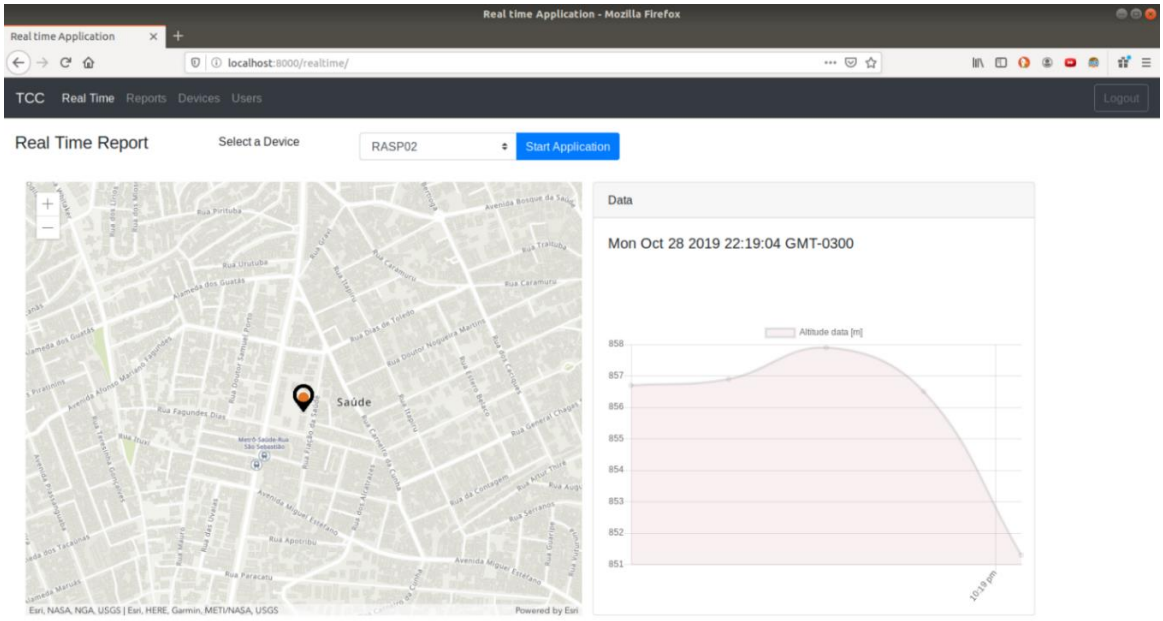
Para atender essa demanda, foi desenvolvido um módulo nomeado *Consumer*, responsável por definir as regras de funcionamento do *WebSocket*. Neste módulo, são especificadas as ações tomadas para os casos de solicitação de conexão pelo Cliente, para casos de recebimento de mensagens e na ocasião da conexão ser encerrada.

Também faz parte do desenvolvimento um código nomeado *Routing* que expõe um endereço URL de conexão ao *Websocket*, desta forma o cliente pode solicitar a conexão.

Na plataforma da aplicação *Web*, com a finalidade de estabelecer uma conexão via *WebSocket* e acessar os dados em tempo real, o usuário deve selecionar um dispositivo no menu apresentado e apertar o botão de início. Assim que a conexão é validada pelo *Consumer*, uma requisição é então enviada para o *Broker*, a fim de subscrever o Cliente a um tópico definido pelo Dispositivo Selecionado "*FieldData*/Dispositivo Selecionado". Caso a solicitação retorne com sucesso, as novas mensagens recebidas pelo *Broker* são processadas pelo Servidor e Disponibilizadas para visualização do usuário no *Browser*.

Para ilustrar a utilização de outras informações, foi incluído um gráfico ao lado do mapa que contém informações de Altitude, enviadas também pelo GPS na sentença GGA. Esse gráfico demonstra a possibilidade de incluir outras informações relevantes ao projeto, como temperatura, pressão, humidade, entre outras.

Figura 28 - Visualização em Tempo Real na Aplicação Web

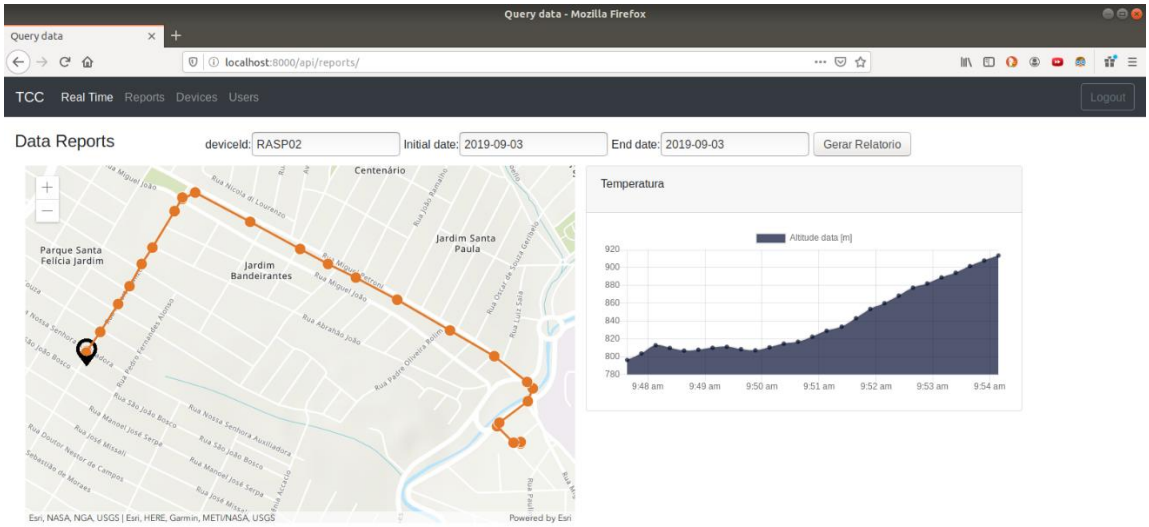


4.2.3.3 Visualização histórica dos dados

Para visualizar os dados armazenados no Banco da aplicação, o Django oferece ferramentas de consulta que são semelhantes à linguagem SQL. Para filtrar os dados, foi utilizado um formulário contendo a identificação do Dispositivo e as datas de início e fim.

Assim, com a entrada dessas informações pelo usuário, a aplicação retorna um mapa com as coordenadas encontradas e, também, um gráfico com os dados de Altitude, como foi explicado no item 4.2.3.2 anteriormente.

Figura 29 - Visualização de dados históricos – resultado da consulta de dados



4.2.3.4 Gerenciamento de Dispositivos e Usuários

Por fim, foi desenvolvido um módulo para Gerenciamento dos dispositivos utilizando ferramentas disponíveis no Django. Essas ferramentas proveram as funcionalidades CRUD (*Create, Read, Update, Delete*) de forma simples com os modelos de dados e tabelas do Banco PostgreSQL. Essas funcionalidades têm o objetivo de realizar operações de criação de registros, leitura das informações registradas, atualização ou exclusão dessas informações.

O módulo que faz o gerenciamento dos usuários é inteiramente provido pelo Django. Esse módulo apresenta uma interface diferente ao resto dos recursos da aplicação, mas foi decidido manter apenas a sua funcionalidade para o ganho de tempo no desenvolvimento, uma vez que o objetivo inicial do projeto é demonstrar o seu funcionamento apenas.

Capítulo 5

Resultados e Discussões

No estudo teórico, um grande esforço foi despendido no entendimento e aplicação dos conceitos do Modelo de Referência. O motivo principal foi a falta de familiaridade com os conceitos, muitos deles advindos de modelos de arquitetura de software. Pode-se ressaltar também que o texto apresenta diversos aspectos abstratos, onde é desejável alguma experiência prévia para o seu entendimento.

Na parte prática de desenvolvimento, o trabalho se dividiu em várias etapas subdivididas por componentes da arquitetura e por funcionalidades da Aplicação Web. Com a análise do esforço despendido e resultados colhidos, é possível avaliar se as expectativas do projeto foram atendidas, além de auxiliar na definição de trabalho futuros.

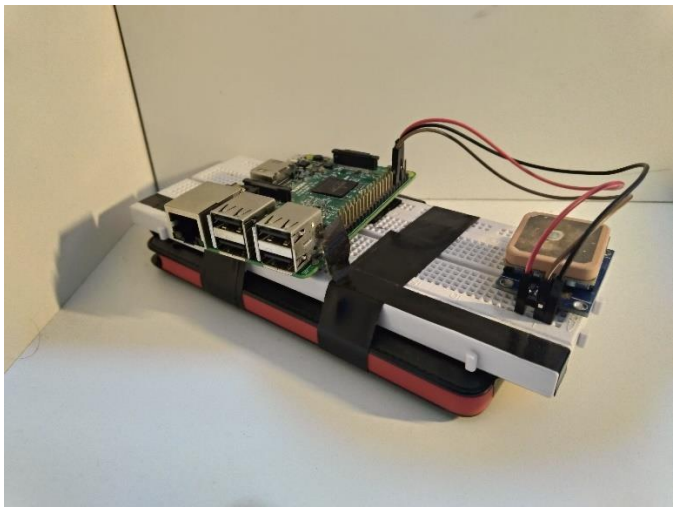
Começando pelo Dispositivo, não houveram dificuldades na montagem de *hardware* pela facilidade de conexão do receptor GPS com a Raspberry Pi e amplo material de apoio disponível. A conexão UART foi utilizada para esse fim e se mostrou bem simples, necessitando apenas de uma configuração no uso da porta serial no dispositivo.

Contudo, na parte de desenvolvimento do software embarcado, exigiu um maior período de testes, à moda da tentativa e erro, para o entendimento correto de uso, uma vez que os documentos consultados não explicitaram o modo de funcionamento do *buffer* de leituras de dados, além das funções utilizadas para o manipular.

No funcionamento do dispositivo em ambiente remoto, foi utilizado uma bateria convencional, de 10.000 mAh, que proporcionou energia ao Raspberry Pi sem interrupções, ao menos avaliado nos períodos de testes em campo. Não foram realizados testes de tempo de funcionamento desta configuração.

Para acesso à Internet, necessário para a comunicação com o *Broker*, não foi implementado um módulo específico de comunicação. Na configuração realizada, foi feito uso de um aparelho celular padrão para fornecer rede e conexão ao dispositivo. Pode-se conferir o funcionamento do dispositivo na **Figura 19**, e o dispositivo utilizado na **Figura 30**.

Figura 30 – Receptor GPS acoplado ao Microcontrolador e bateria



O uso do *Broker*, provido pela AWS IoT, foi de fácil implementação, o portal *web* possui uma interface amigável e intuitiva para uso, ao menos apresentado no escopo limitado desta PoC. Para projetos mais complexos, que envolvem o gerenciamento de usuários e autenticação mais confiável de certificados do que o proposto, pode haver uma curva maior para o aprendizado.

Houve um problema, no entanto, na utilização do SDK (*Software Kit Development*) provido pela empresa que não funcionou corretamente. Nesta ocasião, foi necessário o uso de uma biblioteca *open source*, chamada Paho MQTT, para criar as conexões no padrão MQTT entre Dispositivo-*Broker*, Serviço-*Broker* e Aplicação *Web*-*Broker*.

O desenvolvimento da API, que aconteceu em paralelo com o programa de conexão ao *Broker*, também não apresentou problemas para ser implementado. Com o *framework* REST, disponibilizar um serviço de entrada de dados no Banco foi direto e simples. Vale ressaltar, contudo, que a utilização de *frameworks*, apesar de acelerar bastante o tempo de desenvolvimento, acaba limitando em partes as regras de processamento da Aplicação Web, e pode dificultar uma eventual necessidade de processamento dos dados antes que sejam inclusos no Banco da aplicação.

Este problema também esteve presente no desenvolvimento do módulo Gerenciador de Dispositivos implementado na Aplicação Web, que utilizou uma classe pronta para

viabilizar a entrada, visualização, atualização e remoção de informações no modelo CRUD. Com isso, verificou-se uma maior dificuldade em formatação de formulários de preenchimento, tipos de dados e estética dos elementos apresentados ao usuário. Neste caso, por considerar apenas uma PoC, não foi empregado maior esforço para alterar tais aspectos, considerados de menor importância nessa etapa.

O desenvolvimento da funcionalidade de visualização em tempo real dos dados apresentou maior dificuldade no projeto, devido ao envolvimento de tecnologias variadas, em sua maioria fora do conhecimento do autor, como também necessidade de adaptação para funcionamento correto, de forma a não depender somente de soluções privadas.

A implementação de *WebSockets* com o uso de *Channels* apresentou um resultado bastante satisfatório, possibilitando uma comunicação eficiente e rápida entre Cliente/Servidor nos testes realizados.

Para o processamento de informações geográficas, a utilização da API Arcgis em Javascript para a visualização histórica respondeu muito bem nas ocasiões observadas, demonstrando com clareza os pontos, linhas e janela de informações testados. Já no uso em tempo real, apresentou limitações para atualização dos objetos desenhados no mapa. Foi necessário encontrar uma forma distinta, e não ideal, para lidar com a atualização. Isso ocorreu devido à utilização da solução gratuita que não possui todas funcionalidades disponíveis, pois, segundo documentos apresentados pela empresa responsável, há uma versão paga que atende essa necessidade.

Capítulo 6

Conclusão

O trabalho apresentou alguns dos principais conceitos que envolvem o ecossistema de Internet da Coisas. Entre elas, uma interpretação do termo, potencial de impacto dessas tecnologias no mercado, as nove áreas que podem se beneficiar com IoT e, também, três casos de uso que transformaram negócios.

Em uma segunda etapa, foi realizado um estudo de um Modelo de Referência para o desenho de arquiteturas IoT, apresentando os principais conceitos de Entidades envolvidas, Modelo de Domínio, Modelo Funcional, Modelo de Informação e Modelo de Comunicação. Essas ferramentas foram utilizadas para gerar uma arquitetura para um problema proposto pelo autor.

Por fim, os conceitos da arquitetura foram testados, as funcionalidades foram desenvolvidas com sucesso. O modelo de arquitetura proposto, que conta com um dispositivo gerando dados de geolocalização, enviando para um *Broker* via Internet, salvando o dado em um Banco PostgreSQL e gerando uma visualização imediata dos dados, foi provado. Entretanto, há ainda uma distância grande em entre a arquitetura proposta neste trabalho e Sistemas para soluções comerciais. Deve-se avaliar aspectos de segurança com maior detalhe em todos os componentes da arquitetura, assim como aplicar diversos testes de desempenho com múltiplos usuários e/ou dispositivos, apenas para citar alguns itens.

6.1 Trabalhos Futuros

Diante dos fatores apresentados, define-se como trabalhos futuros, na visão do autor:

- Estudar os impactos da tecnologia 5G no ecossistema IoT;

- Implementar um módulo de comunicação M2M no dispositivo para envio de dados;
- Adicionar sensores diversos para geração de dados;
- Avaliar características de segurança da informação em toda a arquitetura;
- Efetuar melhoras em Gerenciamento de Usuários na Aplicação Web;
- Efetuar melhorias no Gerenciamento de Dispositivos da Aplicação Web, com o desenvolvimento de uma comunicação bidirecional entre Aplicação Web e dispostivo;
- Hospedar a solução inteiramente em ambiente Cloud;
- Testar extensivamente o desempenho em várias configurações de usuários, dispositivos e ambientes utilizados.
- Adicionar mais camadas na comunicação dos dispositivos, possibilitando a adição de mais configurações para utilização da arquitetura.

Referências Bibliográficas

AHMED, E. et al. **The role of big data analytics in internet of things.** Computer Networks, v.129. 2017. p. 459-471. Disponível em:<https://doi.org/10.1016/j.comnet.2017.06.013>. Acesso em: 03 Nov. 2019.

ATZORI, L.; IERA, A.; MORABITO, G. **The internet of things: a survey**. Computer Networks, v.54. 2010. Disponível em: <https://doi.org/10.1016/j.comnet.2010.05.010>. Acesso em: 03 Nov. de 2019.

AWS WEB SERVICES. **Aws iot: developer guide.** 2019. Disponível em: <https://docs.aws.amazon.com/pt_br/iot/latest/developerguide/what-is-aws-iot.html>. Acesso em: 03 Nov. de 2019.

BAUER, M. et al**. Enabling things to talk: designing iot solutions with the iot architectural reference model.** Springer, 2013.

BHATIA, A. et al. **Beyond predictive maintenance: The 'art of the possible' with iot.** Boston Consulting Group. 2019. p. 3-16. Disponível em:<http://media-publications.bcg.com/Microsoft-BCG-Focus-MSFT-IoT-Article-V10-Final-r.pdf>. Acesso em: 03 Nov. de 2019.

BRISCOE, N. **Understanding the osi 7-layer model.** PC Network Advisor, Jul. de 2010. Disponível em:<https://www.os3.nl/_media/2014-2015/info/5_osi_model.pdf>. Acesso em: 03 Nov. de 2019.

DJANGO SOFTWARE FOUNDATION. **Django documentation.** 2005-2019. Disponível em:<https://docs.djangoproject.com/en/2.2/>. Acesso em: 03 Nov. de 2019.

DUARTE, Aldrey Rocha. **Metodologia rails: análise da arquitetura model view controller aplicada.** 2011. 33 f. Trabalho de Conclusão de Curso - Universidade Federal de Minas Gerais, Belo Horizonte, 2011.

ENCODE OSS LTD**. Django rest framework.** 2011-2019. Disponível em:<https://www.django-rest-framework.org/>. Acesso em: 03 Nov. de 2019.

ESRI. **Arcgis api for javascript.** 2019. Disponível em:<https://developers.arcgis.com/javascript/>. Acesso em: 03 Nov. de 2019.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. **Mqtt-s – a publish-subscribe protocol for wireless sensor networks.** 2008. IEEE, 2008. Disponível em:<https://doi.org/10.1109/COMSWA.2008.4554519>. Acesso em: 03 Nov. de 2019.

LANGLEY, R. B., **Nmea 0183: a gps receiver interface standard**. GPS World, 1995. Disponível em:<http://gauss.gge.unb.ca/papers.pdf/gpsworld.july95.pdf>. Acesso em: 03 Nov. de 2019.

MIT. **Charts.js.** Disponível em:<https://www.chartjs.org/docs/latest/>. Acesso em: 03 Nov. de 2019.

PGADMIN DEVELOPMENT TEAM. **pgadmin 4.** 2013-2019. Disponível em:<https://www.pgadmin.org/docs/pgadmin4/latest/index.html>. Acesso em: 03 Nov. de 2019.

PIMENTEL, Victoria. **Communicating and displaying real-time data with websocket.** 2012. IEEE Internet Computing. v.16. p.45-53. 2012. Disponível em:<https://doi.org/10.1109/MIC.2012.64>. Acesso em: 03 Nov. de 2019.

RASPBERRY PI FOUNDATION. **Raspberry pi documentation.** Disponível em:<https://www.raspberrypi.org/documentation/>. Acesso em: 03 Nov. de 2019.

ROZANSKI, N.; WOODS, E. **Software systems architecture.** 2012. Disponível em:<https://www.viewpoints-and-perspectives.info/>. Acesso em: 03 Nov. de 2019.

SRINIVASAN, L.; SCHARNAGL, J.; SHILLING, K. **Analysis of websockets as the new age protocol for remote robot tele-operation.** 2013. Elsevier. IFAC Proceedings Volumes v.46. n.29. p.83-88. 2013. Disponível em:<https://doi.org/10.3182/20131111-3-KR-2043.00032>. Acesso em: 03 Nov. de 2019.

SUN MICROSYSTEMS. **Java BluePrints - J2EE Patterns** - JAVA Sun. Disponível em: <http://java.sun.com/blueprints/patterns/mvc-detailed.html>. Acesso em: 03 Nov. de 2019.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **Postgresql 12.0 documentation.** 1996-2019. Disponível em:<https://www.postgresql.org/docs/12/index.html>. Acesso em: 03 Nov. de 2019.

U-BLOX. **Neo-6 u-blox 6 gps modules: data sheet.** 2011

WAYRICH, M.; EBERT, C. **Reference architectures for the internet of things.** 2015. IEEE Software, v.33, n.1, p.112-116, Jan.-Fev. 2016. Disponível em:<https://doi.org/10.1109/MS.2016.20>. Acesso em: 03 Nov. de 2019.

YOKOTANI, T.; SASAKI, Y. **Comparison with http and mqtt on required network resources for iot.** 2016. IEEE, 2016. Disponível em:<https://doi.org/10.1109/ICCEREC.2016.7814989>. Acesso em: 03 Nov. de 2019.

Apêndice A – Código do Dispositivo

```
import datetime, time, serial
import paho.mqtt.client as mqtt

# Timestamp
def datetime_now():
   ts = time.time()
   return datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%dT%H:%M:%S')


# ------------------- AWS IoT Config ---------------------------------

DeviceID = 'RASP02'

# MQTT Server Endpoint
endpoint = "a9s9ibiau5ux0-ats.iot.sa-east-1.amazonaws.com"

port = 8883

# Defining relative path to certificates and keys
rootca_path = './Certificates/Rasp02/AmazonRootCA1.pem'
key_path = './Certificates/Rasp02/ba596f331b-private.pem.key'
ca_path = './Certificates/Rasp02/ba596f331b-certificate.pem.crt'

DeviceMQTT = mqtt.Client('Device-' + DeviceID)

# --------- MQTT Callback Functions ------------------

def on_connect(client, userdata, flags, rc):
   if rc == 0:
      print("Connection returned result: ", rc)
   else:
      print("Connection Failed with code: ", rc)


def on_log(client, userdata, level, buf):
   print("log: " + buf)


DeviceMQTT.on_connect = on_connect
DeviceMQTT.on_log = on_log

DeviceMQTT.tls_set(
   ca_certs=rootca_path,
   certfile=ca_path,
   keyfile=key_path,
)

# ----------------- Device Serial Port Config ----------------------------
# Device using UART to read GPS Module data

SerialPort = serial.Serial("/dev/ttyS0", 9600)
SerialPort.reset_input_buffer()
print('Serial port activated')

# ------------------------------------------------------------------------
```

```
#---------- CALLBACK FUNCTIONS ----------
def receiving(serialport):
    # Receive messages and groups then, return a list with all readings from de buffer

    print("Buffer Size:", serialport.in_waiting)
    buffer = serialport.read(size=serialport.in_waiting).decode('ascii')
    print("Length buffer:", len(buffer))
    if '\n' in buffer:
        last_received = buffer.split('\r\n')

        print("Length last_received", len(last_received))
        ult = len(last_received) - 1    # last element
        last_received = last_received[1: ult]

    else:
        print('buffer if not working')
        last_received = []

    return last_received

def parsing(list):
    # Divide msgs received among other lists to separate type o message (NMEA Sentences)

    GGAMsg = []
    for item in list:
        if item[0:6] =="$GPGGA":
            GGAMsg.append(item)

    return GGAMsg[-1]

# ---------- ROUTINE ----------
try:
    DeviceMQTT.connect(endpoint, port=8883, keepalive=60)
    print("Device Connected with Broker")
    DeviceMQTT.loop_start()

    count = 0
    while True:
        try:
            time.sleep(3)
            print("Loop:", count)
            count += 1
            last_msg = receiving(SerialPort)
            GGAMsg = parsing(last_msg)
            DeviceTimestamp = datetime_now()    # str
            msg = DeviceID + ';' + DeviceTimestamp + ';' + GGAMsg
            print("Sending msg:", msg)
            DeviceMQTT.publish('FieldData/' + DeviceID, msg)

            time.sleep(12)
            SerialPort.reset_input_buffer()

        except Exception as parse_error:
            pass

except Exception as e:
    print(e)
```

```
    pass

except KeyboardInterrupt:
    SerialPort.close()
    DeviceMQTT.loop_stop()
    DeviceMQTT.disconnect()
    print('\nPrograma Finalizado \n')
```

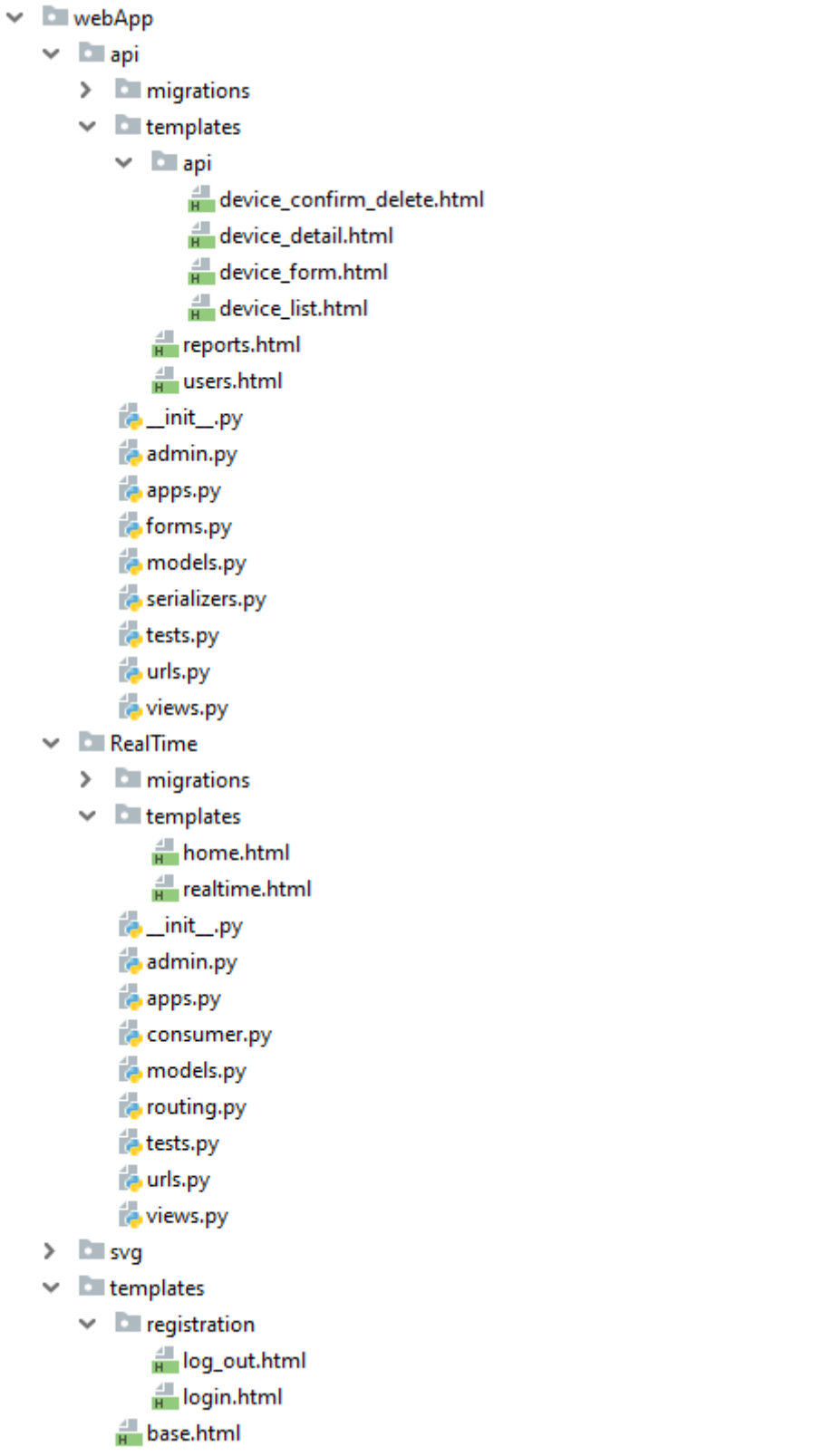
Apêndice B – Código do Receptor

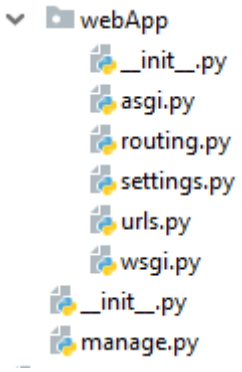
```
import pynmea2
import requests
import paho.mqtt.client as mqtt
import json
import datetime, time

# URL to POST REQUEST
url = 'http://localhost:8000/api/gga/api/'

# MQTT Server Endpoint
endpoint = "a9s9ibiau5ux0-ats.iot.sa-east-1.amazonaws.com"

port = 8883

# Defining relative path to certificates and keys
rootca_path = 'Certificates/server-script/AmazonRootCA1.pem'  # PC
key_path = 'Certificates/server-script/c27a558eff-private.pem.key'
ca_path = 'Certificates/server-script/c27a558eff-certificate.pem.crt'

# NMEA Reader
streamreader = pynmea2.NMEAStreamReader()

#MQTT Client
serverClient = mqtt.Client('Server')

# Time Function: Get DateTime now
def datetime_now():
    ts = time.time()
    return datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%dT%H:%M:%S')

# ---------- Callback functions ----------------

def on_connect(client, userdata, flags, rc):

    if rc == 0:
        print("Connection returned result: ", rc)
    else:
        print("Connection Failed with code: ", rc)


def on_log(client, userdata, level, buf):
    print("log: " + buf)


def on_message(client, userdata, msg):

    ServerTimestamp = datetime_now()

    message = msg.payload.decode('utf-8')
    DeviceID, DeviceTimestamp, payload = message.split(';')

    payload = pynmea2.parse(payload)

    PostPayload = {
```

```
        'DeviceID': DeviceID,
        'ServerTimestamp': ServerTimestamp,
        'DeviceTimestamp': DeviceTimestamp,
        'UTCTime': payload.timestamp,
        'Lat': payload.latitude,
        'LatDirection': payload.lat_dir,
        'Lon': payload.longitude,
        'LonDirection': payload.lon_dir,
        'GPSQual': payload.gps_qual,
        'NumSats': payload.num_sats,
        'Hdop': payload.horizontal_dil,
        'MslAlt': payload.altitude,
        'MslAltUnit': payload.altitude_units,
        'GeoidSeparation': payload.geo_sep,
        'GeoidSeparationUnit': payload.geo_sep_units,
        'AgeLastDgpsUpdate': payload.age_gps_data,
        'DgpsStationID': payload.ref_station_id
    }

    # Altitude Null value Correction
    if payload.altitude == None:
        PostPayload['MslAlt'] = ''

    for item in PostPayload:
        PostPayload[item] = str(PostPayload[item])
        print("{} : {}".format(item, PostPayload[item]))



    print('\nEnviando mensagem para WebApp:')
    PostRequest = requests.post(url, PostPayload)
    print('Status Code: {} | Reason: {} \n'.format(PostRequest.status_code, PostRequest.reason))

# -------- Setting Routine ------------------

serverClient.on_connect = on_connect
serverClient.on_message = on_message
serverClient.on_log = on_log

serverClient.tls_set(ca_certs=rootca_path,
                     certfile=ca_path,
                     keyfile=key_path)

try:
    serverClient.connect(endpoint, port=8883, keepalive=60)
    time.sleep(1)
    serverClient.subscribe("FieldData/#")
    time.sleep(1)
    serverClient.loop_forever()


except Exception as e:
    print(e)

except KeyboardInterrupt:
    serverClient.loop_stop()
    serverClient.disconnect()
    print('Programa finalizado')
```

Apêndice C – Códigos da Aplicação Web

11.1 Estrutura de Diretório da Aplicação Web





11.2 Código dos principais componentes

Abaixo estão transcritos os códigos dos principais componentes da Aplicação Web:

- **reports.html**: código da página que gera as visualizações históricas dos dados;
- **views.py:** código que faz a consulta no Banco e retorno de dados e páginas para usuário;
- **realtime.html:** código da página que implementa a visualização em tempo real;
- **consumers.py:** código que gerencia a conexão com o Broker e respostas ao usuário.

11.2.1 webApp > api > templates > api > reports.html

```
<!-- Report Page -->

{% extends "base.html" %}

{% block title %}

  Query data

{% endblock title %}

{% block head %}

{% endblock head %}

{% block content %}

<style>
  :root{
    --var-height : 800px;
  }
  .main-container{
    /*border: 1px solid gray;*/
    /*margin: 10px;*/
```

```
    max-width: 1920px;
  }

  .col-body{
    /*border: 1px solid gray;*/
    height: 500px;
  }

  .menu-row{
    height: 100px;
  }

</style>

<!-- Style Maps Div -->
<style>
  #viewDiv {
    padding: 0;
    margin: 0;
    height: 100%;
    width: 100%;
  }
</style>

<!-- Request Data -->
<script>

  var charts = {{ charts|safe }};
  var v_lat = charts.Lat;
  var v_lon = charts.Lon;
  var v_dateHour = charts.DeviceTimestamp;
  var v_AltData = charts.MslAlt;

  for(var i = 0; i < v_lat.length; i++){
    v_lat[i] = parseFloat(v_lat[i]);
    v_lon[i] = parseFloat(v_lon[i]);
    v_dateHour[i] = moment(v_dateHour[i]);
    v_AltData[i] = parseFloat(v_AltData[i]);
  };

  var v_DataAltChart = [];

  for (var i = 0; i < v_AltData.length; i++){
    v_DataAltChart.push({
      x: v_dateHour[i],
      y: v_AltData[i]
    })
  };

  console.log(v_DataAltChart);
</script>

<!-- Container Principal da pagina -->
<div class="container main-container float-left">

<!-- Titulo -->
  <div class="row" style="margin-top: 20px">
    <div class="col-2">
      <h4>Data Reports</h4>
```

```
    </div>

    <div class="col-10">

      <!-- DJANGO FORMS
        Aplica filtros para extracao de dados-->

      <form action="/api/reports/" method="POST">
        {% csrf_token %}
        {{ forms }}
        <input type="submit" value="Gerar Relatorio">
      </form>
    </div>
  </div>


<!-- Conteudo -->
    <div class="col-12 col-body" style="margin-top: 10px">


<!-- Maps -->
      <div class="col-6 float-left" id="viewDiv"></div>
      <!-- {{charts}} -->
      <script>
        require([
          "esri/Map",
          "esri/views/MapView",
          "esri/Graphic"
        ],

        function(Map, MapView, Graphic) {
          var map = new Map({
            basemap: "topo-vector"
          });

          lastDataLen = v_lat.length;    //Data Array length

          var view = new MapView({
            container: "viewDiv",
            map: map,
            center: [ v_lon[lastDataLen - 1], v_lat[lastDataLen - 1]],
            zoom: 16
          });


// Insert markers to Map using the coordinates

          // Round marker
          var simpleMarkerSymbol = {
            type: "simple-marker",
            color: [226, 119, 40],
            outline:{
              width: 1,
              color: [255, 255, 255]
            }
          };

          // Line marker
          var simpleLineSymbol = {
```

```
    type: "simple-line",
    color: [226, 119, 40],
    width: 2
};

// Last location symbol
var lastLocSymbol = {
  type:"picture-marker",
  // url:"https://developers.arcgis.com/labs/images/bluepin.png",
  url: "https://image.flaticon.com/icons/svg/34/34468.svg",
  width: "40px",
  height: "40px"
};

linePath = []; //coordinates used to draw a line

for(var i=0; i < lastDataLen; i++){

  var point = {
    type:"point",
    longitude: v_lon[i],
    latitude: v_lat[i]
  };
  // Popup to show data in every point of the map
  var attributes = {
    date: v_dateHour[i].format("DD/MM/YY"),
    time: v_dateHour[i].format("hh:mm:ss"),
    altitude: v_AltData[i]
  };

  var popupTemplate = {
    title: "<h4>Ponto " + i + "</h4>" ,
    content: "<ul>" +
    "<li>Date: {date}</li>" +
    "<li>Time: {time}</li>" +
    "<li>Altitude: {altitude}</li>" +
    "</ul>"
  };

  //applies different marker in the last coordinate
  if (i==lastDataLen-1) {
   var pointGraphic = new Graphic({
    geometry: point,
    symbol: lastLocSymbol,
    attributes: attributes,
    popupTemplate: popupTemplate
  });
  } else {
    var pointGraphic = new Graphic({
    geometry: point,
    symbol: simpleMarkerSymbol,
    attributes: attributes,
    popupTemplate: popupTemplate
  });
  };

  view.graphics.add(pointGraphic);

  //Draw line
```

```
            linePath.push([v_lon[i], v_lat[i]]);

          };

          var polyline = {
            type: "polyline",
            paths: linePath
          };

          var polylineGraphic = new Graphic({
            geometry: polyline,
            symbol: simpleLineSymbol
          });

          view.graphics.add(polylineGraphic);
        });
      </script>


<!-- Altitude CHART -->
      <div class="col-5 float-left">
        <div class="card">
          <div class="card-header"><p>Temperatura</p></div>
          <div class="card-body">
            <canvas id="myChart" width="600" height="250"></canvas>
          </div>
        </div>
      </div>


      <script>
        var ctx = document.getElementById('myChart').getContext('2d');
        var myChart = new Chart(ctx, {
          type: 'line',
          data:{
            datasets: [{
              label: 'Altitude data [m]',
              showline: true,
              backgroundColor: 'hsla(227, 90%, 10%, 0.69)',
              data: v_DataAltChart
            }]
          },
          options: {
            scales: {
              xAxes: [{
                type: 'time',
                distribution: 'linear',
                time: {
                  unit: 'minute',
                  displayFormats:{
                    day: 'DD/MM HH:mm:ss'
                  }
                }

              }],
              yAxes: [{
                ticks: {
                  beginAtZero: false
                }
```

```
                }]
            }
        }
    });
  </script>
 </div>
</div>


{% endblock content %}
```

11.2.2 webApp > api > views.py

```
from django.shortcuts import render
from django.views.generic import TemplateView
from django.db.models import Q

from rest_framework import viewsets
from .models import Device, GpsGGA
from .serializers import GGASentence

from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User

from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy

import json
import datetime

# ----------------- REST APIs ---------------------------

class GGAView(viewsets.ModelViewSet):

   queryset = GpsGGA.objects.all()
   serializer_class = GGASentence

# ------------------------------------------------------
"""
Page used to extract data form Database using filter parameters passed by user
through django forms
"""

#Forms in Reporting page
from .forms import reportsForm, DeviceForm
import datetime

@login_required
def reports(request):
  if request.method == 'POST':
    reports_form = reportsForm(request.POST)

    if reports_form.is_valid():

      initial_date = datetime.datetime.strptime(request.POST['initial_date'], '%Y-%m-%d')
      end_date = datetime.datetime.strptime(request.POST['end_date'], '%Y-%m-%d')

      print(initial_date)

      end_date = end_date + datetime.timedelta(hours=23, minutes=59, seconds=59)

      print(end_date)

      dataQuery = GpsGGA.objects.filter(DeviceID=request.POST['deviceId'],
                  DeviceTimestamp__gte=initial_date,
                  DeviceTimestamp__lte=end_date)
```

```
        dataQuery = dataQuery.exclude(Q(GPSQual=None) | Q(GPSQual=0))
        dataQuery = dataQuery.order_by('DeviceTimestamp')

        # Data to render charts
        charts = {
          'ServerTimestamp': [],
          'DeviceTimestamp': [],
          'UtcTime': [],
          'Lat': [],
          'LatDirection': [],
          'Lon': [],
          'LonDirection': [],
          'GPSQual': [],
          'Hdop': [],
          'MslAlt': [],
        }

        for data in dataQuery:

          charts['Lat'].append(str(data.Lat))
          charts['Lon'].append(str(data.Lon))
          charts['MslAlt'].append(str(data.MslAlt))
          charts['GPSQual'].append(data.GPSQual)
          charts['DeviceTimestamp'].append(data.DeviceTimestamp.strftime('%Y-%m-%dT%H:%M:%S'))

        charts = json.dumps(charts)
        print("charts: ", charts)


        return render(request, 'reports.html', {'forms': reports_form,
                                'queryset': dataQuery,
                                'charts': charts})
    else:
      reports_form = reportsForm()
      return render(request, 'reports.html', {'forms': reports_form})



# ----------------------Real Time ----------------------------------
@login_required
def realtime(request):

    return render(request, 'realtime.html', {})


# ---------------------- Devices ----------------------------------

# Query Database to collect all registered Devices
# Class Base View

# @login_required
class DeviceListView(ListView):
    model = Device

# @login_required
class DeviceCreate(CreateView):
    model = Device
    fields = ['deviceId']
    success_url = reverse_lazy('device-view')
```

```
# @login_required
class DeviceUpdate(UpdateView):
    model = Device
    fields = ['deviceId']
    success_url = reverse_lazy('device-view')

# @login_required
class DeviceDelete(DeleteView):
    model = Device
    success_url = reverse_lazy('device-view')

# -------------------------- Users ---------------------------------------
@login_required
def users(request):
   users = []
   queryset = User.objects.all().values('username',
                        'email',
                        'is_superuser',
                        'is_active',
                        'last_login',
                        'date_joined')
   for query in queryset:
      users.append({
         'username': query['username'],
         'email': query['email'],
         'is_superuser': query['is_superuser'],
         'is_active': query['is_active'],
         'last_login': query['last_login'].isoformat(),
         'date_joined': query['date_joined'].isoformat(),
         })

   print(json.dumps(users))

   return render(request, 'users.html', {"users": json.dumps(users)})
```

11.2.3 webApp > RealTime > templates > realtime.html

```
<!-- Report Page -->

{% extends "base.html" %}

{% block title %}

  Real time Application

{% endblock title %}

{% block head %}

<!-- ############################## SCRIPTS ###################################
-->

<script>

// Variables to handle data
var v_lat = [];
var v_lon = [];
var v_dateHour = [];
var v_hour;
var v_alt;
var v_coordinates = [];    //used in maps
var ult = 0;
var v_temp;

// Variables for Charts
var v_chartData_alt = [];  //Altitude
// var v_chartData_alt;


// -------------------------Data Manipulations-------------------------------------
// Used to conform data in the format necessary to render in charts and map application
function handle_data(arg){

  // console.log('ult variable:' + ult);

  v_temp = JSON.parse(arg);

  v_temp.DeviceTimestamp = moment(v_temp.DeviceTimestamp);

  v_lat.push(parseFloat(v_temp.Lat));
  v_lon.push(parseFloat(v_temp.Lon));
  v_dateHour.push(v_temp.DeviceTimestamp);

  console.log('v_dateHour:' + v_dateHour[ult]);

  v_chartData_alt.push({
    x: v_temp.DeviceTimestamp,
    y: v_temp.MslAlt
  });

};
```

```
//------------------------ Maps----------------------------------

function mapFunc(event){

  require([
    "esri/Map",
     "esri/views/MapView",
     "esri/Graphic",
     "esri/layers/GraphicsLayer",
  ],
  function(Map, MapView, Graphic, GraphicsLayer){

    var map = new Map({
      basemap: "topo-vector"
    });

     // console.log("Qntde dados: " + ult);

    var v_center = [-47.898757, -22.002804];   //default center map
    var v_zoom = 7;

     var view = new MapView({
       container: "mapsDiv",
       map: map,
       center: v_center,
       zoom: v_zoom
     });

     var layer = new GraphicsLayer({
       graphics: []
     });

    // Insert markers to Map using the coordinates

    // Round marker

    var symbol = {

      simpleMarker: {
         type: "simple-marker",
         color: [226, 119, 40],
         outline:{
           width: 1,
           color: [255, 255, 255]
         }
       },

       simpleLine: {
          type: "simple-line",
          color: [226, 119, 40],
          width: 2
       },

       lastLocSymbol: {
         type:"picture-marker",
         // url:"https://developers.arcgis.com/labs/images/bluepin.png",
         url: "https://image.flaticon.com/icons/svg/34/34468.svg",
         width: "40px",
```

```
      height: "40px"
    }
};

 var linePath = []; //coordinates used to draw a line

 var point = {
   type: "point",
   longitude: null,
   latitude: null
 };

 var pointGraphic = [];

 var attributes = {
   date: null,
   time: null
 };

 var popupTemplate = {
   title: "<h4>Ponto " + ult + "</h4>" ,
   content: "<ul>" +
   "<li>Date: {date}</li>" +
   "<li>Time: {time}</li>" +
   // "<li>Altitude: {altitude}</li>" +
   "</ul>"
 };

 // Instancing Draw Line
 var polyline = {
   type: "polyline",
   paths: linePath
 };

var polylineGraphic = new Graphic({
  geometry: polyline,
  symbol: symbol.simpleLine
});


 function addPoint(lat, lon, DeviceTimestamp){

   var point = {
     type:"point",
     longitude: lon,
     latitude: lat,
   };
   // Popup to show data in every point of the map
   var attributes = {
     date: DeviceTimestamp.format("DD/MM/YY"),
     time: DeviceTimestamp.format("hh:mm:ss"),
     // altitude: v_AltData[i]
   };



   //applies different marker in the last coordinate
   if (ult != 0) {
```

```
    layer.graphics.remove(pointGraphic[ult-1]);

    pointGraphic[ult-1].symbol = symbol.simpleMarker;

    layer.graphics.add(pointGraphic[ult-1]);

  };

  point.longitude = lon;
  point.latitude = lat;

  pointGraphic[ult] = new Graphic({
    geometry: point,
    symbol: symbol.lastLocSymbol,
    attributes: attributes,
    popupTemplate: popupTemplate
  });

  layer.graphics.add(pointGraphic[ult]);

  //Draw line
  linePath.push([lon, lat]);
  polylineGraphic.geometry = polyline;
  view.graphics.add(polylineGraphic);

  map.add(layer);


  view.goTo({
    center: [lon, lat],
    zoom: 15,
  },
  {
    duration:300
  });

 view.watch("animation", function(response){
   if(response && response.state === "running"){
     console.log("Animation in progress");
   }
   else{
     view.goTo({
       center: [lon, lat],
       zoom: 16,
     },
     {
       duration: 500
     });
   }
 });

};


//---------------------------- Charts -------------------------------------

//Altitude Chart

var ctx = document.getElementById('altChart').getContext('2d');
```

```
    var altChart = new Chart(ctx, {
      type: 'line',
      data:{
        datasets: [{
          label: 'Altitude data [m]',
          showline: true,
          backgroundColor: '#A241',
          data: v_chartData_alt
        }],
        labels: []
      },
      options: {
        scales: {
          xAxes: [{
            type: 'time',
            distribution: 'linear',
            time: {
              unit: 'minute',
              displayFormats:{
                day: 'DD/MM HH:mm:ss'
              }
            }

          }],
          yAxes: [{
            ticks: {
              beginAtZero: false
            }
          }]
        }
      },
    });

    // ------------------------Websockets-----------------------------------

    var event = event;
    // console.log(event);
//    });
// };
    // Websocket to open communication with AWS IoT Broker and transmit data in real time
    function websocket_initiate(event){
    // function (event){
      var selectedDevice = document.getElementById('deviceSelection').value;
      var socket = new WebSocket('ws://' + window.location.host + '/ws/realtime/home');

      socket.onopen = function() {
        alert("Connection established with Device: " + selectedDevice);
        socket.send("Starter;" + selectedDevice);
        console.log('Starter sent');
        // socket.send("Close"); To use in future, implement another action starter
        console.log('Starting program');
      };
      socket.onmessage = function(event){
        console.log('Message received at:' + new Date());
        console.log(event.data);

        handle_data(event.data);

        addPoint(v_lat[ult], v_lon[ult], v_dateHour[ult]);
```

```
          // addData(altChart, data=v_chartData_alt);    //atualiza altChart
          altChart.update();
          document.getElementById('dataCard-x').innerHTML = v_dateHour[ult];

          ult += 1;
        };
        socket.onclose = function(event) {
          socket.send("Closer");
        };

    };
    websocket_initiate();
  // };
  });    // Map Function close
};

</script>



{% endblock head %}

{% block content %}

<style>
  :root{
    --var-height : 1000px;
  }
  .main-container{
/*    border: 1px solid gray;*/
    /*margin: 10px;*/
    max-width: 1920px;
  }

  .col-body{
    /*border: 1px solid gray;*/
    height: var(--var-height);
  }

  .menu-row{
    height: 100px;
  }

  .content-container{
    overflow: hidden;
    height: 600px;
  }

</style>

<script>
  // Parser function to handle Object

</script>



<!-- <div class="container float-left"> -->
```

```
<div class="container main-container float-left">

<!-- Titulo -->
  <div class="row" style="margin-top: 20px">
    <div class="col-sm-2 col-md-2">
      <h4>Real Time Report</h4>
    </div>

    <div class="col-sm-9">


        <div class="col-2 float-left">
          <span class="align-right">Select a Device</span>
        </div>

        <div class="col-sm-3 col-md-4 float-left">
          <div class="input-group float-left" style="outline-offset: 10px">
            <!-- <select class="custom-select" id="deviceSelection" aria-label="Example select with button
addon" onchange="printSelectDevice();"> -->
            <select class="custom-select" id="deviceSelection" aria-label="Example select with button
addon">
              {% for device in devices %}
              <option value={{device.deviceId}}>{{device.deviceId}}</option>
              {% endfor %}
            </select>
            <div class="input-group-append">
              <!-- <button class="btn btn-primary" type="button" onclick="websocket_initiate();">Start
Application</button> -->
              <button class="btn btn-primary" type="button" onclick="mapFunc();">Start
Application</button>
            </div>
          </div>
        </div>
    </div>
  </div>


  <div class="col-12 wrapper-container float-left">
  <!-- Conteudo -->
  <div class="col-6 content-container float-left" id="mapsDiv" style="margin-top: 30px">


  </div>

  <div class="col-5 float-left content-container" style="margin-top: 30px">
    <!-- Data Card -->
    <div class="card" style="height: 600px">
      <div class="card-header">
        Data
      </div>
      <div class="card-body">
        <blockquote class="blockquote mb-0" id="dataCard-x">

<!--            <footer class="blockquote-footer">Someone famous in <cite title="Source Title">Source
Title</cite></footer> -->
        </blockquote>
      </div>

      <div class="container chart-container">
```

```
        <canvas id="altChart" width="500" height="350"></canvas>
      </div>

    </div>

  </div>
  </div>

{% endblock content %}
```

11.2.4 webApp > RealTime > consumers.py

```
# websocket/consumers.py
from asgiref.sync import async_to_sync
from channels.generic.websocket import WebsocketConsumer
import json
import time, datetime
import paho.mqtt.client as mqtt
import pynmea2

from channels.consumer import SyncConsumer
import json


# # MQTT Server Endpoint

endpoint = "a9s9ibiau5ux0-ats.iot.sa-east-1.amazonaws.com"

port = 8883

# Defining relative path to certificates and keys

rootca_path = '../Certificates/testClient/AmazonRootCA1.pem'  # PC
key_path = '../Certificates/testClient/96b663ac56-private.pem.key'
ca_path = '../Certificates/testClient/96b663ac56-certificate.pem.crt.txt'


# ----------------Call back function for MQTT Paho------------------

def on_connect(client, userdata, flags, rc):

    # Function to print Message on connection attempt
    if rc == 0:
        print("Connection returned result: ", rc)
    else:
        print('Connection Failed with code: ', rc)

def on_log(client, userdata, level, buf):
    # Callback function to print log messages
    print("log: "+buf)


# ---------- handle Message Sent by Device ---------------
import pdb

def handle_msg_data(msg):

    ServerTimestamp = datetime.datetime.now().isoformat()

    v_Message = msg.payload.decode("utf-8")

    DeviceID, DeviceTimestamp, payload = v_Message.split(';')

    payload = pynmea2.parse(payload)

    PostPayload = {
        'DeviceID': DeviceID,
```

```
        'ServerTimestamp': ServerTimestamp,
        'DeviceTimestamp': DeviceTimestamp,
        'UTCTime': payload.timestamp.isoformat(),
        'Lat': payload.latitude,
        'LatDirection': payload.lat_dir,
        'Lon': payload.longitude,
        'LonDirection': payload.lon_dir,
        'GPSQual': payload.gps_qual,
        'NumSats': payload.num_sats,
        'Hdop': payload.horizontal_dil,
        'MslAlt': payload.altitude,
        'MslAltUnit': payload.altitude_units,
        'GeoidSeparation': payload.geo_sep,
        'GeoidSeparationUnit': payload.geo_sep_units,
        'AgeLastDgpsUpdate': payload.age_gps_data,
        'DgpsStationID': payload.ref_station_id
    }

    # print("\nPayload String:", PostPayload)

    return PostPayload

# ----------------Consumer--------------------------------

class MQTTConsumer(WebsocketConsumer):

    def websocket_connect(self, event):

        self.accept()
        # self.send({
        #     "type": "websocket.accept",
        # })

    def websocket_receive(self, event):

        receivedMsg = event['text'].split(',')
        print(receivedMsg)

        # Callback function to received messages on subscription
        def on_message(client, userdata, msg):

            print('\nReceiving new MQTT message')

            # print("msg: ", msg.payload.decode("utf-8"))

            payload = handle_msg_data(msg)

            if payload['GPSQual'] == 1:

                payload = json.dumps(payload)

                print('\nSending MQTT Message via Websocket')

                # Send to Browser
                self.send(text_data = payload)
            else:

                print('\nGPS Value Invalid')
```

```
    if receivedMsg[0] == "Starter":    # Implement MQTT subscription

        #MQTT-------------------------
        global mqttClient

        ConsumerId = time.time()
        mqttClient = mqtt.Client('DjangoConsumer' + str(ConsumerId))

        # Callback functions
        mqttClient.on_connect = on_connect
        mqttClient.on_log = on_log
        mqttClient.on_message = on_message

        mqttClient.tls_set(ca_certs=rootca_path,
                           certfile=ca_path,
                           keyfile=key_path)

        mqttClient.connect(endpoint, port=8883, keepalive=60)
        time.sleep(1)
        mqttClient.subscribe("FieldData/" + receivedMsg[1])
        time.sleep(1)
        mqttClient.loop_start()

    def websocket_disconnect(self, event):
        mqttClient.loop_stop()
        mqttClient.disconnect()
        self.close()
```