UNIVERSIDADE ESTADUAL DA PARAÍBA

CAMPUS I – CAMPINA GRANDE

CENTRO DE CIÊNCIA E TECNOLOGIA

CURSO DE BACHARELADO EM COMPUTAÇÃO

EDSON SALES DE SOUZA NETO

DESENVOLVIMENTO DE UM COMPONENTE SERVIDOR PARA GERENCIAMENTO DE DADOS MÉDICOS

Campina Grande

2018

EDSON SALES DE SOUZA NETO

DESENVOLVIMENTO DE UM COMPONENTE SERVIDOR PARA GERENCIAMENTO DE DADOS MÉDICOS

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba como requisito à obtenção do título de Bacharel em Ciência da Computação.

Área de concentração: Internet das Coisas

Orientador: Prof. Dr. Paulo Eduardo e Silva Barbosa.

Campina Grande

2018

S719d Souza Neto, Edson Sales de.

Desenvolvimento de um componente servidor para gerenciamento de dados médicos [manuscrito] : / Edson Sales de Souza Neto. - 2018.

59 p. : il. colorido.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2018.

"Orientação : Prof. Dr. Paulo Eduardo e Silva Barbosa , Coordenação do Curso de Computação - CCEA."

1. Internet das coisas. 2. Programação para aplicações. 3. Cuidados médicos. 4. Gerenciamento de informações. I. Título

21. ed. CDD 003.5

Elaborada por Camile de A. Gomes - CRB - 15/559 BC/UEPB

Edson Sales de Souza Neto

DESENVOLVIMENTO DE UM COMPONENTE SERVIDOR PARA GERENCIAMENTODE DADOS MÉDICOS

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 27 de Junho de 2018.

Prof. Dr. Paulo Eduardo e Silva Barbosa (UEPB)
Orientador(a)

Profa. Dra. Sabrina de Figueiredo Souto (UEPB)
Examinador(a)

Profa. Dra. Kézia de Vasconcelos Oliveira Dantas (UEPB)
Examinador(a)

AGRADECIMENTOS

Agradeço, inicialmente, a Deus por ter me abençoado durante todo o curso, me enchendo de coragem, perseverança, paciência e dedicação.

A toda minha família, minha mãe, Claudia Maria Barbosa Sales de Souza, meu pai, Francisco Alves de Souza Neto, meu irmão, Arthur Barbosa Sales de Souza, por me apoiarem em diversos momentos, me motivando, dando ensinamentos, sempre se preocupando com meu bem-estar e desempenho no decorrer da graduação. Um agradecimento especial a minha avó, Maria Amélia Barbosa, por me acolher em sua casa, dando total suporte.

A minha namorada, Beatryz da Nóbrega Marques, por toda a atenção, carinho e auxilio, principalmente, nas questões acadêmica que foram de fundamental importância para meu crescimento na universidade.

Ao professor Paulo Eduardo e Silva Barbosa, pelas oportunidades que me proporcionou, juntamente, com sua orientação e amizade durante meu processo de formação acadêmica tanto como coordenador quanto professor de projetos e monitoria.

Aos meus amigos de sala, por compartilharem momentos marcantes nessa fase da minha vida, sendo meus companheiros de trabalho, estudos e projetos, destacando, Douglas Rafael, por fornecer sua experiência mais avançada em diferentes trabalhos, essencialmente, este trabalho.

E aos demais que contribuíram de forma direta, ou indireta para minha formação profissional e pessoal.

RESUMO

A inclusão dos avanços tecnológicos nos cuidados médicos à saúde proporcionou melhorias na adesão à tratamentos de doenças crônicas por parte dos pacientes pois a tecnologia fornece uma comodidade tanto para o profissional quanto para o paciente possibilitando acompanhamento integral. Isso advém de sistemas que utilizam a Internet das coisas para obter controle sobre uma rede de diferentes dispositivos médicos conectados, comunicando entre si e trocando uma grande quantidade de dados. Inserido nesse contexto, diversas soluções surgiram, dentre elas o HANIoT (acrônimo de *Health* Analytics Internet Of Things), um sistema genérico que tem como objetivo analisar e auxiliar nos diferentes contextos *Healthcare* utilizando a internet das coisas como solução. Este trabalho busca uma forma de centralizar os dados gerenciados pelo *HANIoT* proporcionando uma melhor distribuição das informações. Todo o processo de desenvolvimento se baseou na implementação de uma Interface de Programação para aplicações (API – *Application Programming Interface*) que fosse responsável por fornecer os serviços de aquisição e gerenciamento de dados no Sistema *HANIoT.* Com isso, foi possível integrar a aplicações diferentes a um ambiente único.

Palavras chave: Internet das coisas, *Health Care, HANIoT*

ABSTRACT

The inclusion of technological advances in medical health care has resulted in improvements in patients' adherence to chronic disease treatments, since the technology provides a comfort for both the professional and the patient, enabling integral follow-up. This comes from systems that use the Internet of Things to gain control over a network of different connected medical devices, communicating with each other and exchanging a large amount of data. HANIoT (acronym for Health Analytics Internet Of Things), a generic system that aims to analyze and assist in different contexts Healthcare using the internet of things as a solution. This work looks for a way to centralize the data managed by HANIoT providing a better distribution of the information. The entire development process was based on the implementation of an API (Application Programming Interface) that was responsible for providing the data acquisition and management services in the HANIoT System. With this, it was possible to integrate different applications into a single

Keywords: Internet of Things, *Health Care*, *HANIoT*

LISTA DE ILUSTRAÇÕES

LISTAS DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
HANIoT	*Health* Analytics Internet of Things
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
NPM	*Node Package Manager*
NUTES	Núcleo de Tecnologias Estratégicas em Saúde
OCARIOT	*Smart Childhood Obesity Caring Solution using IoT potential*
PHD	*Personal Health Device*
US	*User Story*

SUMÁRIO

1. INTRODUÇÃO

1.1 Motivação

Com a ascensão tecnológica, várias áreas do conhecimento romperam suas limitações inovando através dos recursos tecnológicos avançados. Dentre as diversas áreas, a tecnologia afetou significativamente a saúde em diferentes eixos, tais como: comunicação, tratamento, pesquisa e coleta de informações. No decorrer dessa ascensão, os avanços impactaram na qualidade geral dos cuidados à saúde para pacientes e médicos por todo o mundo.

As informações coletadas utilizadas como base para os recursos tecnológicos voltados para o cuidado à saúde têm, em resumo, dois tipos de fontes: a científica e os dados dos usuários. A primeira advém das pesquisas publicadas que são realizadas por profissionais, estudantes e instituições da área. Já a segunda, principalmente nos últimos anos, são os dados fornecidos por usuários que utilizam aplicativos de cuidados a saúde (*Health care*) que contribuem, direta ou indiretamente, para a comunidade. Como consequência, os aplicativos mantem um histórico de dados de cada usuário, auxiliando no diagnóstico e acompanhamento de doenças e/ou adesão ao tratamento, facilitando, assim, o trabalho dos profissionais de saúde pelo acesso a informação com maior controle e de forma segura.

As atividades executadas por aplicativos, com esses diferentes tipos de dados, devem atender às características de uma estrutura *Healthcare*, definida como uma conexão entre uma coleção de dispositivos médicos, aplicações e sistemas de TI de saúde por meio das redes de computadores disponíveis no contexto (Oliveira, 2017). Logo, as estruturas *Healthcare* contam com *Hardwares* e Softwares específicos que proporcionam uma interoperabilidade de qualidade e tornam o fluxo de dados e seu gerenciamento seguro e rápido.

Em virtude disso, instituições e organizações planejam projetos que aplicam a estrutura *Healthcare* em seu escopo com o objetivo de diagnosticar, prevenir ou tratar doenças crônicas. Dentre elas, doze instituições do Brasil e da Europa se reuniram no projeto *Solução Inteligente de Tratamento da Obesidade Infantil por Meio do Potencial*

da Internet das Coisas, também conhecido como OCARIoT[1]. O projeto começou em novembro de 2017 e tem duração de três anos e pretende desenvolver aplicativos associados a sensores, para monitorar em tempo real informações do dia a dia das crianças – atividades física realizadas, batimentos cardíacos, queima calórica, refeições etc. – tendo como base uma rede de IoT (RNP - Rede Nacional de ensino e pesquisa).

1.2 Proposta do Sistema

Diante dessas circunstâncias, o Núcleo de Tecnologia Estratégicas em Saúde (NUTES) planejou um sistema que fornece soluções a requisitos baseados em diferentes cenários e projetos *Heatlhcare* – nomeada como *HANIoT* (*Health ANalytics Internet of Things*). Planejada não só para capturar e analisar dados médicos como também auxiliar nos diagnósticos e outros dilemas da saúde, o *HANIoT* se concretizou após o NUTES ingressar no projeto Ocariot, juntamente com outras instituições brasileiras e europeias.

A estrutura básica do *HANIoT* era de uma ferramenta responsável por coletar dados de determinados dispositivos médicos pessoais, armazenando-os na base de dados local de um smartphone. Com o ingresso do NUTES no Ocariot, o Haniot passou a adquirir novos requisitos para serem incorporados, resultando em um aspecto de sistema *Healthcare* complexo.

Dentro deste contexto, este trabalho de conclusão de curso propõe, através do Programa Institucional de Bolsas de Iniciação Cientifica (PIBIC), desenvolver uma solução para garantir a evolução do *HANIoT* como um sistema *Healthcare* possibilitando o controle e análise dos dados armazenados no smartphone mais complexo e com um poder de processamento maior que um simples dispositivo móvel. Para isso, o programa incentivou o desenvolvimento de um componente servidor que se responsabilizasse em desempenhar esse papel, aumentando o desempenho do sistema e distribuindo as obrigações para diferentes elementos dentro do *HANIoT.*

[1] ocariot.com

1.3 Objetivos

1.3.1 Objetivos Gerais

Este trabalho de conclusão de curso visa desenvolver um componente servidor para o gerenciamento dos dados capturados pela aplicação cliente do sistema HANIoT.

1.3.2 Objetivos Específicos

- Persistir as informações em uma base de dados centralizada em um servidor na nuvem, comportando uma estrutura de dados expansível;
- Disponibilizar os serviços de armazenamento de dados, gerenciamento e manipulação para outras aplicações de acordo com o tipo de autenticação;
- Autenticação de usuário responsável por controlar o acesso aos serviços de acordo com o nível de permissão especificado para um usuário;
  Histórico de armazenamento na base de dados por meio de filtros referentes a estrutura do dado.

1.4 Estrutura do Trabalho

Este trabalho é dividido nos seguintes capítulos:

Este trabalho está dividido em:

- Referencial Teórico: aborda sobre os fundamentos dos principais elementos aplicados tanto no processo de desenvolvimento quanto na estruturação da solução.
- Metodologia: Capítulo onde são apresentados cada artefato gerado pelas fases metodológicas e métodos empregados no desenvolvimento do componente servidor.
- Desenvolvimento do Componente: Capítulo responsável por detalhar o desenvolvimento do componente, descrevendo a construção de cada funcionalidade.
- Validação e Resultados: Breve apresentação da integração do componente servidor com o sistema *HANIoT.*

- Conclusão: O capítulo apresenta as principais conclusões obtidas e propostas para futuros trabalhos.

2. REFERENCIAL TEÓRICO

Este capítulo é responsável por apresentar os principais fundamentos utilizados na construção da solução. Os tópicos em sequência abordarão sobre os conceitos a respeito da estrutura de armazenamento dos dados médicos; os detalhes sobre a importância da internet das coisas no escopo do projeto; a concepção do processo de desenvolvimento da aplicação; e uma explicação dos padrões de programação adotados.

2.1. Banco de dados *NoSQL*

O *NoSQL* é um paradigma de banco de dados, conhecido também como banco de dados não relacional, cuja terminologia foi definida pela primeira vez por Carlos Strozzi em 1998, que alegou sobre o movimento *NoSQL* "é completamente distinto do modelo relacional [...]". Tal movimento tem como objetivo resolver o problema de escalabilidade dos bancos tradicionais, pois os custos e a complexidade de escalar um banco SQL são elevados.

As principais características do *NoSQL* são:

- Não relacional – significa que os bancos *NoSQL* armazenam os dados de forma mais isolada. Essa qualidade não exclui todo o relacionamento dessa estrutura, porém permite um melhor desempenho em aplicações com um grande volume de dados, pois os relacionamentos podem comprometer determinadas funcionalidades que exigem um tempo de resposta alto.
- Esquemas de dados flexíveis: contrário aos relacionais, os registros de dados não relacionais podem ser feitos em formatos diferentes sem acarretar em problemas na arquitetura, ou seja, as diferentes propriedades de um registro de dados armazenado não afetam os demais.
- Diferentes modelos/formas de armazenamento de dados – os bancos *NoSQL* não são todos iguais na estrutura de armazenamento nem nos conceitos de

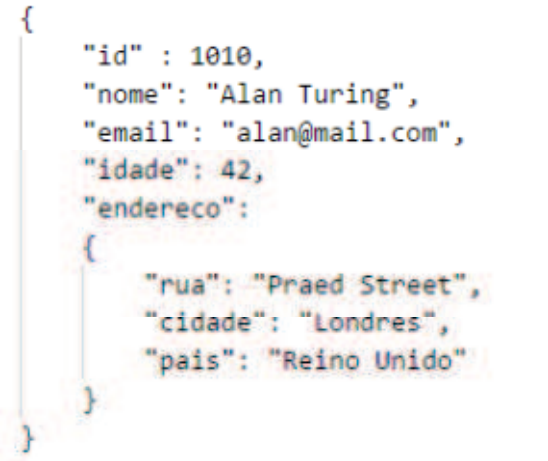
modelagem. Cada banco possui sua técnica em particular como detalhado mais a seguir.

MongoDB

MongoDB[2] é uma aplicação de código aberto, escrito em C++[3], criado para banco de dados *NoSQL*, orientado a documentos que fornece um ótimo desempenho, de alta disponibilidade (*High Availability*) e com escalabilidade automatizada.

Um registro no MongoDB é um documento que é composta, como ilustrado na figura 1, de pares de campos e valores. Os documentos são semelhantes aos objetos JSON (JavaScript *Object Notation* – Notação de Objetos JavaScript), onde os valores dos campos podem incluir outros documentos, matrizes e matrizes de documentos. O uso desse tipo de estrutura de dados tem como vantagens a correspondência a tipos de dados nativos em muitas linguagens de programação, redução da necessidade de *joins* (ação em bancos SQL) por causa da incorporação dos *arrays* e outros documentos, esquema dinâmico com suporte a polimorfismo fluente.

Figura 1 Estrutura de um documento JSON



Fonte: Própria autoria

Vantagens e características do MongoDB:

[2] mongodb.com
[3] Linguagem de programação multiplataforma e multiparadigma

- Linguagem de consulta avançada – O mongoDB suporta uma linguagem de consulta avançada para suportar operações de leitura e gravação (CRUD), bem como: agregação de dados e pesquisa de texto e consultas geoespaciais.
- Alta performance – Fornece alta performance na persistência de dados dando suporte para modelos de dados incorporados e reduzindo a atividade de I/O no sistema de banco de dados.
- Escalabidade horizontal – Principal característica do mongoDB. Fácil escalonamento utilizando um método para distribuir dados em várias máquinas.
- Atualizações rápidas no local
- Replicação e alta disponibilidade
- Não há necessidade de mapear ou converter objetos de uma aplicação para objetos de banco de dados.

Motivos para abdicar ao uso do MongoDB:

- Aplicações com Big Data
- Necessidade de gerenciamento de conteúdo e entrega
- Infraestrutura móvel e social
- Gerenciamento de dados do usuário
- Hub de dados

Por essas motivações, o MongoDB se enquadra no escopo desse projeto, pois atende as necessidades presentes no gerenciamento dos dados feito pelo sistema em que os dados podem ser acumulados e analisados em grande escala.

2.2. Internet das Coisas

Figura 2: Blocos básicos da IoT

Fonte: (Internet das Coisas: da Teoria à Prática,

O termo internet das coisas ou IoT (acrônimo do inglês *Internet of Things*), conforme (Singer, 2012), é utilizado para designar processos que envolvam objetos conectados em rede e que produzam e/ou processem informações em tempo real e de forma autônoma.

Esses processos que envolvem os objetos conectados em rede são definidos por (Internet das Coisas: da Teoria à Prática, 2016, p. 5) como “Blocos básicos de construção da IoT”. Cada bloco tem seu papel importante na composição da IoT, dispondo de tecnologias para viabilizar a integração dos objetos no ambiente físico ao mundo digital.

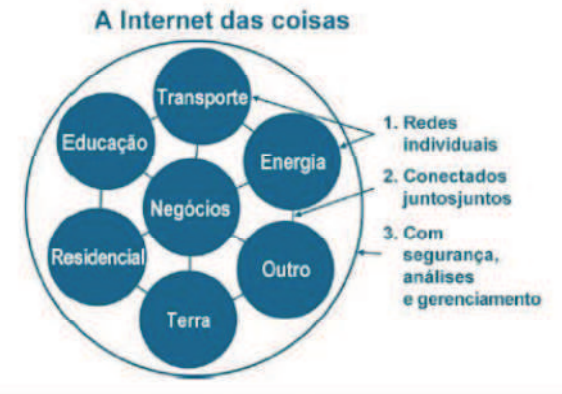
A responsabilidade de cada bloco no processo de construção da IoT está ilustrada pela figura 5, onde:

- Identificação – identifica os objetos unicamente para conectá-los à internet
- Sensores/Atuadores – se responsabiliza em coletar e armazenar/encaminhar os dados.
- Comunicação – está atrelado aos meios tecnológicos utilizados na conexão dos objetos inteligentes.
- Computação – trata da capacitação computacional dos objetos inteligentes na carência do contexto que está inserido.
- Serviços – destaca as diversas classes de serviços providos pela IoT.

- Semântica – refere-se à habilidade de extração de conhecimento dos objetos na IoT.

Todo o processo está inserido no modelo básico de arquitetura (figura 7) representado por três camadas: Camada de percepção, onde se encontra o bloco de sensores; Camada de rede, em que se é estabelecido a comunicação, efetua-se a identificação e todo o serviço de gerenciamento; por último, a Camada de aplicação, incumbida a apresentar as informações ao cliente e permite que ele interaja com toda a arquitetura de forma abstrata.

Figura 3: A IoT ser vista como uma rede das redes



Fonte: Cisco IBSG, abril de 2011

Figura 4: Arquitetura para IoT

Fonte: (Internet das Coisas: da Teoria à Prática,

Uma coleção livre de redes diferentes e criadas para determinada finalidade compõe a IoT. Como especificado na figura 6, cada contexto tem sua configuração de rede para responder as suas exigências constituindo uma IoT como uma rede das redes (Evans, 2011).

Tendo ciência de todo o cenário da Internet das coisas, esse projeto se encontra nas camadas mais acima da arquitetura IoT. Nele é estabelecido uma comunicação com os dispositivos responsáveis por coletar as informações e efetua o tratamento dos dados para serem gerenciados por futuros módulos de análise de dados. Sua versão atual serve como um componente de integração das redes da IoT.

2.3. Application Programming Interface (API)

Uma Interface de Programação de Aplicativos (em português) é software intermediário que faz com que duas aplicações se comuniquem. Basicamente, a API fornece um conjunto de especificações sobre a lógica de como os componentes de uma aplicação interagem. Através disso, as funcionalidades de uma aplicação podem ser aproveitadas por terceiros servindo como complemento.

A API se responsabiliza em disponibilizar essas funcionalidades para diferentes clientes com acesso permitido as especificações. O cliente solicita um serviço e a API retorna com uma resposta gerada a partir da manipulação do que foi solicitado. Entretanto, para que esse consumo por parte do cliente seja bem-sucedido, ele deverá cumprir com os padrões exigidos pela API como rotinas e protocolos específicos. Além disso, a respostas de algumas APIs têm também seus critérios.

Há diferentes tipos de APIs para sistemas operacionais, aplicações e websites. O Windows API, por exemplo, os programas desenvolvidos para o sistema operacional Windows interagem com essa API que facilita e padroniza a construção das aplicações. Outra API bem famosa é a Google Maps API, a qual permite os clientes adicionarem as funcionalidades do Google Maps em seus websites, aplicações mobiles, entre outras, utilizando uma interface para JavaScript.

A importância de uma Interface de programação de aplicativos está relacionada com a distribuição das funcionalidades contidas no projeto em questão, com o intuito de centralizar os serviços fornecidos. Nisso, aplicações client-side podem consumir tais serviços sem ter a necessidade de implementá-los.

2.5. Modelo de processo ágil – Scrum

Criado por Jeff Sutherland e sua equipe, na década de 90, o Scrum, cujo etimologia é decorrente de uma atividade em uma partida de rúgbi, é uma metodologia extremamente ágil e flexível aplicada no desenvolvimento de qualquer produto ou gerenciamento de qualquer trabalho. Como dito por (Pressman, 2011), os princípios do Scrum são consistentes com o manifesto ágil e incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega.

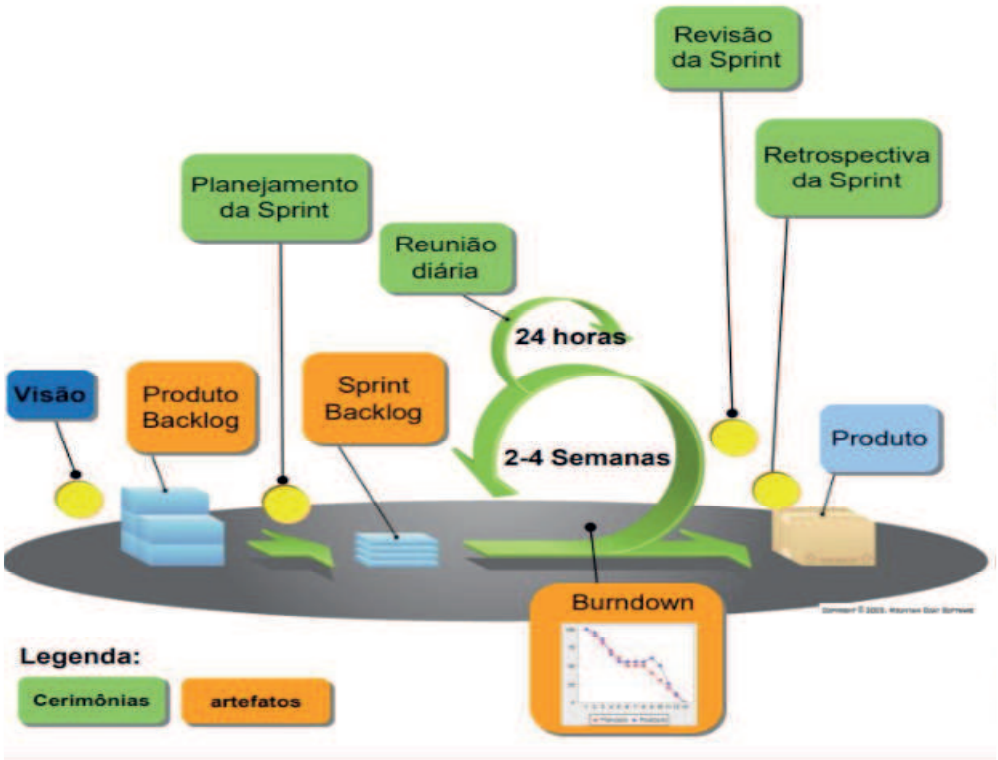
Todo o processo iterativo e incremental do Scrum, de acordo com (Sommerville, 2011), é dividido em três fases:

1ª Fase: Planejamento geral – Nessa fase inicial, a equipe se reúne para estabeler os objetivos gerais do projeto e definir dos requisitos gerando um Product Backlog;

2ª Fase: Ciclos de Sprint – Uma Sprint é uma iteração composta por um objetivo vindo dos requisitos, um período que não ultrapassa 30 dias e um conjunto de cerimônias, detalhados mais a frente;

3ª Fase: Encerramento do projeto – Na última fase do projeto são produzidos materiais instrucionais do produto. A execução dessas fases ocorre dentro de uma Arquitetura Scrum Framework, bem definida, composta por um conjunto de elementos peculiares que integrados realizam o processo. Para ter uma visão melhor desse Framework e seus elementos, a *figura 5* ilustra com detalhes todo o conteúdo da arquitetura.

Figura 5 Arquitetura do Scrum



Fonte: (SCRUM Experience (O tutorial SCRUM), 2009)

A partir dessa ilustração consegue-se notar que as cerimônias e artefatos apontados pela imagem são os alicerces do Scrum. São colocados em práticas e produzidos por três grupos diferentes e seus respectivos papeis, em que cada grupo tem seu papel dentro da arquitetura. São eles:

- *Product Owner* (Dono do Produto) – conhecido como cliente é responsável não só por definir uma visão do produto elaborando e mantendo o Product Backlog como também por aceitar ou rejeitar os entregáveis.
- *Scrum Master* (Mestre do Scrum) – elemento da equipe que se responsabiliza pela gestão do produto e por liderar as práticas executadas na arquitetura. Ele acompanha todos processos no intuito de facilitar a comunicação entre cliente e equipe de desenvolvimento.
- *Scrum Team* (Equipe Scrum) – é responsável por estimar tempos de cada tarefa definida, desenvolver o produto, garantir sua qualidade e apresentá-lo ao cliente a cada fim de sprint.

Tendo conhecimento da importância de cada papel, pode-se fazer uma descrição mais detalhada dos elementos, cerimônia e artefatos, do Scrum Framework.

Artefatos:

- *Product Backlog* – é uma lista de todas as funcionalidades definidas pelo cliente através de User Stories. Comumente conhecido como requisitos.
    - *User Story* – é uma descrição literal mais detalhada de algum item do *Backlog*. Elas auxiliam no entendimento do que está sendo planejado para o produto.
- *Sprint Backlog* – conjunto de tarefas associadas à uma equipe durante um *Sprint*. Essas tarefas passam por um planejamento em que cada integrante da equipe estima o tempo que vai levar para cumprir sua tarefa com base no nível de dificuldade.
- *BurnDown* – ferramenta que representa graficamente o gerenciamento do processo de desenvolvimento de software.

Cerimônias:

- Reunião de Planejamento da Sprint – Esta reunião tem como objetivo definir os itens do *backlog* com maior prioridade e projetar as tarefas que serão executadas durante a sprint. Para isso, ocorrem duas etapas:
    1. O *Product Owner* descreve as funcionalidades de prioridade mais elevadas para a equipe.
    2. A equipe discute a quebra de tarefas sobre as funcionalidades descritas e distribuem entre si.

  No fim da reunião, que tem como duração oito horas (ver tabela 2), as tarefas planejadas dão origem ao *Sprint Backlog.*

Tabela 1 Detalhes sobre a reunião de planejamento da Sprint

Fase	Inicial
Tempo (Máximo)	Oito horas (8 h)
Participantes	Product Owner, Team Scrum e Scrum Master

Fonte: (SCRUM Experience (O tutorial SCRUM), 2009)

- Reunião Diária – Durante cada dia do *Sprint* é realizada uma reunião rápida entre as equipes e o *scrum master, com* o objetivo de identificar problemas enfrentados no processo de desenvolvimento de alguma tarefa. Essa cerimônia, de acordo com a tabela 3, dura, geralmente, 15 minutos.

Tabela 2 Estrutura de uma Reunião diária

Fase	*Sprints*
Tempo	Aproximadamente quinze minutos (15 min)
Participantes	Team Scrum e Scrum Master

Fonte: (SCRUM Experience (O tutorial SCRUM), 2009)

- Revisão da Sprint – Ao final de cada *sprint* é feito uma revisão (*Sprint Review Meeting)* em que o *scrum team* apresenta para os envolvidos (observar tabela 4) o que foi produzido durante o *sprint.* Todo o processo da cerimônia deve durar em média 4 horas.

Tabela 3 Detalhes sobre a revisão da Sprint

Fase	Segunda e Terceira Fase
Tempo	Quatro horas (4 h)
Participantes	Product Owner, Team Scrum e Scrum Master

Fonte: (SCRUM Experience (O tutorial SCRUM), 2009)

- Retrospectiva do *Sprint* – De acordo com a tabela 5, ocorre tanto ao final de um *Sprint* quanto na fase de encerramento. Ela consiste em fazer a identificação do que funcionou bem durante a *sprint,* o que pode ser melhorado e quais ações devem ser tomadas para que sejam melhoradas.

Tabela 4 Estrutura de uma retrospectiva da sprint

Fase	Segunda e Terceira Fase
Tempo	Três horas (3 h)
Participantes	Team Scrum e Scrum Master

Fonte: (SCRUM Experience (O tutorial SCRUM), 2009)

A aplicação de cada elementos das regras desse *framework* em algum projeto é uma tarefa cautelosa, pois acelera o desenvolvimento afetando diretamente os desenvolvedores e seu uso de forma inadequada pode trazer atrasos no processo. Diversas empresas, como: Google, Philips, Intel, entre outras, fazem uso dessa metodologia em seus projetos, porém, uma análise sobre seu uso determina projetos. Nisso, normalmente, é necessário adaptar a estrutura para atender as necessidades ou definir a adesão de outra metodologia.

No processo de desenvolvimento da aplicação deste trabalho, a metodologias desse modelo foram adaptados (ver no capítulo 3) por conta de limitações como a quantidade de equipe e a exigência da demanda. Essa adaptação atribuiu alterações no tempo de cada sprint e na execução de algumas cerimônias. O controle das atividades foi feito através da ferramenta Trello, esclarecido na próxima seção.

2.6. Postman

O Postman é um ambiente que auxilia no desenvolvimento de API dando suporte a todas as etapas do seu ciclo de vida. O aplicativo (app), de início, era disponibilizado apenas como extensão do Google Chrome realizando requisições HTTP através de uma interface executada no próprio browser. Com a demanda de usuários utilizando o ambiente, o app passou a ter um executável no desktop disponível para os Sistemas Operacionais Windows, Linux e Mac abandonando a dependência do navegador.

A ferramenta disponibiliza diferentes recursos para a administração da API. Dentre eles, apenas, as funcionalidades de monitoramento, documentação e coleção foram aplicadas neste projeto.

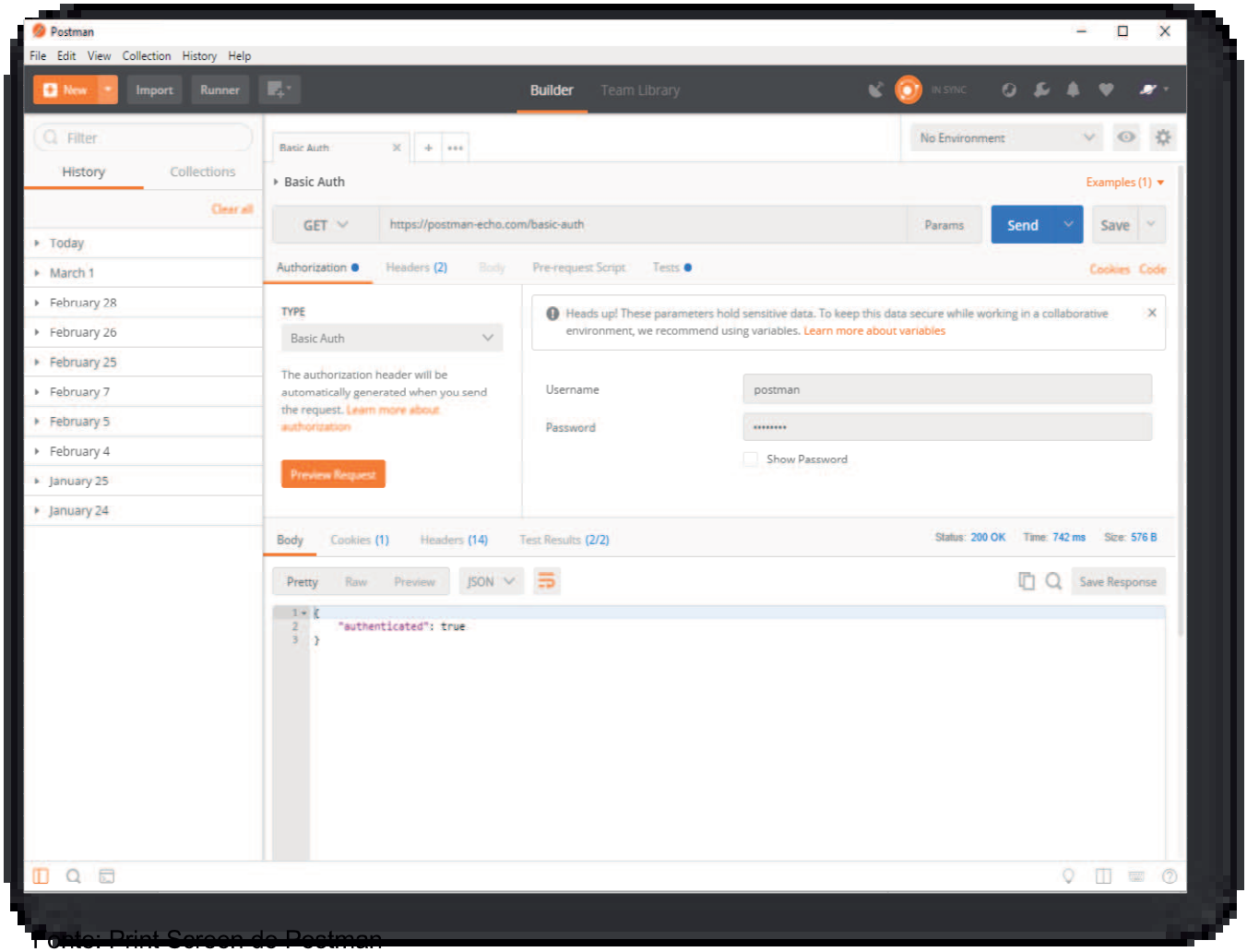
Monitoramento – O desenvolvedor, através de uma interface intuitiva, consegue identificar o tempo de resposta de uma requisição; avaliar seu status; analisar os dados recebidos como respostas e gerenciar os dados que devem ser utilizados durante o teste. Esse monitoramento se encontra, como exibido na figura 7, na parte central da interface.

Documentação – O Postman tem um sistema que gera a documentação geral da API testada, através dos campos de descrições preenchidos na seção de configurações paras os testes de rotas e suas funcionalidades. Ele coleta os resultados de cada teste; concatena com sua descrição, título e dados de entrada, caso tenha; em seguida faz os mesmos procedimentos com os demais na ordem que estão salvos; gerando, assim, toda a documentação. O acesso a esse recurso é feito através da coleção por meio do menu de opções.

Coleção – A coleção é uma forma de organizar os testes em diretórios e uma maneira de deixá-los prontos para uma documentação geral e um autoteste, em que as funcionalidades de uma pasta específica ou do projeto como um todo são testados

de uma só vez, produzindo como resposta gráficos relacionados ao desempenho de cada um. Pode ser encontrado na barra da lateral esquerda da interface e seu processo de registro se assemelha ao sistema de gerenciamento de diretório de um Sistema Operacional.

Figura 6: Tela principal do Postman



Fonte: Print Screen de Postman

3. METODOLOGIA

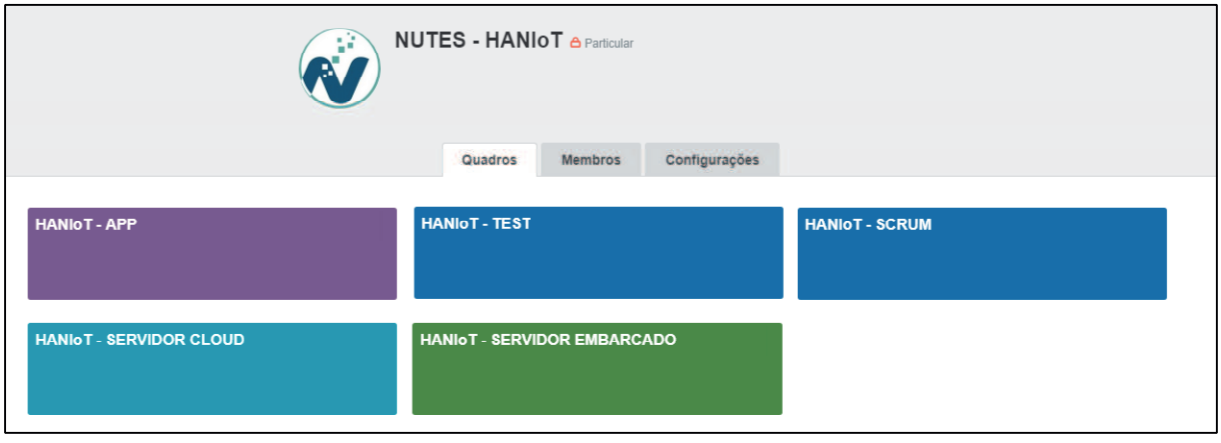
Neste capítulo são apresentadas as etapas metodológicas realizadas durante o desenvolvimento deste trabalho. Os métodos aplicados em cada etapa estão descritos nos tópicos a seguir.

3.1. Administração das Atividades

Com a intenção de administrar e organizar as atividades executadas ao longo deste trabalho, optou-se por utilizar uma plataforma que auxilia na organização e administração da produção de forma flexível, colaborativa e focado na produtividade, chamado Trello[4]. Seus quadros, listas e cartões permitem que o usuário priorize e organize de forma flexível os projetos.

Para aplicar os parâmetros do *Scrum* dentro da plataforma, houve a necessidade de dividir os compromissos entre diferentes quadros do Trello. Um quadro geral ficou encarregado de registrar o *Product Backlog,* com toda a lógica de negócio definida, e supervisionar os demais quadros. Esses outros quadros, , como ilustrado na figura 6, registram atividades mais específicas de uma determinada área do projeto e conta com suas respectivas atividades atreladas ao quadro geral.

Figura 7 Quadros do Trello do projeto HANIoT



Fonte: Print Screen do Trello

[4] https://trello.com/

O quadro “*HANIoT – Scrum*”, como citado anteriormente e apresentado na imagem acima, tem o papel de supervisionar e registrar o desempenho dos quadros: “APP”, “Teste”, “Servidor Cloud e “Servidor Embarcado”. Como a própria nomenclatura já deixa claro, cada quadro, além do Scrum, controla de tarefas especificas sobre componentes do sistema HANIoT. Essas tarefas são descritas em respectivos cartões e inseridos numa lista que representa o estado atual da tarefa que, caso seja alterado, o cartão é transferido para a lista correspondente ao seu novo estado.

No total, uma tarefa pode transitar entre quatro estados:

- Planejamento – O estado em que a tarefa é retirada da *user story* e detalhada para o componente específico.
- Andamento – A tarefa está sendo executada de acordo com o que foi planejado.
- Finalizada – Ao concluir a tarefa, seu cartão é transferido para o quadro geral, onde será analisado e validado.
- Problema de validação – Caso a tarefa que estava finalizada tiver algum problema durante o processo de validação, é transferido para esse estado mais uma cópia para o planejamento.

Seguindo esse fluxo de administração das atividades, o processo de desenvolvimento deste e de outros trabalhos, envolvendo o sistema *HANIoT*, obtiveram retornos positivos como: acompanhamento da solução, relatório de desempenho, registros da evolução de uma atividade, entre outros. Assim, com esses retornos, as reuniões passaram a ter pautas mais objetivas e proveitosas.

3.2. Processo de Desenvolvimento

O desenvolvimento do componente servidor seguiu um processo ágil baseado no modelo Scrum (conceito na seção 2.6). Como o Scrum proporciona flexibilidade em sua metodologia, algumas adaptações foram efetuadas em razão do andamento prévio do desenvolvimento em relação ao gerenciamento. Sendo assim, segue abaixo a descrição de cada fase adaptada do processo:

1. Planejamento – Essa fase consistiu na organização das *User Stories* com seus respectivos critérios de aceitação, dividindo as reponsabilidades entre a aplicação mobile e o componente servidor. Durante a organização, algumas funcionalidades já implementadas foram revisadas e enquadradas tomando como base o *Product Backlog* reformulado nessa fase.
2. Ciclos de *Sprints* – O *sprint backlog* foi sistematizado juntamente com as revisões planejadas na primeira fase. Nisso, cada *sprint* estava relacionada a uma tarefa programada no quadro do Trello, como visto no tópico anterior. Cada *sprint* tinha seu tempo variado de acordo com as dificuldades encontradas na *refactoring* (refatoração) de algumas funcionalidades já implementadas e a sincronização com as novas *user Stories* definidas. As reuniões diárias eram feitas apenas com a equipe de desenvolvimento e ocorriam durante o surgimento de empecilhos no fluxo de codificação, em que era necessário revisar a arquitetura e fazer determinados ajustes. A arquitetura do projeto como um todo foi modelada durante essa fase em paralelo com a codificação.
3. Encerramento do projeto – Não houve a entrega de um produto nessa fase, em vez disso, um *release* foi encerrado e utilizado como protótipo para teste sobre o fluxo geral do sistema, analisando a integração com as outras aplicações e certificando o cumprimento dos critérios de aceitação.

Com a implantação desse processo mais a incorporação do fluxo *Gitflow Workflow* (apêndice A) durante o desenvolvimento, proporcionou uma organização melhor e controle sobre a implementação das *User Stories* mais apropriado. Dessa forma, cada *feature* iniciada no projeto do repositório está relacionada diretamente com um conjunto de tarefas de um *user story,* onde os *commits* detalham sobre codificações efetuadas.

3.3. Recursos Aplicados no Desenvolvimento

O gerenciador de pacotes do node.js[5] disponibiliza, conforme o site (Module Counts, 2018), cerca de 590.736 módulos com diversas funcionalidades pré-programadas, prontas para serem integradas aos projetos e de código aberto, sendo

[5] https://nodejs.org/en/

totalmente acessível à comunidade. Dentre essa vasta quantidade de módulos, foram aplicados neste trabalho um total de 10, sendo quatro deles com maior relevância. São eles:

Express – Framework escrito em JavaScript, disponível no gerenciador de módulos do Node e fornecedor de um conjunto de recursos robustos utilizados tanto em produção quanto no desenvolvimento de aplicações web. Se responsabiliza, inteiramente, por levantar o servidor, disponibilizando uma estrutura de *view* intuitivo (MVC); um potente sistema de roteamento; entre outros recursos que agilizam o processo de desenvolvimento tornando o código mais legível, escalável e de fácil manutenibilidade.

Mongoose – Esse módulo auxilia na integração do Node.js com o MongoDB (mencionado na seção 2.2.2) fornecendo uma solução direta e baseada em esquemas que modelam os dados da aplicação. Ele inclui conversão de tipo incorporada, validação, criação de consultas, ganchos (*hooks*) de lógica de negócios e muito mais.

JWT – Um *JSON Web Token* (JWT) é definido como um intermédio seguro, compacto e autónomo usado na transmissão de informações restritas. Geralmente, é usado para enviar informações por meio de uma assinatura digital (Token). O módulo responsável por fornecer as funções essências do JWT é o *jwt-simple* que, juntamente, com o *passport-jwt* auxiliam no processo de autenticação do projeto.

MomentJS – Módulo desenvolvido para analisar, validar, manipular e exibir datas e horas para ambientes JavaScript. O sistema de histórico de dados do componente servidor, incorporou esse pacote, o que auxiliou na filtragem dos dados.

Com a implantação desses módulos, foi possível não só acelerar a implementação das principais funcionalidades deste trabalho como também ter suporte de terceiros através da interação com a comunidade que faz uso dos pacotes.

3.4. Estrutura da persistência dos dados

Os dados no MongoDB possuem um esquema (*schema*) flexível que define um modelo de dados para uma entidade que se registrará na base de dados. Como visto na seção 2.1, os dados no MongoDB são armazenados como documentos, assim, um agrupamento desses documentos formam uma Coleção, equivalente a uma tabela em banco de dados relacionais. Essas coleções, diferente das tabelas, não são obrigadas a seguir uma definição de schema, ou seja, os documentos dentro de uma coleção podem ter campos diferentes, porém, normalmente, todos os documentos em uma coleção têm um objetivo semelhante ou relacionado.

Tendo ciência disso, a estruturação dos *schemas* de cada entidade do projeto *HANIoT* tomou como base um diagrama entidade relacional para sua modelagem. Isso ocorreu devido ao fato de que o projeto já estava em andamento e a implantação do componente servidor, centralizando o armazenamento desses dados, precisou seguir fielmente a modelagem predefinida.

Então, para fazer o mapeamento da estrutura SQL para o MongoDB, foram necessárias diversas consultas na documentação[6] da ferramenta. Nela há seções que detalham sobre a configuração do ambiente, exemplos de *schemas,* entre outros, em que uma seção especifica apresenta várias instruções SQL sendo correspondidas por instruções do MongoDB, veja a figura 8.

Figura 8 Comparação de uma inserção em SQL e MongoDB

Fonte: Documentação MongoDB

[6] https://docs.mongodb.com/

Dessa forma, cada entidade do sistema *HANIoT* foi mapeado da sua modelagem relacional para uma estrutura não relacional.

3.5. Validação de funcionalidade

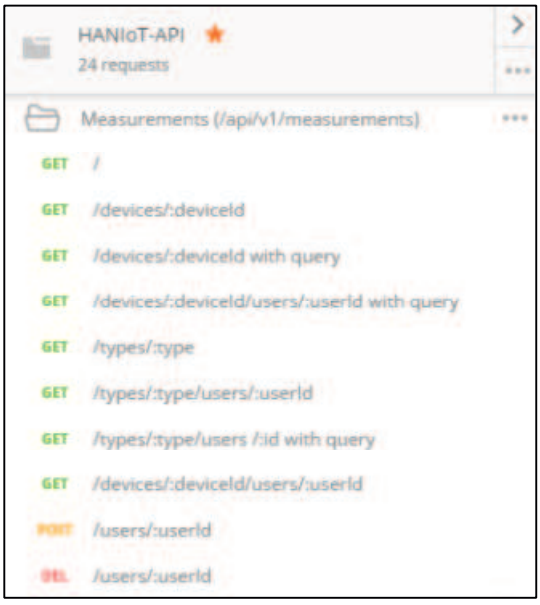
Durante o desenvolvimento de cada funcionalidade, validações utilizando a ferramenta Postman eram efetuadas para verificar a procedência da implementação de acordo com os critérios de aceitação. Como citado na seção 2.6, o Postman permite organizar os testes de uma rota em diretórios específicos. Portanto, cada entidade, como ilustrado na figura a seguir, possui seu respectivo diretório com suas rotas agrupadas nele.

Figura 9 Diretórios da API HANIoT

Fonte: Própria Autoria

As rotas, geralmente, são nomeadas com uma palavra chave que indica seu papel na API, porém, neste projeto as rotas foram nomeadas de acordo com o

Figura 10 Requisições do diretório "Measurements"



Fonte: Projeto do Postman (própria autoria)

complemento da rota raiz da entidade. Analisando a figura 10, pode se observar que o diretório “Measurements (/api/v1/measurements)” foi aberto e que dentro dele encontra-se seus complementos juntamente com a marcação do método de requisição HTTP. Por exemplo, o método *GET* com a nomenclatura “*/devices/:deviceId*” indica que o monitoramento sobre a requisição *GET* será feito na rota “/*api/v1/measurements/devices/:deviceId*”, onde “*deviceId*” representa um parâmetro, no qual deverá ser incorporado no caminho da rota influenciando na resposta.

Além do diretório e nomenclatura dos testes, o Postman permite descrever e monitorar a funcionalidade desejada. A descrição serve tanto para orientações gerais da execução, tipo de retorno esperado, estrutura para requisição quanto para compor a geração da documentação automática. Já o monitoramento, simula a requisição HTTP de acordo com o método especificado. Com isso, a implementação de cada funcionalidade pode ser validade de forma rápida e eficiente.

4. DESENVOLVIMENTO

Neste capítulo apresentamos as definições e materiais sobre a estrutura geral do sistema HANIoT, definimos a arquitetura de projeto do servidor e detalhamos as funcionalidades desde seu desenvolvimento até sua execução. Para isso, foi necessário dividir este capítulo em dois tópicos específicos. Sendo eles: Estruturação geral e Funcionalidades do componente.

4.1 Estruturação Geral

De início, o projeto tinha como objetivo capturar dados de dispositivos médicos pessoais, como: termômetro, balança e aferidor de pressão, utilizando uma conexão Bluetooth como meio de captação dos dados e armazenando-os em um aparelho *SmartPhone*. Um protótipo inicial foi desenvolvido com essas funcionalidades servindo como modelo básico para extração de novos requisitos, aperfeiçoando o sistema. Em paralelo à análise de requisitos, pesquisas sobre produtos no mercado foram realizadas com o intuito de identificar e incorporar finalidades complementares ao

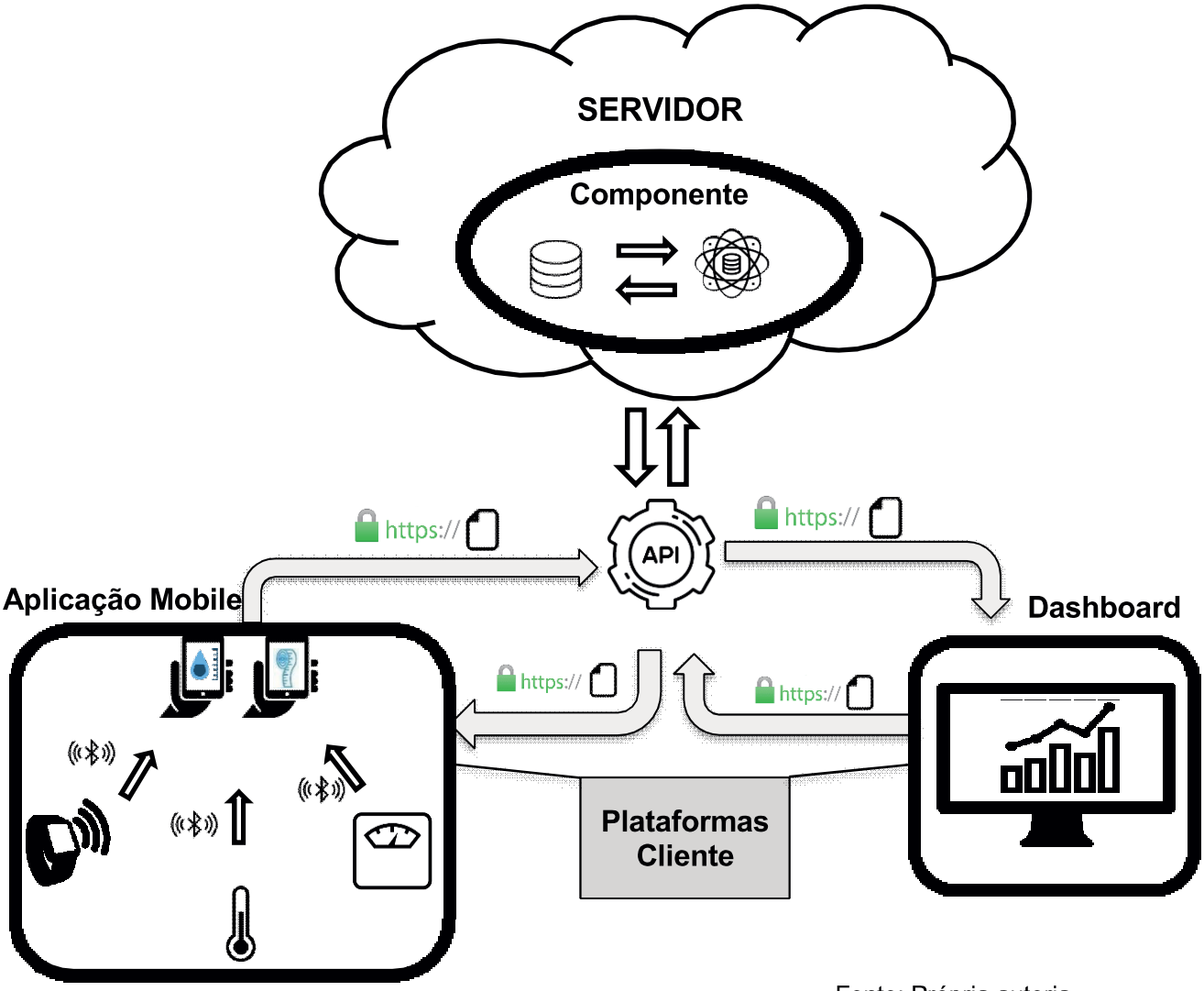
projeto. Nesse processo, observou-se que determinadas empresas, como Philips[7] e a YUNMAI[8], disponibilizam serviços online, junto a suas aplicações, exibindo informações adicionais a respeito das medições do usuário, apenas, acessando seus dados capturados. Com o relatório dessa pesquisa, foram detectados requisitos aplicáveis na composição do projeto, os quais centralizariam os dados capturados em um servidor cloud e fornecendo serviço sobre os dados armazenados.

Um servidor escalável, responsável pelo gerenciamento e manutenibilidade dos dados nele armazenado foi projetado com a finalidade de ser inserido no ambiente IoT, se enquadrando em diversos contexto, através de uma estrutura adaptável. O componente servidor além de fornecer uma API encarregada em distribuir serviços para aplicações clientes, dará suporte aos estudos dos dados (Data Science), a partir dos dados coletados. Esses estudos seriam disponibilizados para profissionais de saúde com informações mais estruturadas e com detalhamento científico. Para este trabalho de conclusão de curso, serão apresentados, apenas, a arquitetura e o desenvolvimento das funcionalidades disponibilizadas pelo componente desenvolvido.

[7] https://www.philips.com.br/
[8] https://yunmaibrasil.com.br/

4.1.1 Visão geral do Sistema

Figura 11 Fluxo geral do sistema HANIoT



Fonte: Própria autoria

O sistema HANIoT é constituído por três extremos interdependentes, conforme ilustrado na figura 11, em que cada extremo tem sua importância específica dentro do fluxo principal seguindo o escopo da arquitetura IoT (seção 2.3). A aplicação mobile se responsabiliza em fornecer os dados capturados dos dispositivos médicos pessoais, enquanto o dashboard consome informações contidas na base de dados do servidor, por meio de serviços fornecidos pelas APIs. Para isso, o servidor dispõe de

um componente responsável por gerenciar essas APIs e executar os serviços requisitados pelas plataformas clientes.

A estrutura de cada extremo está dentro do escopo da arquitetura Iot. A plataforma mobile está presente nas três camadas dessa arquitetura, pois além de enviar os dados captados, exibe os mesmos através de uma interface própria. O dashboard, apenas na camada superior, pois sua interação é direta com o usuário. Inserido em duas camadas, temos o servidor que se encarrega de transitar as informações para as plataformas clientes, mas também age como receptor de dados e requisições.

Todos os elementos desse sistema, embora sejam de projetos diferentes, compartilham das mesmas regras de negócio definidas no Product Backlog através dos critérios de aceitação contidos nas User Stories (US). Para esse fim, foi efetuado uma análise das US e extraído as entidades envolvidas e todo relacionamento delas por meio de diagramação.

4.1.2 Modelagem do Servidor

Na diagramação do projeto foi estabelecido padrões para o desenvolvimento de cada elementos do sistema garantindo sua integridade. Antes de definir qualquer diagrama, as entidades constituintes do sistema foram determinadas durante a análise dos requisitos.

A definição dessas entidades partiu das descrições de cada User Story modelada durante o processo de pesquisa e investigação de requisitos que podem ser vistas na tabela a seguir.

Tabela 5 Produto Backlog com todas as User Stories

User Story	Título	Descrição
US01	Cadastro de usuário	O usuário deve ser capaz de realizar o cadastro em qualquer plataforma do sistema usando o aplicativo móvel **Critérios de Aceitação:** • Segurança no armazenamento; • Validação no cadastro de acordo com a estrutura do Usuário. • Usuário deve ter E-mail único
US02	Autenticação de Usuário	O acesso a qualquer plataforma do sistema deve exigir a autenticação de usuários previamente cadastrados. **Critérios de Aceitação:** • Nível de acesso; • Autenticação deve limitar acesso a diferentes níveis: Super. e comum; • Gerador de Token.
US03	Cadastro de dispositivos pessoais de saúde (PHD)	O usuário cadastrado no sistema deve ser capaz de associar novos Personal Health Devices a sua conta. **Critérios de Aceitação:** • Apenas o usuário comum efetua esse cadastro; • Validação para estrutura de dispositivo definida; • Dispositivo pode ser associado a um ou mais usuários; • Remoção geral de um dispositivo da base de dados se tiver, apenas, um usuário associado.
US04	Aquisição de dados dos PHDs	Os usuários autenticados no aplicativo móvel devem ser capazes de conectar dispositivos cadastrados na sua conta e receber medições realizadas por estes. **Critérios de Aceitação:** • O Usuário conectado no aplicativo móvel deve ser capaz de se conectar a dispositivos cadastrados na sua conta; • O usuário conectado no aplicativo móvel deve ser capaz de receber medições de dispositivos por meio de um protocolo específico.

US05	Armazenamento de dados de saúde	Os dados de saúde dos usuários devem ser armazenados temporariamente pelo aplicativo móvel, e permanentemente pelo servidor WEB **Critérios de Aceitação:** • O servidor deve efetuar o armazenamento permanente dos dados do usuário em um BD não relacional. • O modelo de dados do componente servidor e do aplicativo móvel deve ser validado de acordo com as consultas e operações previstas
US06	Sincronização de dados de saúde do usuário	O aplicativo móvel deve sincronizar com o servidor os dados salvos localmente na conta do usuário conectado no aplicativo móvel. A sincronização deve acontecer de forma automática e/ou sob demanda do usuário. **Critérios de Aceitação:** • Sincronização das medições capturadas pelo aplicativo móvel deve ser disponibilizadas automaticamente e manualmente. • Caso não tenha conexão com servidor, os dados devem ser armazenados temporariamente no aplicativo móvel e lançados de uma vez ao se conectar com o servidor. • A sincronização de medições com servidor deve ser feita por meio de conexão criptografada.
US07	Visualização de histórico de dados de saúde do usuário	O aplicativo móvel deve permitir a visualização do histórico dos dados de saúde associados ao usuário. A visualização inclui dados salvos localmente na conta do usuário conectado no aplicativo móvel, na ausência de acesso à internet, e dados no servidor WEB. **Critérios de Aceitação:** • O usuário conectado ao aplicativo móvel deve ser capaz de visualizar os valores das últimas medições realizadas por um dispositivo cadastrado. • A consulta ao histórico de medições salvas no servidor deve ser feita por meio de conexão criptografada entre aplicativo móvel e servidor.

Fonte: Equipe HANIoT

Definição das entidades:

- **User (usuário) –** Abstraindo os stakeholders que interagem com o sistema, user é um esquema de algumas informações pessoais úteis mais o id de uma entidade para acesso. Em resumo, o objeto usuário tem como atributos: nome, data de nascimento, sexo, altura e o id da entidade Login.

- **Login –** Responsável por gerenciar as permissões de acesso dos usuários aos serviços do servidor. Nisso, o login é composto pelos seguintes atributos: e-mail, senha e grupo, no qual é divido em dois:
    - Grupo de super usuários: Têm acesso a todas as informações, contidas na base de dados do servidor, sobre as outras entidades e os futuros serviços para tratamento desses dados. Entretanto, não pode fazer nenhuma alteração na base de dados.
    - Grupo de usuários comum: Acessam suas informações e podem efetuar cadastros de medições associadas ao seu perfil.

- **Device (dispositivo) –** É o registro dos dispositivos responsáveis por realizar as medições de cada usuário. Um user associa seu dispositivo (device) de medição, cadastrando-o na base de dados do servidor e fornecendo seu id como atributo desta entidade. Seus atributos são: seu endereço mac, nome, fabricante, número do modelo, um id do usuário e o id do tipo que define qual sua função.

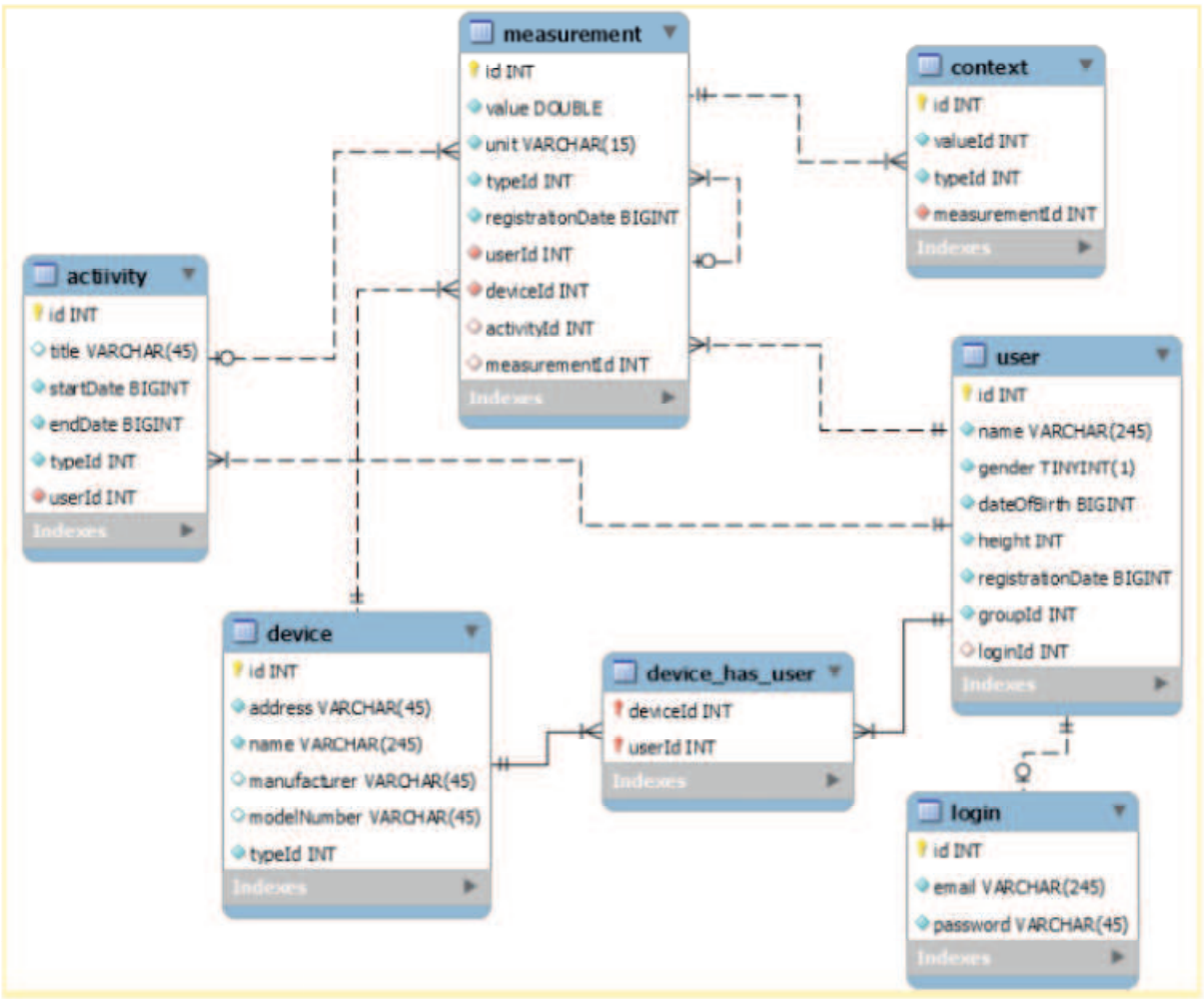
- **Context (contexto) –** Está relacionado com as circunstâncias de estado durante as medições, exemplo: “o momento em que a medição foi efetuada o usuário correu?”; “Já havia almoçado?”; “Acabou de acordar?”. Cada resposta a essas perguntas corresponde a um estado ou contexto da medição. Esses estados são ponderados por seus respectivos valueId e typeId
- **Measurement (medição) –** Principal entidade, corresponde a junção dos ids das demais entidades mais seus valores capturados pelo device. O fluxo da medição começa do dispositivo e termina com sua estrutura de atributos após percorrer pelos procedimentos internos do servidor. Seus atributos armazenados na base de dados são: valor, unidade, data de registro, tipo de

medição, id do usuário, id do dispositivo e dois vetores de ids um para objetos Context e outros para medições.

- **Activity (atividade) –** Entidade inserida em recentes Users Stories e que corresponde a prática de alguma atividade feita pelo usuário utilizando algum dispositivo responsável por controlar determinadas medições de acordo com os exercícios praticados em um intervalo de tempo específico. Na primeira versão essa entidade não foi incorporada por falta de demanda, entretanto, foi definida para futuras versões.

Relacionamento das entidades

Figura 12 Diagrama entidade relacional



Fonte: Equipe HANIoT

No diagrama entidade relacional acima (figura 12) pode-se notar a centralização da entidade *measurement* (medição) onde as demais entidades se relacionam com ela diretamente ou indiretamente. Isso ocorre por causa do fluxo que o dado capturado passa até ser armazenado na base de dados. Desde sua captação pelo dispositivo (*device*) até ser associado a algum usuário, a medição incorpora novas informações através dos atributos aplicados durante o armazenamento. Dessa forma, pode se garantir a transparência de cada medições, facilitando futuros tratamentos.

4.1.3 Arquitetura do Projeto

Para ser inserido no ambiente, esboçado anteriormente, foi necessário estabelecer uma arquitetura que atendesse à expansão de requisitos mantendo ainda sim a integridade do servidor. Aderiu-se, então, o módulo Express com a finalidade de fornecer uma base forte dentro do servidor. Com ele o sistema de roteamento da API pode ser desenvolvido e os tratamentos das requisições via HTTP simplificados.

O padrão de arquitetura de software empregado pelo Express é o MVC (*model-view-controller*) no qual separa a representação da informação em camadas de responsabilidades. Entretanto, neste projeto, a parte da *view* não é aplicada, em vez disso a interação com usuário ficou encarregada pelas aplicações cliente (*cliente-side*). Em APIs, o *controller* recebe a requisição da rota, faz a chamada para o model realizar a lógica de negócio e retorna a resposta para o cliente que, no caso desse projeto, são outras aplicações.

Essa separação de responsabilidades feita pelo *MVC* facilita a modularização das funcionalidades da aplicação e possibilita:

- Manutenibilidade: Qualquer alteração que precisar ser feita terá mais facilidade de ser implementada e para localizar onde ocorrerá impactos.
- Desacoplar *models e views* facilitando os testes em ambos isoladamente.
- Reutilização de código.

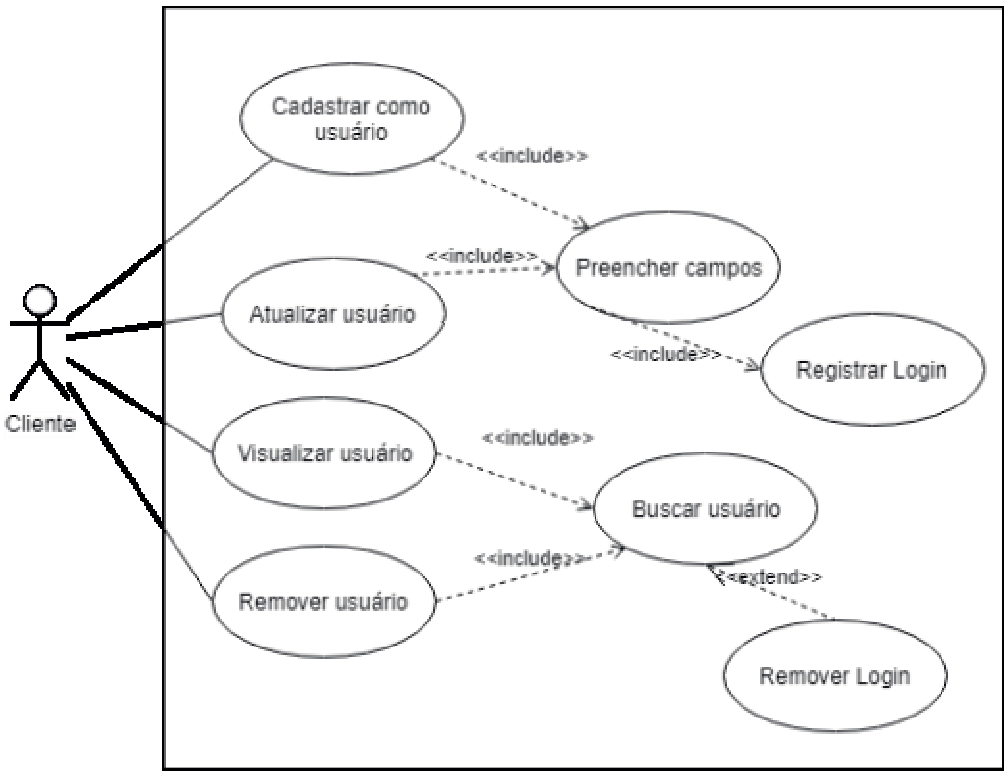
4.2. Funcionalidades do Componente

Neste tópico será apontado, detalhadamente, as sete user stories presentes no Produto Backlog e descrito como foram implementadas dentro do projeto. Cada user story tem um conjunto de atividades específicas para um elemento integrante do sistema. No caso deste trabalho, as tarefas esclarecidas serão os referentes ao componente servidor. Assim, a user story sobre aquisição de dados não será abordada.

4.2.1 Cadastro de Usuário

Este requisito engloba duas entidades: User e Login. Como já descritas anteriormente (tópico 4.1.2) essas entidades compõem o stakeholder que tem acesso a diversas funcionalidades fornecidas pelo sistema. Uma forma de referenciar ambas é associando o identificador (id) de uma entidade na outra, sendo assim, usuário armazena em seu documento uma propriedade *loginId* do tipo *Object* Id. Desse modo, quando o user for solicitado, um método do Mongoose popula os dados existentes no documento de Login do id especificado.

Figura 13 Diagrama de Caso de uso da *User Story 01*



Fonte: Própria autoria

O cadastro de usuário só é efetuado após a validação dos campos e o armazenamento de login na base de dados, visto que seu id é incorporado pelo usuário. Durante esse processo de armazenamento do login ocorre a criptografia da senha. Neste momento, um método “pre-save” do login é executado. Nesse método, o módulo *bcrypt* exerce sua função de criptografar a senha, vinda do corpo da

Figura 14 Documento do objeto User e Login

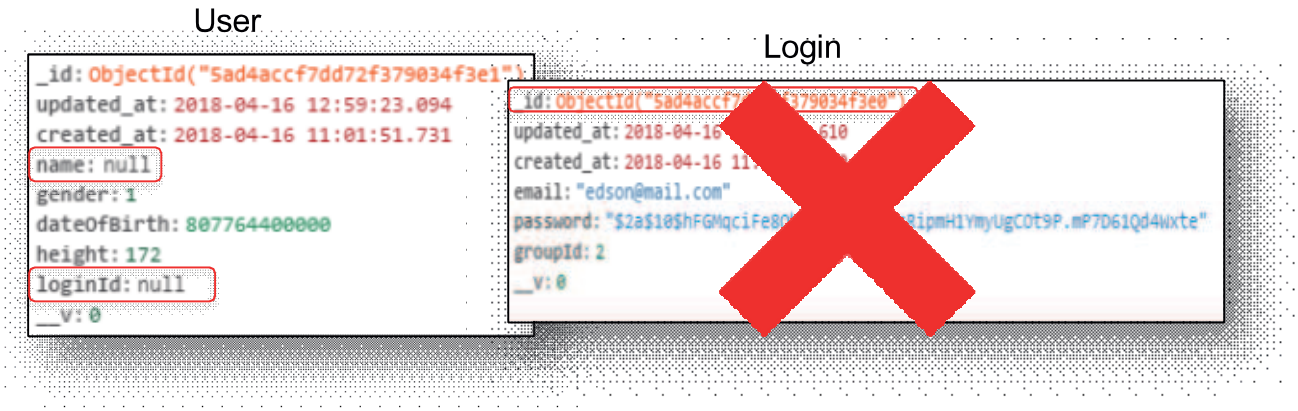
Fonte: Print screen do MongoDB

requisição, gerando um *hash* de no máximo 255 caracteres. Com o resultado da criptografia obtido, o valor anterior da senha é sobrescrito dando continuidade no processo de criação do login. Ao concluir com sucesso o cadastro do login, a função responsável por esse procedimento retorna o id da entidade para que o cadastro do usuário possa ser finalizado.

No procedimento de remoção, descrita na US01, as propriedades altura, data de aniversário e gênero de usuário são mantidos, enquanto o nome e o loginId são atualizados para *null.* Feito isso, a entidade Login que dá acesso ao usuário é deletado de sua coleção.

Em relação à atualização dos dados já armazenados, existem dois métodos distintos: um incumbido de atualizar, apenas, os dados pessoais do usuário (US01-T08) e outro para atualizar a senha de acesso (US01-T09). Ambos necessitam de uma autenticação prévia para que a alteração seja efetuada com sucesso, validando a permissões referentes a essas ações.

Figura 15 Exclusão do documento Login e consequencia no documento User



Fonte: Próprio autor

De acordo com a arquitetura aboradada no desenvolvimento, o módulo realiza a lógica de negócio e retorna a resposta a cada requisição vinda de qualquer aplicação cliente com permissões de consumo adequados.

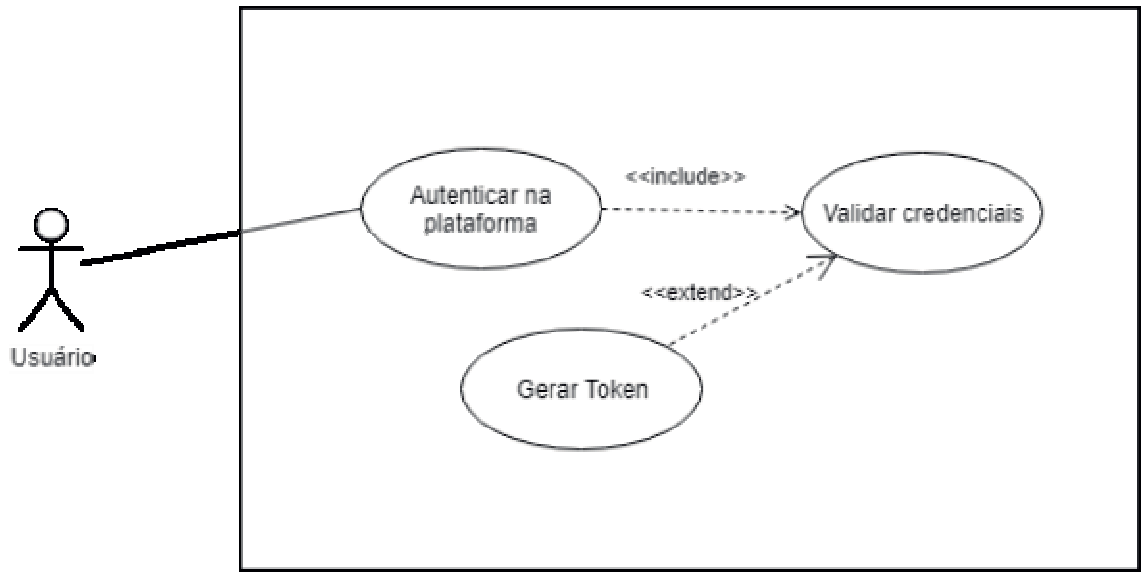
As rotas para consumir este serviço são as seguintes:

- [endereço de hostedagem]/api/v1/users/signup (POST) – Efetua o cadastros e as devidas validações
- [endereço de hostedagem]/api/v1/users/:userId (PUT) – Atualiza os dados pessoais do usuario com o id passado pela url
- [endereço de hostedagem]/api/v1/users/:userId/password (PUT) – Atualiza a senha do usuário com o id contido no parâmetro da url
- [endereço de hostedagem]/api/v1/users/:userId (DELETE) – Realiza o processo de remoção sobre o identificador do usuário contido na url.

4.2.2 Autenticação de Usuário

A autenticação do usuário funciona como um módulo de acesso e segurança para transferência de dados, encarregado de administrar as permissões de consumo das funcionalidades fornecidas pela API. Uma forma simples de implementar essas exigências dessa *User Story*, foi a adesão do JSON Web Token (JWT).

Figura 16 Diagrama de Caso de Uso para *User Story 2*



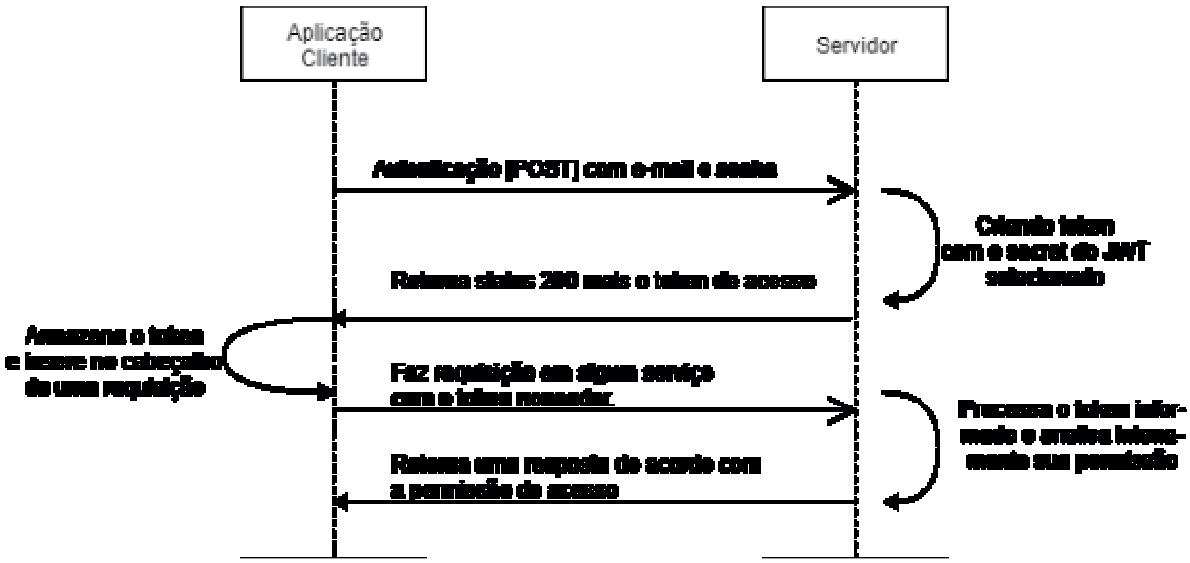
Fonte: Própria autoria

O JWT, citado em 3.1.1, é um sistema de transferência de dados que pode ser enviado tanto via *POST* quanto por cabeçalho HTTP (*header*), de maneira segura. Nela a informação é assinada digitalmente por algum algoritmo de *Hash*. Sua estrutura formada por 3 partes: *Header*, *Payload* e Assinatura; determinam como a informação será repassada de forma segura.

O header é representado por duas partes: o tipo, que neste caso é JWT e o algoritmo de *Hash* HS256 de JWT *Secret* igual *'K3W|y"}%KGpwhX!'.* Payload é um objeto JSON que representa quais dados são enviados pela transferência JWT, nesta situação passamos o usuário com o identificador do login como evidência. Por fim, a assinatura é o produto da codificação do cabeçalho com o *payload* definindo, assim, um *token*. Essa assinatura será usada para verificar se o emissor da JWT é realmente "quem" se diz ser e certificar se não ocorreu alguma modificação no caminho.

Tendo ciência disso, todos as rotas que necessitam de permissão para serem consumidas chamam o método *authenticate* para descompactar o usuário que efetuou uma autenticação e saber se ele tem autorização em acessar aquele serviço. Para que isso ocorra adequadamente, as aplicações clientes precisam armazenar o Token retornado pela requisição de autenticação e associá-lo ao cabeçalho das rotas que deseja consumir.

Figura 17 Diagrama de sequência do JWT



Fonte: Própria Autoria

4.2.3 Cadastro de Dispositivos Pessoais de Saúde

Cada dispositivo está encarregado de captar um ou mais tipos de medições para o servidor. Dependendo do contexto o mesmo dispositivo pode ser utilizado por mais de um usuário do sistema. Então, para que cada usuário tenha seu registro de dispositivo, um cadastro deve ser efetuado associando o id do usuário com os dados

cadastrais do dispositivo. Consequentemente, uma medição só será realizada se o dispositivo estiver cadastrado.

Tabela 6 Tarefas relacionadas à US3

US03	
US03 – T01	Criar módulo de cadastro de dispositivos
US03 – T02	Recuperar os dispositivos associados ao usuário comum
US03 – T03	Recuperar os dispositivos cadastrados através dos usuários comuns a partir do super usuário
US03 – T04	Recuperar um dispositivo específico a partir do super usuário
US03 – T05	Recuperar um dispositivo específico de um usuário
US03 – T06	Remover um dispositivo da base de dados se estiver associado com apenas um usuário, caso contrário remover apenas a associação.

Fonte: Sprint Backlog do projeto

Esse módulo de cadastro dos dispositivos pessoais de saúde valida a existência ou não do próprio, evitando que mais de um dispositivo seja cadastrado no mesmo usuário. Essa validação toma como base o endereço *mac* do dispositivo e o *userId*, ou seja, caso ambos se repitam uma mensagem de erro será lançada.

Com essa associação o super usuário pode separar medições feitas por diferentes dispositivos e ter uma filtragem maior de cada usuário comum que esteja supervisionando. Algumas rotas estão cedidas apenas para o super. Sendo elas:

- [endereço de hostedagem]/api/v1/devices/ (*GET*) – Retorna todos os devices já cadastrados no sistema
- [endereço de hostedagem]/api/v1/devices/users/:userId (*GET*) – Entrega todos os devices cadastrados pelo usuário cujo id foi especificado no parâmetro.
- [endereço de hostedagem]/api/v1/devices/:deviceId (*GET*) – devolve o objeto do device cujo id foi posto na url.
- [endereço de hostedagem]/api/v1/devices/:deviceId/users/:userId (*GET*) – retorna um dispositivo específico com um determinado usuário a partir dos seus respectivos identificadores descritos nos parâmetros.

4.2.4 Armazenamento de dados de saúde

O armazenamento ocorre posterior ao processo de aquisição de dados executada pela aplicação mobile. Armazenar os dados é um critério relatado na US05 no qual o servidor teria um modelo de banco de dados específico e uma estrutura própria de persistência dos dados.

A *User Story* exige um modelo não-relacional para a base de dados, onde a seleção e implantação do tipo de banco *NoSQL* ficaria por conta do desenvolvimento. Então, a adesão de um banco de dados não-relacional orientado a documento, para a aplicação, foi implantada usando o *MongoDB*, como fornecedor, integrado através do módulo *Mongoose*.

Tabela 7 Tarefas relacionadas à US5

US05	
US05 – T01	Selecionar e implantar BD não-relacional

US05 – T02	Modelagem dos dados para armazenamento no BD não relacional.
US05 – T03	Implementar módulo de armazenamento no BD não relacional
US05 – T04	Validação do BD não relacional de acordo com consultas e operações previstas.

Fonte: Sprint Backlog do projeto

A aplicação só é inicializada quando se estabelece uma conexão com o banco de dados, evitando problemas futuros. Isso acontece quando o arquivo *app.js* executa as configurações do módulo *config*.

Quando *app.js* é executado, todas as estruturas de *schemas,* projetados com o *Mongoose*, são registradas e a aplicação inicializa as coleções de cada entidade, já estão configuradas, para receberem os documentos com as informações dos objetos cadastrados. Nesses *schemas* são especificadas as regras de negócios; validações básicas como: campos obrigatórios e tipo de cada campo; relacionamento entre as entidades; e seus métodos.

Figura 18: Model do Device

device.js

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const deviceSchema = new Schema({
    address: { type: String, required: [true, 'Dado necessário'] },
    name: { type: String, default: null },
    manufacturer: { type: String, default: null },
    modelNumber: { type: String, default: null },
    userId: { type: Schema.Types.ObjectId, ref: 'User' },
    typeId: { type: Number, required: [true, "Tipo requerido"] }
}, { timestamps: { createdAt: 'created_at', updatedAt: 'updated_at' } });

deviceSchema.index({ address: 1, userId: 1 }, { unique: true });

module.exports = mongoose.model('Device', deviceSchema);
```

Fonte: Trecho da model de device (Própria Autoria)

4.2.5 Sincronização de dados de saúde do usuário

Neste contexto não houve necessidade de criar uma rota específica de sincronização, pois a própria rota de cadastrar medições é aproveitada nessa situação. Todas as medições armazenadas localmente pelo aplicativo móvel são enviadas em um *array.* O módulo de armazenamento efetua, previamente, um processo de validação sobre cada elemento do *array* para em seguida cadastrá-los ao mesmo tempo.

Tabela 8 Tarefas relacionadas à US6

US06	
US06 – T01	Criar APIs para sincronização de medições com o aplicativo móvel
US06 – T02	Criar módulo para sincronização de medições
US06 – T03	Validar medições recebidas do aplicativo móvel

US06 – T04	Salvar medições recebidas do aplicativo móvel
US06 – T05	Criar conexão criptografada com o app para sincronização de medições

Como pode ser observado na figura 20, há uma *promise* encarregada de preparar o *array* de medições provenientes da requisição. Após obter sucesso nas validações em que o valor de *result* for diferente de *null* todas as medições validadas são armazenadas na base de dados pelo método *create* do *Mongoose* e enviado como resposta o status 201 mais um *json* vazio, visto que o aplicativo móvel não precise de

Fonte: Sprint Backlog do projeto

um json de confirmação, o status é o suficiente.

Figura 19 Promise responsável pelo armazenamento de medição

```
let save = (req, res) => {
    if (req.user.loginId.isSuper() !== false || req.params.userId != req.user._id) {
        return res.status(401).send({ "message": "User unauthorized" });
    }
    prepareMeasurements(req.body.measurements, req.params.userId).then(result => {
        create(result).then(measurements => {
            if (!measurements) return res.status(500).send({ message: "No measurements were saved" });
            return res.status(201).send({});
        }).catch(err => {
            res.status(500).send({ message: err.message });
        });
    }).catch(err => {
        res.status(500).send({ message: err.message })
    });

};
```

Fonte: Trecho do código de Measurements (própria autoria)

4.2.6 Visualização de histórico de dados de saúde do usuário

Há um registro em cada medição armazenado na base de dados através da propriedade *registrationDate* presente no *schema* da entidade *measurement*. Essa propriedade, indica o momento em que a medição foi captada pelo aplicativo móvel, o qual insere a informação nesse atributo no *json* da medição. Dessa forma, os dados podem ser monitorados tanto pelo super usuário quanto pelo usuário comum.

Para fazer isso, um módulo do npm é utilizado no tratamento das informações específicas, solicitadas por meio das *queries strings* da rota. O moment.js faz todo o processo de conversão de datas e calcula as diferenças com métodos simplificados.

Tabela 9 Tarefas relacionadas à US7

US07	
US07 – T01	Fornece APIs para consulta de histórico de medições pelo aplicativo móvel
US07 – T02	Criar módulo para consulta de medições armazenadas no servidor
US07 – T03	Validar usuários para os diferentes tipos de consultas
US07 – T04	Criar conexão criptografada com o aplicativo móvel para consulta de medições.

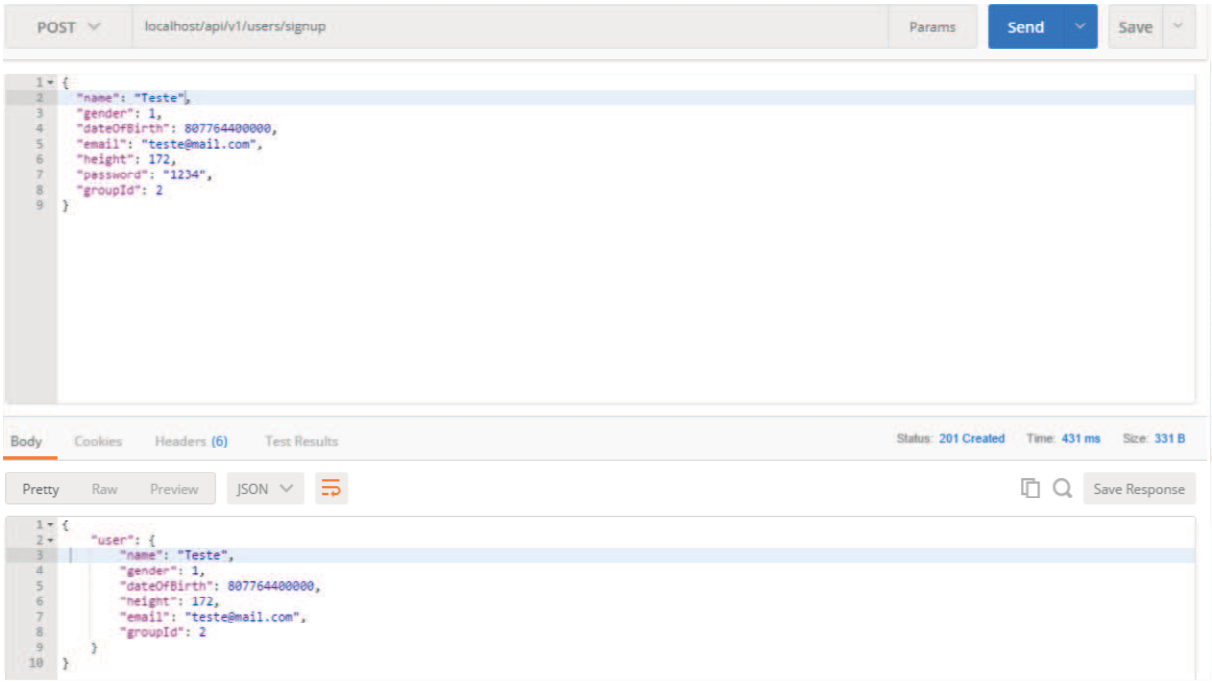
Fonte: Sprint Backlog do projeto

5. VALIDAÇÃO

As validações por serem efetuadas durante a implementação consistiam em efetuar uma requisição na rota do serviço implementado e verificar o retorno da requisição, comparando com o esperado. Assim, esse capítulo apresentará, apenas as principais requisições e seus respectivos resultados de acordo com os critérios de aceitação.

No cadastro de usuários foi necessário validar a omissão da senha e registrar a conta com apenas um único e-mail. Quando a estrutura de usuário é enviada no

Figura 20 Monitoramento do Cadastro

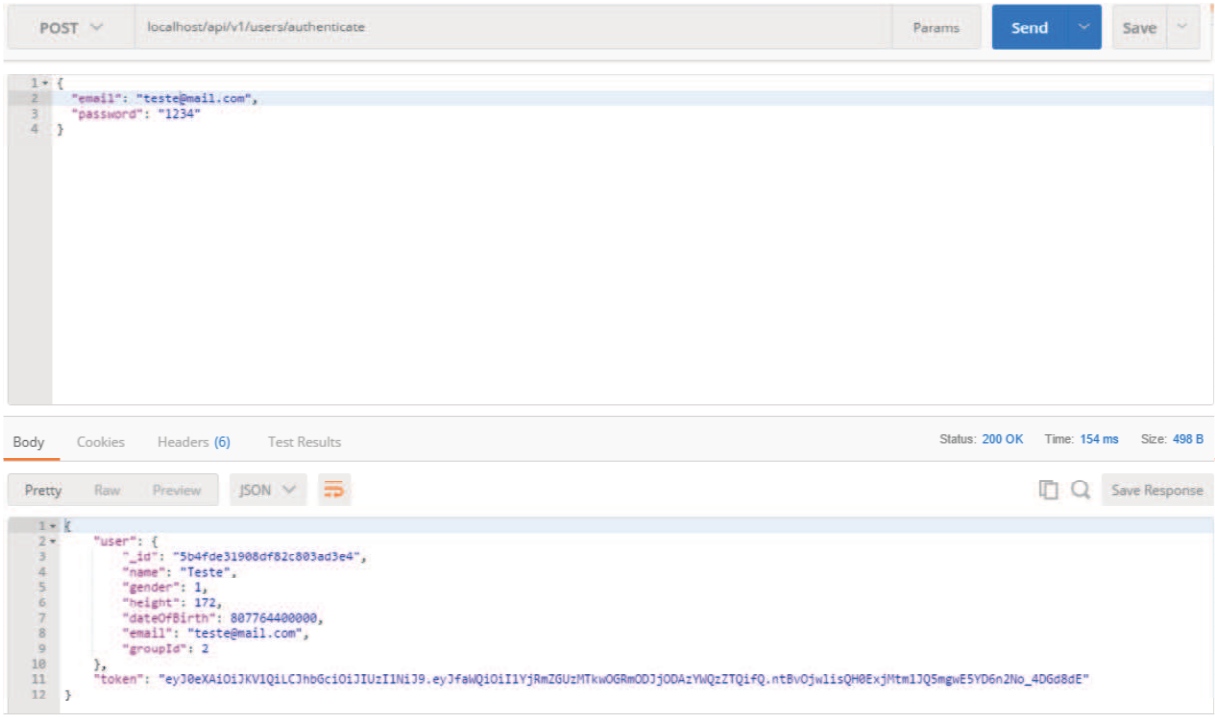


Fonte: Postman *HANIoT*

corpo da requisição, como mostra na figura 20, o status do retorno tem como valor 201 e um *json* resposta com os dados registrados. Ao tentar cadastrar um outro usuário com o mesmo e-mail obtemos uma mensagem de retorno alertando sobre o erro e um status de valor 400.

Na autenticação a *user story* solicita o retorno de um *token*, o qual será utilizado para acessar os demais serviços da API. Na figura 21, pode-se observar o retorno do token mais os dados essenciais da autenticação com o status da requisição igual à 200, significando sucesso.

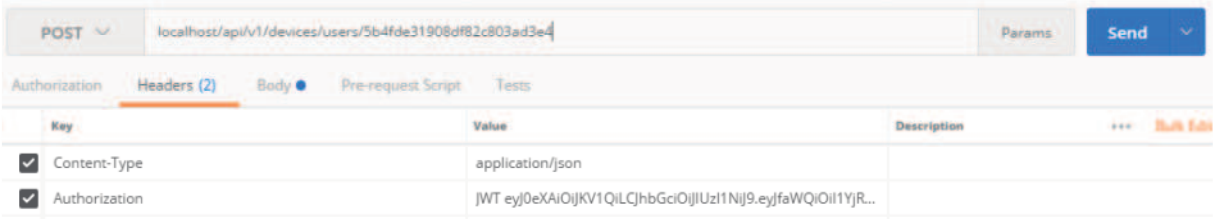
Figura 22 Requisição de autenticação monitorada



Fonte: Postman *HANIoT*

O uso do *token* é feito no cabeçalho da requisição, na qual será avaliada antes de atender o que está sendo solicitado pelo método especificado (POST, GET, PUT...) e o possível corpo no caso de métodos como POST e PUT. A figura 22 exibe como é feito a associação do *token* no header pelo Postman.

Figura 21 Header de uma requisição Post



Fonte: Postman *HANIoT*

A detecção de falhas ou erros são registrados em cartões no quadro do Trello para serem solucionados e analisados com mais detalhes. A cada conjunto de erros ou falhas, relacionadas a uma funcionalidade, relatado, gera-se uma *branch hotfix*, para aplicar as correções sem afetar a ramificação principal do projeto.

6. CONCLUSÃO

Com este trabalho foi possível tornar o sistema *HANIoT* desacoplado, adequado para cenários variados na área da saúde e atendendo aos diversos requisitos necessários de um sistema *Healthcare*. Sua contribuição, como servidor juntamente com o conjunto de *APIs*, possibilitou a estabilidade de uma base sólida, a qual se expande de acordo com novas exigências.

Desenvolvido como elemento integrante de um sistema com arquitetura *IoT*, este trabalho traz características essenciais para a evolução de projetos complementares inseridos no mesmo contexto. Sendo eles: a solução mobile de aquisição de dados e a interface dashboard para usuários específicos. Ambos são trabalhos que estão em desenvolvimento e exigem das funcionalidades desenvolvidas neste projeto para sua evolução e conclusão. Para isso, o componente servidor, por meio de *API*, disponibiliza serviços que se responsabilizam pelo armazenamento, gerenciamento, segurança e tratamento dos dados utilizados pelos trabalhos citados.

Mesmo com um módulo de autenticação funcional, observou-se, através dos testes, que a segurança em determinados cenários poderia vir a falhar. Diante disso, conclui-se que o projeto, nas suas próximas versões, deverá isolar o serviço de autenticação da lógica de negócio, tornando-o um módulo mais autêntico e reaproveitável para os demais extremos do sistema *HANIoT*. Isso aumentará o controle de acesso dos usuários do sistema e a manutenibilidade do módulo, aumentando, proporcionalmente, sua segurança e qualidade.

Por conta da implantação de arquiteturas de microserviços em sistemas de grande porte e o aspecto expansivo do sistema *HANIoT*, faz com que as próximas releases do componente servidor sofram com possíveis fragmentações em seus módulos internos cuja funcionalidades passem a exigir isolamento de execução dentro do sistema.

7. REFERÊNCIAS

Bitbucket. (n.d.). *Tutorials - Gitflow Workflow.* Retrieved from Atlassian: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Chacon, S. (n.d.). *Primeiros passos - Sobre Controle de versão*. Retrieved from Git: https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o

DeBill, E. (2018, Março 04). *Module Counts*. Retrieved from Module Counts: http://www.modulecounts.com

Departamento de Ciência da Computação da universidade Federal de Minas Gerais - UFMG. (2016, Maio 20). Internet das Coisas: da Teoria à Prática. *Minicurso - Simpósio Brasileiro de Redes de Computadores*, p. 50.

Evans, D. (2011, Abril). A Internet das Coisas: Como a próxima evolução da Internet está mudando tudo. *CISCO - White Paper*, p. 13.

Malavasi, A. (2017, 11 15). *Afinal, Javascript e ECMAScript são a mesma coisa?* Retrieved from Medium: https://medium.com/trainingcenter/afinal-javascript-e-ecmascript-s%C3%A3o-a-mesma-coisa-498374abbc47

Mustafa, E. (2016, Janeiro 11). *JavaScript - 20 anos de história e construção da web*. Retrieved from iMasters: https://imasters.com.br/front-end/javascript/javascript-20-anos-de-historia-e-construcao-da-web/?trace=1519021197

Oliveira, L. B. (2017). DESENVOLVIMENTO DE UM AGREGADOR DE DADOS PARA DISPOSITIVOS MÉDICOS PESSOAIS. p. 51.

Portal Saúde Business. (2017, julho 26). *Impacto da tecnologia em saúde.* Retrieved from Saúde Business: http://saudebusiness.com/noticias/o-impacto-da-tecnologia-em-saude-infografico/

Pressman, R. S. (2011). Software Engineering: a Practitioner's Approach. In R. S. Pressman, *Software Engineering: a Practitioner's Approach* (p. 779). New York: The McGraw-Hill Companies, Inc.

RNP - Rede Nacional de ensino e pesquisa. (n.d.). *Tecnologia baseada em internet da Coisas promete redução da obsesidade infanfil.* Retrieved from RNP - Rede

Nacional de ensino e pesquisa: https://www.rnp.br/destaques/tecnologia-baseada-internet-coisas-promete-reducao-obesidade-infantil

Santos, R. F. (2009, junho 20). *SCRUM Experience (O tutorial SCRUM).* Retrieved from Inovação, Sustentabilidade e Tecnologia : http://www.rildosan.com/2009/06/scrum-experience-o-tutorial-scrum.html

Singer, T. (2012, Outubro 10). Tudo conectado: conceitos e representações da internet das coisas. *Simpósio em tecnologias digitais e sociabilidade*, p. 15.

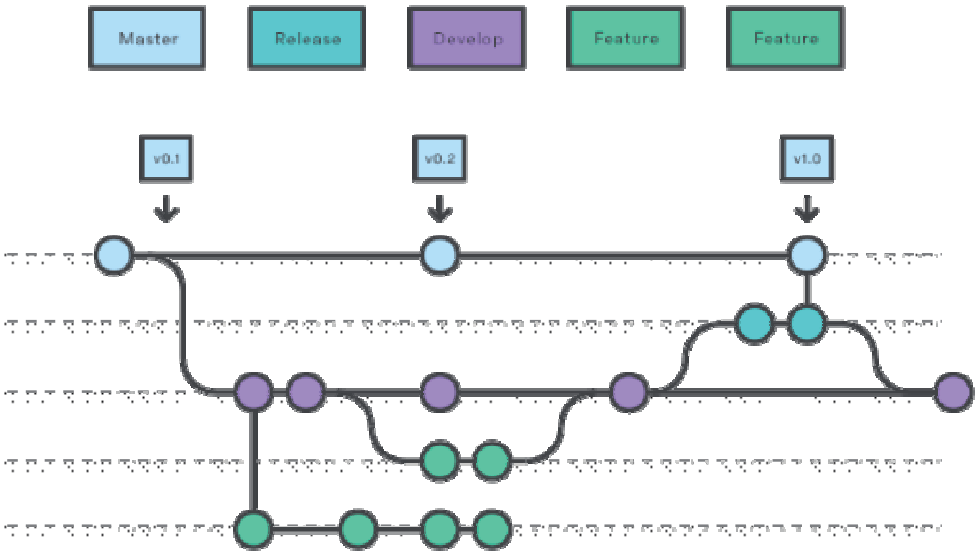
Sommerville, I. (2011). Engenharia de Software. In I. Sommerville, *Engenharia de Software* (p. Pearson Education Brasil).

The OCARIoT Consortium. (2017). *Home*. Retrieved from Smart Childhood Obesity Caring Solution using IoT potential: http://www.ocariot.com/

APÊNDICE A – GITFLOW WORKFLOW

Criado e publicado por Vicent Driessen, Gitflow Workflow é um modelo de fluxo de trabalho do Git, ou seja, é uma extensão ou complemento do fluxo padrão do Git definido como um modelo estrito de ramificação projetado em torno da versão do projeto. Isso fornece uma estrutura robusta para gerenciar projetos maiores (Tutorials - Gitflow Workflow).

Figura 23 Estrutura do Gitflow Workflow



Fonte: (Tutorials - Gitflow Workflow)

Esse fluxo foi abdicado para controlar as versões deste projeto pelo simples fato de ser o mais utilizado em aplicações com ciclo de vida alto. Sua aplicação em projetos, facilita o controle sobre as ramificações (*branches* – no inglês) focando na implementação de um conjunto de características (*feature*) que ao serem finalizadas dão origem a uma versão (*release)*.

Como ilustrado na *figura 20*, o ciclo de vida de uma *feature* começa e termina na *branch develop* que concentra as alterações em sua linha do tempo até que um *release* seja iniciado e finalizado, funcionando como um controle de qualidade antes de chegar na ramificação raiz (*master*), onde se firma o que foi desenvolvido e revisado.