UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

ERITON DOS SANTOS GOMES

SENSE API: UMA REST API PARA APLICAÇÕES M-HEALTH QUE UTILIZAM SENSORES VESTÍVEIS PARA OBTER INFORMAÇÃO DE SAÚDE DE PESSOAS

CAMPINA GRANDE
2018

SENSE API: UMA REST API PARA APLICAÇÕES M-HEALTH QUE UTILIZAM SENSORES VESTÍVEIS PARA OBTER INFORMAÇÃO DE SAÚDE DE PESSOAS

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

Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Frederico Moreira Bublitz.

CAMPINA GRANDE
2018

G633s Gomes, Eriton dos Santos.

Sense API [manuscrito] : uma REST API para aplicações *m-Health* que utilizam sensores vestíveis para obter informação de saúde de pessoas / Eriton dos Santos Gomes. - 2018.

56 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. Frederico Moreira Bublitz , Departamento de Computação - CCT."

1. REST API. 2. Requisitos de software. 3. Scrum. 4. SENSE API. 5. Tecnologias vestíveis.

21. ed. CDD 005.3

Elaborada por Giulianne Monteiro Pereira - CRB - 15/714 BC/UEPB

Eriton dos Santos Gomes

SENSE API: UMA REST API PARA APLICAÇÕES M-HEALTH QUE UTILIZAM SENSORES VESTÍVEIS PARA OBTER INFORMAÇÃO DE SAÚDE DE PESSOAS

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 28 de Junho de 2018.

Prof. Dr. Frederico Moreira Bublitz (DC - UEPB)
Orientador(a)

Prof. Dr. Andrei Guilherme Lopes (DEF - UEPB)
Examinador(a)

Prof. Dr. Wellington Candeia de Araújo (DC - UEPB)
Examinador(a)

Aos meus pais, pela educação, paciência, zelo e amizade, DEDICO.

AGRADECIMENTOS

À Deus pelo dom da vida, por me dar sabedoria, entendimento e, sobretudo, um amor inestimável.

À minha mãe Marluce dos Santos Gomes, por tudo o que ela fez e ainda faz por mim; educação, zelo, cuidado e amor pelos filhos. Sem ela eu não poderia ter desenvolvido este trabalho.

Ao meu pai Ednaldo Gomes da Costa, pela educação dada, conselhos, amizade e todo o seu tempo em labuta para sustentar a família.

À minha avó (*in memoriam)*, que embora fisicamente ausente, desde cedo me incentivara a estudar, além de ter passado sua educação pelo próximo, bem como me ensinou a cuidar e amar meus pais.

Às minhas tias, Vera Lúcia e Luciene Aguiar, as quais estavam ao lado da minha mãe como ensinara a minha avó, dando-me conselhos.

Ao meu orientador, Frederico Moreira Bublitz, por seu tempo dedicado à orientação, pelas lições sobre pesquisa científica e pela compreensão no decorrer da disciplina, a qual contribuíra para o desenvolvimento deste trabalho.

À universidade, por oferecer um curso de nível superior de muita importância para a formação do pensamento científico, na qualificação profissional, inclusive nas oportunidades para desenvolver a capacidade de docência.

RESUMO

Uma REST API ou Interface de Programação de Aplicativos REST é um tipo de servidor da Web que permite aos clientes, automatizados ou operados por usuários, acessarem recursos que modelam dados e funções dos sistemas computacionais. Uma Web API permite simplificar e desacoplar a arquitetura cliente-servidor, a interface do usuário com as regras e modelo de domínio da aplicação, bem como simplifica a comunicação entre eles, deixando a linguagem mais próxima do entendimento humano: o protocolo HTTP é usado para isso. Tem-se como objetivo geral desse trabalho o desenvolvimento de uma API REST, denominada Sense API, responsável por disponibilizar serviços web para a comunicação das aplicações *m-Health*, cuja finalidade principal é fornecer um mecanismo capaz de gerenciar o armazenamento dos dados referentes ao monitoramento de saúde de pessoas obtidos por meio dessas aplicações, de modo que este mecanismo de armazenamento integre numa única solução as informações de vários dispositivos vestíveis. As Web APIs tornaram-se parte integrante de modernas aplicações, sejam elas aplicações para dispositivos móveis – aplicativos – ou grandes aplicações Web. Elas são importantes para cenários de integração de programas computacionais em empresas e na computação em nuvem.

Palavras-Chave: Web API. REST. M-health. Tecnologias vestíveis.

ABSTRACT

A REST API or REST Application Programming Interface is a type of Web server that allows automated or user-operated clients to access features that model data and functions of computing systems. A Web API allows simplifying and decoupling the client-server architecture, the user interface with the rules and domain model of the application, and simplifies the communication between them, leaving the language closer to human understanding: the HTTP protocol is used to that. The goal of this work is the development of a REST API, called Sense API, responsible for providing web services for the communication of m-Health applications, whose main purpose is to provide a mechanism capable of managing the storage of health data monitoring obtained by theses apps, so that this mechanism integrates in a single software solution the information from various wearable devices. Web APIs have became an integral part of modern applications, whether they are softwares for mobile devices - applications - or large Web applications. They are important for integration scenarios of computer softwares in enterprise computing and cloud computing, and especially when talking about the app development.

Keywords: Web API. API. REST. M-health. Wearable Technologies.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Program Interface
APP	Application
E-Health	Eletronic Health
HTML	HyperText Markup Language
HTTP	HypeText Transfer Protocol
JSON	JavaScript Object Notation
M-Health	Mobile Health
REST	Representational State Transfer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

As Tecnologias da Informação e Comunicação contribuíram para a criação de uma recente prática de cuidados a saúde suportada por processos eletrônicos e comunicação. Trata-se da *Eletronic Health* ou comumente chamada de *e-Health*.

Segundo a *World Health Organization* (WHO, 2018), em português, Organização de Saúde Mundial, *e-Health* é o uso das tecnologias da informação e comunicação para a saúde. Para Eysenbach, 2001, *e-Health* pode ser definida como um campo emergente na intersecção da informática médica, saúde pública e negócios, no que diz respeito aos serviços de saúde e informações prestadas ou reforçadas através da Internet e relacionados a tecnologias.

Ainda assim, o advento dessas tecnologias da informação e comunicação, as quais, hoje, fazem parte do cotidiano das pessoas, em especial os dispositivos móveis, a exemplo dos *smartphones*, *tablets* e PDA's, permitiu criar uma especialidade de *e-Health*: a *m-Health*.

A *m-Health,* ainda não tem uma definição estabilizada, mas ela pode ser considerada como a prática médica e de saúde pública suportada por dispositivos móveis, como *smartphones* e dispositivos de monitoramento de pacientes, assim como os assistentes digitais pessoais (PDA) e outros dispositivos sem fios (WHO, 2018, p. 14).

Na *m-Health*, existem os aplicativos de *m-Health*. Os aplicativos *m-Health* estão em todas as áreas de cuidados de saúde, como atividade física, dietas, combate a obesidade, perda de peso, sistemas de informação de saúde, autogestão da asma e controle do sono.

O foco desse trabalho está nas aplicações *m-Health* que utilizam tecnologias vestíveis com a finalidade de aferir sinais vitais no corpo humano, como a pressão arterial, temperatura corporal e frequência cardíaca. Um exemplo dessas tecnologias vestíveis são as pulseiras inteligentes – *smartbands* –, que monitoram as atividades de atletas, e os relógios de pulso inteligentes – smartwatches.

Num futuro próximo, a informação de saúde poderá ser obtida não somente por um dispositivo vestível, mas por meio de vários dispositivos vestíveis que talvez sejam úteis no sentido de prever mudanças no corpo, tendo em vista prevenir emergências.

As aplicações *m-Health* são consideradas como intermediárias, pois elas são responsáveis por receber as informações dos sensores e depois transmiti-las para o usuário. Um dos pontos críticos para essas aplicações decorre do fato de utilizarem um mecanismo para armazenamento local dos dados no próprio dispositivo móvel.

O armazenamento local dos dados limita-se à arquitetura do dispositivo, o que pode ser um problema quando se tem um volume de informações relevantes a ser armazenado. Este tipo de armazenamento local no dispositivo não é seguro, pois está sujeito ao correto funcionamento do equipamento, ao bom funcionamento do sistema operacional, bem como ao software responsável por proteger os dados contra o acesso de terceiros.

Informações sensíveis não podem ser armazenadas no dispositivo. A segunda categoria de riscos em plataformas móveis, que pode ser consultada no Top 10 Mobile Risk, diz respeito ao Armazenamento Inseguro dos Dados. Aplicativos que utilizam banco de dados SQL, como o SQLite; armazenamento de dados em arquivos XML ou arquivos manifesto; armazenamento de dados binários; armazenamento de Cookies; armazenamento de dados no cartão SD, bem como o sincronismo com serviços de computação em nuvem, podem apresentar vulnerabilidades de segurança (OWASP, 2016).

Em razão disso, esses aplicativos necessitam armazenar as informações de saúde do indivíduo em um banco de dados localizado em servidores web, os quais são seguros e oferecem uma estrutura dedicada ao armazenamento de dados.

Ainda assim, é importante que os aplicativos *m-Health* possam utilizar não somente um sensor vestível para obter a informação de saúde, mas vários. Sendo assim, os dados de cada aplicação precisam estar armazenados de forma integrada.

Portanto, para que esse procedimento seja realizado, a comunicação da aplicação com o banco de dados necessita de uma interface de comunicação intermediária a qual fornecerá serviços web capaz de receber e responder as requisições de vários aplicativos, bem como se comunicar com um banco de dados responsável pelo gerenciamento do armazenamento de dados.

Mediante ao exposto, tem-se como objetivo geral o desenvolvimento de uma API REST, denominada Sense API, responsável por disponibilizar serviços web para a comunicação das aplicações *m-Health*, cuja finalidade principal é fornecer um mecanismo capaz de gerenciar o armazenamento dos dados referentes ao monitoramento de saúde de pessoas obtidos por meio dessas aplicações, de modo que este mecanismo de armazenamento integre numa única solução as informações de vários dispositivos vestíveis.

A Web API, além de fornecer um mecanismo responsável, principalmente, por manter e processar os dados gerados por aplicações *m-Health*, também fornecerá um serviço cuja finalidade será disponibilizar informações para outros sistemas de cuidados à saúde.

O desenvolvimento dessa API poderá contribuir para o desenvolvimento de novas Web APIs para aplicações web ou mobile, pois, dado que ela foi desenhada e implementada,

servirá como base para a escolha da arquitetura, de ferramentas e frameworks a serem utilizados no desenvolvimento, da metodologia de desenvolvimento de API, bem como na definição do modelo de domínio a ser adotado.

2. REFERENCIAL TEÓRICO

Nesta parte do trabalho, será apresentada a fundamentação teórica estudada, abordando os assuntos sobre o modelo arquitetural REST e sobre API baseada no modelo arquitetural REST, bem como a metodologia ágil de desenvolvimento de software utilizada no desenvolvimento da API.

2.1 REST

Antes do início do século XXI, em 1993, após o surgimento e grande crescimento da Web, o tráfico da web superou a capacidade da infraestrutura da internet, os protocolos centrais da web não eram uniformes e não davam suporte a cache. Nesta ocasião, "A web estava ficando tão grande, rápida demais e estava caminhando para um colapso" (MASSÉ, 2011, p 2).

Nesta época, Roy Fielding, tornou-se interessado pelo problema de escalabilidade da web. Ele identificou que a escalabilidade da web era gorvernada por um conjunto de restrições, que foi agrupado em seis categorias, o qual ficou conhecido como Estilo Arquitetural da Web: *Client-server*; *Uniform interface*; *Layered system*; *Cache Stateless*; *Code-on-demand*.

Graças aos esforços de Fielding, Tim Berners-Lee e outros, no sentido de melhorar a escalabilidade da web, foi possível superar essa crise. Nos anos 2000, época da superação, Fielding nomeou e descreveu o estilo arquitetural da web na sua tese de doutorado: REST - Representational State Transfer, em português, Estado de Transferência Representacional.

As restrições do REST foram definidas, em tradução livre para o português, como Cliente-Servidor, Interface Uniforme, Sistema em camadas, Cache, Sem estado e Código sob demanda. Todas elas são apresentadas nos tópicos seguintes.

2.1.1 Cliente-Servidor

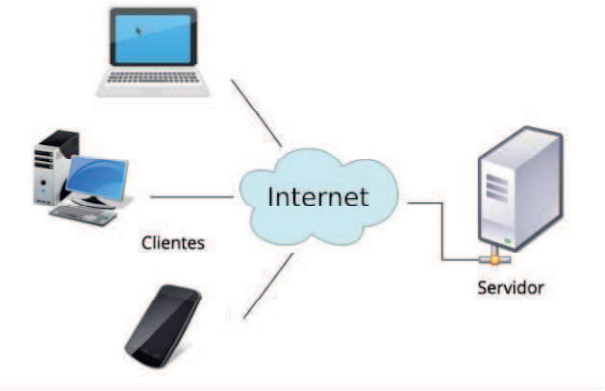
A separação de interesses é o tema central, não mais importante que os outros, mas a base fundamental da restrição Cliente-Servidor da Web. A Web é um sistema baseado em

Cliente-Servidor, no qual existem duas partes distintas, ambos implementados e implantados independentemente de idioma ou tecnologia, contudo devem seguir a Interface Uniforme da Web.

A finalidade dessa divisão consiste em separar a arquitetura e responsabilidades em dois ambientes. Primeiro, tem-se o cliente, ele não se preocupa com tarefas do tipo: comunicação com banco de dados, gerenciamento de cache, log, etc.

Depois, existe o servidor, ele não se preocupa com tarefas como: interface do usuário, experiência do usuário, etc. Permitindo, portanto, a evolução independente das duas arquiteturas.

Figura 1 – Modelo Cliente-Servidor



Fonte: Própria, (2018).

Nesta arquitetura, o servidor (provedor do serviço) recebe as requisições do cliente (consumidor do serviço) e as devolve depois de processá-las.

2.1.2 Interface Uniforme

A Interface Uniforme, basicamente, trata-se de um contrato para a comunicação entre os componentes da Web – cliente, servidor e intermediários baseados em rede –, haja vista que a comunicação entre eles depende de uniformidade de suas interfaces. Se qualquer um dos componentes se desviar das regras, o sistema de comunicação da Web falhará.

Dessa forma, para que esses componentes se comuniquem, as quatro seguintes diretrizes, referentes à interface uniforme, foram estabelecidas: Identificação de recursos,

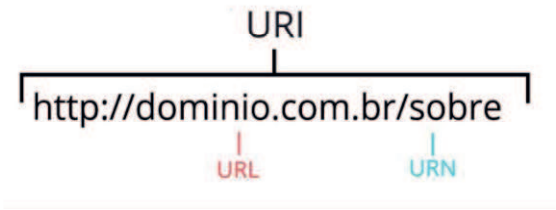
Manipulação de recursos por meio de representações, Mensagens auto-descritivas, e Hypermedia como ferramenta do estado da aplicação (acrônimo em inglês – HATEOAS).

2.1.2.1 Identificação de recursos

Na Web, os dados compartilhados entre os seus componentes são chamados de recursos. Um recurso pode ser uma página da web, um documento de texto, uma imagem, uma pasta de arquivos, ou até mesmo uma impressora. Cada recurso tem o seu identificador único.

Portanto, a identificação desse recurso é feita por uma URI – *Uniform Resource Identifier*, em português, identificador de recurso universal. Por exemplo: http://dominio.com.br/sobre.

Figura 2 – Identificador de Recurso Uniforme



Fonte: Própria, (2018).

A URI é erroneamente chamada de URL, o que não é verdade. Ela é a composição da URL com uma URN. A URL – *Uniform Resource Locator*, em português, o localizador do recurso universal, diz respeito ao local do recurso na internet, por exemplo: domínio.com.br.

Por outro lado, a URN – *Uniform Resource Name*, que em português significa Nome do Recurso Universal, refere-se ao nome do recurso que será acessado e também fará parte da URI. No exemplo acima, /sobre.

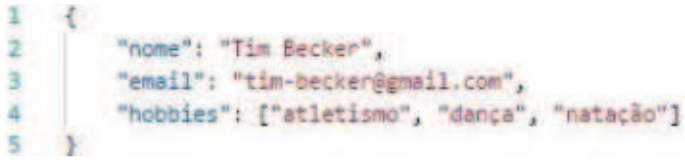
2.1.2.2 Manipulação de recursos por meio de representações

Essa diretriz diz respeito à forma de representação do recurso, ou seja, como ele vai ser devolvido para o cliente. Esta representação pode ser em HTML, XML, JSON, TXT, entre outras. Nesse caso, os clientes manipulam o recurso por meio das representações.

É importante notar, que a ideia central dessa diretriz é que a representação é a forma de interação com o recurso, mas a representação não é o recurso em si. Logo, ele pode ser representado de diferentes formas para diferentes clientes.

A representação mais comum para as aplicações, que pode ser encontrada na atualidade, é a JavaScript Object Notation – JSON. Para os seres humanos, ela é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Um exemplo de representação do recurso no formato JSON pode ser visto na Figura 3:

Figura 3 - Representação do recurso no formato JSON



Fonte: Própria, (2018).

2.1.2.3 Mensagens auto-descritivas

A mensagem auto-descritiva define que podem ser passadas informações adicionais (meta informações) sobre o estado do recurso, nas requisições e nas respostas. Algumas destas informações são: o código de status HTTP, *Host*, tipo de conteúdo, método utilizado, tamanho, formato, entre outras. Essas informações adicionais são organizadas no cabeçalho de uma mensagem HTTP.

2.1.2.4 Hypermedia como ferramenta do estado da aplicação

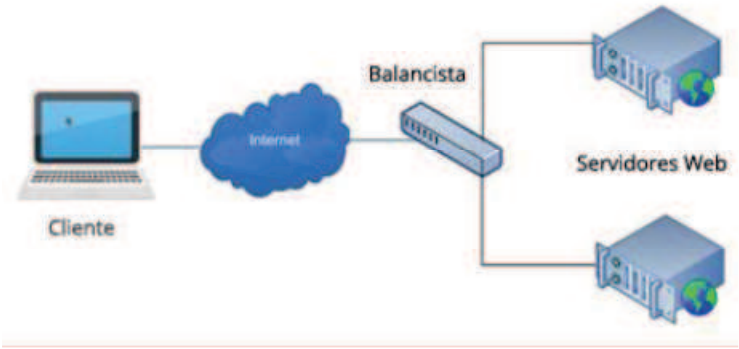
A representação do estado do recurso pode incluir links para representar recursos relacionados. Isso quer dizer que uma resposta enviada para o cliente incluirá, necessariamente, todas as informações necessárias para que o cliente saiba navegar e tenha acesso aos recursos relacionados da aplicação, isto é, um atributo adicional representando outra URI, o método utilizado e sua descrição.

2.1.3 Layered System

A restrição do sistema em camada permite que um intermediário seja transparentemente implementado entre o cliente e o servidor. Isso quer dizer que que o cliente não deve chamar diretamente o servidor da aplicação sem antes passar por um intermediador. O intermediador pode ser um balanceador de carga, um *proxy*, um *gateway* ou qualquer outra máquina que faça a interface com o servidor.

Nesse sentido, a aplicação deve ser projetada em camadas, estas camadas devem ser fáceis de alterar, tanto para adicionar mais camadas, quanto para removê-las. A Figura 4 ilustra um esquema dessa restrição.

Figura 4 – Sistema em camadas usando um balancista de carga



Fonte: Própria, (2018).

2.1.4 Cache

A restrição de cache é uma das mais importantes da arquitetura Web. Esta restrição instrui os servidores web a ter um cache dos dados de resposta, pois como muitos clientes acessam um mesmo servidor, muitas vezes requisitando os mesmos recursos, é necessário que estas respostas sejam cacheadas, evitando processamento desnecessário e aumentando significativamente a performance.

Quando um cliente solicita um recurso, o servidor armazena, temporariamente, esse dado num cache. Feito isto, quando outros clientes solicitarem o mesmo recurso, o servidor devolve o que está no cache sem ter que realizar outro processamento.

2.1.5 Sem estado

Esta restrição determina que não é necessário que um servidor da web guarde qualquer informação a respeito do estado dos seus clientes. Qualquer informação desse tipo deve ser armazenada no próprio cliente, como informações sobre sessão ou tokens de autorização.

2.1.6 Código sob demanda

Esta condição diz que os servidores web podem transferir, temporariamente, programas executáveis, como scripts ou plug-ins para os clientes. Assim, eles podem executar algum código sob demanda, ou seja, estender parte da lógica do servidor para o cliente, Java Applet, JavaScript e Flash são exemplos disso.

O código sob demanda estabelece um acoplamento tecnológico entre os servidores da Web e seus clientes, já que o cliente deve ser capaz de entender e executar o código que é baixado sob demanda. Por essa razão, o código sob demanda é a única restrição opcional.

2.2 REST API

Hoje em dia, o REST vem sendo utilizado constantemente no desenho e implementação de APIs, a qual significa Interface de Programação de Aplicativos. Com ele, uma API expõe os seus dados para uma ou várias aplicações os quais são acessados pela Web através dos serviços REST. Assim, as aplicações podem ser integradas.

O autor Massé, 2011, descreve que "uma API expõe um conjunto de dados e funções para facilitar a interação entre programas de computadores e permitir que eles troquem informações" (p. 5).

Ainda segundo ele, "O modelo arquitetural REST é comumente aplicado na criação de APIs para web services modernos. Uma Web API em conformidade com o REST é uma REST API" (MASSÉ, 2011, p. 5).

As aplicações que utilizam os serviços REST são chamadas de clientes ou programas clientes, os quais se comunicam por meio da interface de programação de aplicativos. Sendo assim, uma Web API é o rosto de um Serviço Web. Ela escuta e responde diretamente as requisições dos clientes (MASSÉ, 2011). Pode-se observar na Figura 5.

Figura 5 - API como face do Serviço Web

Fonte: (MASSÉ, p. 6, 2018).

É importante notar que o REST não é um padrão a ser seguido, ou seja, não há uma obrigação rígida quando se cria uma aplicação em rede, ele é uma espécie de guia com algumas recomendações. O que torna o REST atrativo é a possibilidade de construir aplicações cliente-servidor descentralizadas e distribuídas na web (ALLAMARAJU, 2010).

Como o REST tem, em sua base central, a restrição da arquitetura em camadas cliente-servidor, uma API REST, basicamente, pode ser considerada como um tipo de servidor da Web, o qual permite ao cliente, operado pelo usuário (usando navegadores da web) ou automatizado (por softwares), acessar recursos que modelam dados e funções do sistema.

2.3 PROTOCOLO DE COMUNICAÇÃO

Uma API REST faz uso de todos os aspectos do protocolo *HyperText Transfer Protocol* – HTTP, incluindo seus métodos de requisição, códigos de resposta, corpo da mensagem e cabeçalhos.

Nesse sentido, esta seção apresenta os conceitos fundamentais desse protocolo, já que ele é comumente utilizado na comunicação entre as aplicações, cujo modelo arquitetural é o cliente e servidor. Nesta arquitetura, uma transação HTTP é composta por uma requisição (solicitada pelo cliente) e uma resposta devolvida ao cliente (enviada pelo servidor).

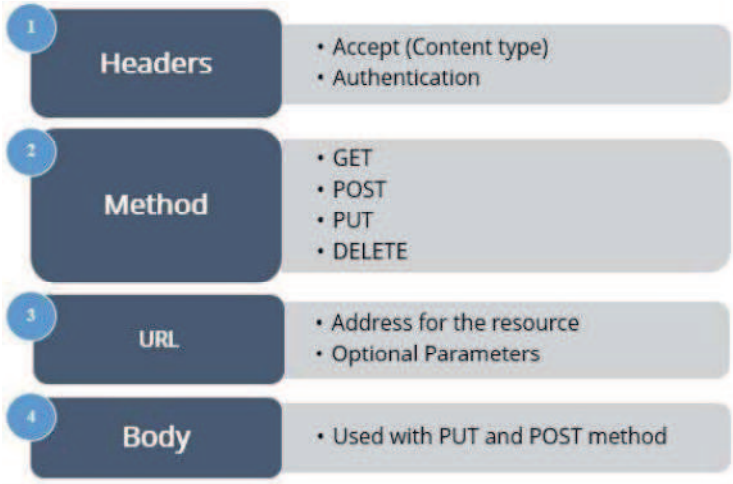
Portanto, os conceitos básicos do HTTP podem ser entendidos através do fluxo de uma requisição e resposta. As seguintes seções explicam, em alto nível, como as requisições são enviadas, quais são os métodos usados na requisição, o que eles se propõem a fazer, o que vai na lista de cabeçalho e corpo da mensagem.

Além disso, existe a resposta da requisição processada pelo servidor, ela consistirá de uma lista de cabeçalho, um código de status e o corpo da mensagem. O ciclo de uma transação usando o protocolo HTTP se concretiza ao receber uma mensagem de resposta.

2.3.1 Modelo da Requisição

Uma requisição HTTP, normalmente, é enviada ao servidor com cabeçalhos, alterando o contexto da transação. A requisição sempre tem um método escolhido; os métodos são verbos do protocolo HTTP. Além disso, existe o corpo da mensagem, que se trata do conteúdo a ser enviado. A Figura 6 ilustra a estrutura de uma requisição.

Figura 6 - Estrutura da requisição HTTP



Fonte: Adaptado. (HUNTER, 2016, p.29).

1. **Headers –** Em português, significa cabeçalhos, eles são usados na requisição para alterar o contexto da transação. Como, por exemplo, data e hora da requisição, tipo do conteúdo (text/html, application/json, etc), tamanho do conteúdo ou qual o dispositivo utilizado.
2. **Method –** Esta parte diz respeito ao método utilizado na requisição. Clientes realizam alguma operação sobre o recurso usando um conjunto fixo de métodos do protocolo HTTP: **GET, POST, PUT e DELETE**, ou seja, o cliente diz ao servidor, na requisição, qual o tipo de ação a ser realizada sobre o recurso. Os métodos mais conhecidos estão na seção 2.3.2.
3. **URI –** A Uniform Resource Identifier refere-se ao identificador único do recurso, composto do protocolo utilizado, o local do recurso acessado, junto com o nome do recurso acessado/requerido. Por exemplo: http://api.example/v1/clients/167.
4. **Body –** Em português, significa corpo da mensagem. O corpo da requisição contém o dado que o cliente quer enviar ao servidor. Nesta parte da requisição, o cliente tem total controle, pois, nela, pode ser enviado qualquer dado.

Uma requisição HTTP sempre será composta, pelo menos, por um método, que representa a ação a ser realizada, e um identificador único do recurso – URI, para a qual será enviada. Dependendo da chamada específica, talvez sejam enviadas informações adicionais no cabeçalho. Se a operação desejada for para inclusão ou alteração do recurso, o conteúdo da requisição fará parte do corpo da requisição.

2.3.2 Métodos ou verbos de uma requisição

Os métodos HTTP, ou verbos, dizem ao servidor qual o tipo de operação o cliente deseja realizar sobre o recurso acessado. Todas as ações de um CRUD – *Create*, *Read*, *Update* e *Delete* são mapeadas para um verbo específico. Os principais métodos HTTP podem ser vistos no Quadro 1:

Quadro 1 - Lista de métodos do protocolo de comunicação

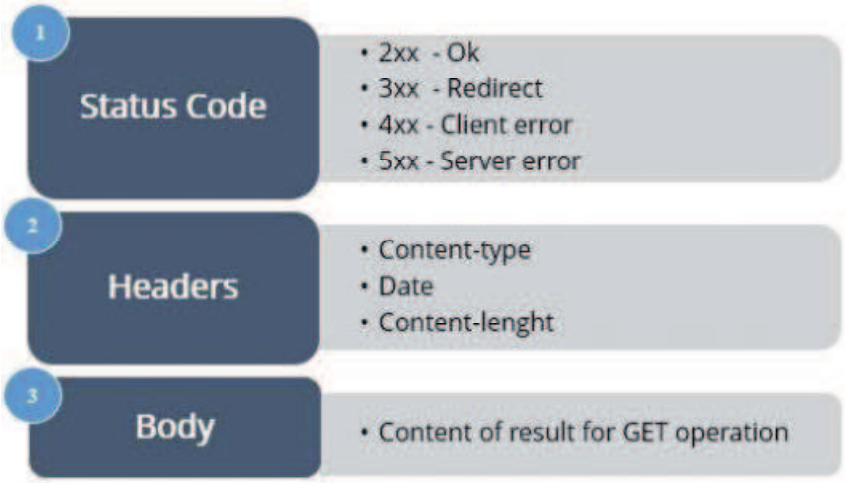
Método	Operação
GET	Seu propósito é receber (read) uma representação do estado do recurso;
POST	Deve ser usado para criar (create) um novo recurso;
PUT	Deve ser usado para adicionar um recurso, ou, frequentemente usado para atualizar (update) um recurso já existente;
DELETE	Usado para remover (delete) um recurso;
HEAD	É usado para obter informações adicionais (metadata) associadas ao recurso;
OPTIONS	Usado para saber quais são os métodos aceitos para um determinado recurso.

Fonte: Própria (2018).

2.3.3 Modelo da Mensagem de Resposta

Uma mensagem HTTP é enviada ao cliente após o servidor processar uma requisição válida. A estrutura dessa resposta é similar à requisição, no entanto a principal diferença é que em vez de ter uma URI e um método, a resposta inclui um status code, código de estado, o qual comunica o resultado da requisição ao cliente. Os demais campos, lista de cabeçalhos e corpo da mensagem, seguem o mesmo formato com os quais uma requisição HTTP trabalha. A Figura 7 ilustra a estrutura da mensagem de resposta.

Figura 7 – Estrutura da resposta HTTP



Fonte: Adaptado. (HUNTER, 2016, p.30).

1. **Status code** – O código de estado, em português, é incluído na resposta para informar o resultado da requisição processada ao cliente. Cada código tem o seu significado e, portanto, deve ser usado corretamente. Por exemplo, 200 indica que a requisição foi recebida corretamente.
2. **Headers** – Nesta parte, estão os cabeçalhos, na qual são inseridas informações adicionais como, por exemplo, data e hora da resposta, tipo do conteúdo (text/html, application/json, etc) e tamanho da resposta;
3. **Body** – Esta parte contém o conteúdo da resposta, ou seja, a representação do estado do recurso acessado. A palavra em português significa corpo.

2.3.4 Status code

Como se sabe, toda requisição tem seu método, assim como toda resposta tem um código de estado. Um status code é composto de três dígitos, o qual tem um único significado. Uma API bem projetada, deve inclui-lo corretamente na resposta, pois quando não usado corretamente, os serviços REST estarão comprometidos. Os códigos de estado estão divididos em cinco categorias distintas. O Quadro 2 descreve cada uma.

Quadro 2 - Categorias de códigos de estado

Categoria	Descrição
1xx	Comunica informação de transferência a nível do protocolo.
2xx	Indica que a requisição do cliente foi aceita com sucesso e está sendo processada.
3xx	Indica que o cliente deve executar alguma ação adicional para concluir sua solicitação.
4xx	Esta categoria indica que provavelmente o cliente cometeu um erro e precisa corrigir sua requisição antes de reenviá-la.
5xx	Indica que ocorreu algum erro no serviço ao processar a solicitação.

Fonte: Própria (2018).

Alguns exemplos básicos de status code mais usados em APIs e que consumidores de serviços REST precisam conhecer, a fim de que os serviços sejam consumidos da forma correta, estão na lista abaixo. Muitos deles são usados nas respostas quando os clientes interagem com o serviço para criar, alterar, atualizar ou consultar recursos da aplicação.

- **200 –** Na maioria dos casos, 200 é o código esperado pelo cliente. Esse código indica a API executou com sucesso qualquer ação solicitada e que nenhum código mais específico na categoria 2xx é apropriado.
- **201 –** O código 201 é utilizado pela API nos momentos em que um novo recurso é criado.
- **202 –** Este código é usado para indicar que a solicitação foi aceita, mas ainda está sendo tratada de forma assíncrona. Em outras palavras, ele é normalmente enviado para indicar ações que demoram muito para serem processadas.
- **204 –** O código 204 deve ser usado quando a requisição foi aceita, mas não há necessidade de enviar o corpo da mensagem. Este código é usado pela API em resposta as requisições do tipo PUT, POST ou DELETE. A API também pode enviar o 204 em resposta a requisição GET indicando que existe o recurso solicitado, porém não possui estado representacional a ser incluído no corpo da resposta.

- **301** – Este código deve ser usado quando a API realocar um recurso. Ele indica que uma nova URI permanente foi atribuída ao recurso solicitado e, portanto, será enviada no cabeçalho Localização.
- **400** – Indica que o cliente pode ter cometido um erro e a requisição não pode ser atendida devido ao erro de sintaxe. Neste caso, a solicitação é considerada como uma requisição ruim.
- **401** – Este código é utilizado quando o cliente tenta interagir com um recurso protegido e há um problema com as suas credenciais. Dessa forma, a API usa o código para indicar que o cliente não está autorizado.
- **403** – A API REST faz uso deste código para informar que o cliente está proibido de acessar o recurso solicitado, mas não todos.
- **404** – Deve ser usado quando a URI acessada não é válida, ou seja, o código de estado 404 indica que a URI não está mapeada para um recurso.
- **405** – A API responde com o código 405 para indicar que o cliente tentou usar um método HTTP o qual não é permitido pelo serviço.
- **500** – O código de estado de erro 500 é incluído numa resposta de erro genérica, no qual informa ao usuário que ocorreu um erro no processamento da solicitação. Neste caso, não há nada inválido na requisição do cliente, mas a solicitação precisa ser reenviada.

2.4 AGILIDADE COM SCRUM

O desenvolvimento da API foi feito utilizando o framework ágil SCRUM. O Scrum tem foco na gestão e planejamento de projetos de software. Ele "é um dos métodos ágeis mais populares na atualidade e tem foco maior nos aspectos gerenciais do desenvolvimento do software" (GOMES, 2014, p. 12).

O Scrum veio como resposta aos problemas enfrentados pelos pesados métodos de gerenciamento de desenvolvimento de software que predominavam em meados do século 90: os "métodos tradicionais". O método tradicional mais conhecido para o desenvolvimento de *software* é o modelo em cascata, ou *waterfall*.

Os métodos tradicionais são fortemente prescritivos, baseados em planos detalhados definidos no princípio do projeto, como custo, escopo e um cronograma detalhado, em processos para o sucesso do desenvolvimento do software, cada vez mais complicados, bem como em extensa documentação.

Nesse modelo, as mudanças são fortemente indesejadas, de forma geral, o *feedback* dado pelo cliente, somente ocorre após meses de desenvolvimento, depois de longas fases de análise e desenvolvimento.

Para Sabbagh, 2014,

> **Scrum** é um framework Ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Scrum é embasado no empirismo e utiliza uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto. (p. 19).

Assim como assim como outros métodos, metodologias e *frameworks*, sua utilização deve seguir os princípios e valores do Manifesto para o Desenvolvimento Ágil de Software. Portanto o *framework* mantém sempre os seguintes valores do manifesto ágil:

- **Indivíduos e iterações** mais que processos e ferramentas;
- **Software em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder a mudanças** mais que seguir um plano.

Apesar dos itens à direita apresentarem um valor, os elementos à esquerda são mais valorizados.

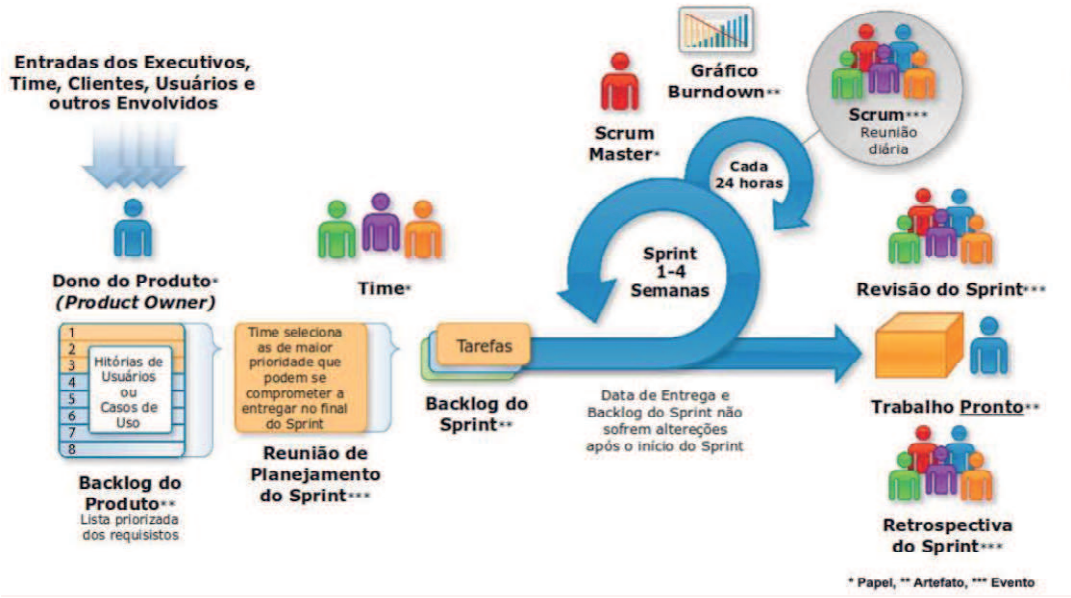
Há também um conjunto de doze princípios a serem seguidos, no entanto, o foco desse capítulo não é estudá-los à fundo. Os doze princípios ágeis são:

- Nossa maior prioridade é satisfazer o cliente por meio da entrega cedo e frequente de software com valor;
- Mudanças de requisitos são bem-vindas, mesmo em fases tardias do desenvolvimento. Os processos Ágeis utilizam a mudança em favor da vantagem competitiva para o cliente;
- Entregar software em funcionamento com frequência, desde a cada duas semanas até a cada dois meses, com uma preferência por prazos mais curtos;
- As pessoas do negócio e os desenvolvedores devem trabalhar em conjunto diariamente ao longo do projeto;
- Construa projetos em torno de indivíduos motivados. Dê-lhes o ambiente e o suporte que precisam e confie neles para realizarem o trabalho;
- O método mais eficiente e efetivo de se transmitir informação para e entre uma equipe de desenvolvimento é a conversa face a face;
- Software em funcionamento é a principal medida de progresso;
- Os processos Ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente um ritmo constante;
- A atenção contínua à excelência técnica e a um bom projeto aumentam a agilidade;
- Simplicidade - a arte de se maximizar a quantidade de trabalho não feito – é essencial;
- As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto-organizam;
- Em intervalos de tempo regulares, a equipe reflete sobre como se tornar mais efetiva e então refina e ajusta seu comportamento de acordo.

Além desses princípios e valores do manifesto ágil, o Scrum complementa suas regras com cinco valores fundamentais: foco, coragem, franqueza, compromisso e respeito (Scrum Alliance, 2012).

2.4.1 Ciclo de Vida do Scrum

Figura 8 - Ciclo de vida do Scrum



Fonte: PEREIRA et al. (2018).

A Figura 8 ilustra um esquema do *framework*. Nele, cada iteração é chamada de *Sprint*. Geralmente, cada *Sprint* tem duração de uma a quatro semanas, ao final do projeto várias sprints terão sido realizadas.

O trabalho se inicia com a criação da **pilha de backlog**, que é a lista de requisitos referentes ao sistema a ser desenvolvido. Esta lista de requisitos é organizada pelo dono do produto. Cada item desta lista é chamado de história.

Toda *Sprint* começa com uma **reunião de planejamento**. Nela, será definido o que será feito, isto é, quais funcionalidades serão implementadas, bem como quais serão as tarefas a serem realizadas pela equipe, fornecendo uma estimativa para realizar o trabalho. Somente após essa etapa, a equipe começa a trabalhar nas tarefas.

A cada dia acontece uma **reunião diária**, com a finalidade de conversar sobre o andamento da *sprint*. Recomenda-se que a reunião aconteça sempre no mesmo horário e local, no início do dia e de preferência em pé, contando com todos os membros da equipe.

Ao final de cada *sprint*, mais duas reuniões acontecem: A **reunião de revisão**, na qual a equipe apresenta ao dono do produto o trabalho que fora desenvolvido durante a *sprint*, o qual será aprovado ou não;

A **reunião de retrospectiva**, onde o foco está na melhoria do processo de desenvolvimento, pois pretende-se identifica o que está ocorrendo bem e deve ser mantido, o que precisa ser melhorado e o que realmente não dá certo e deve ser descartado.

Esses eventos do *Scrum* – Reunião de planejamento, reunião diária, reunião de revisão e reunião de retrospectiva – têm uma duração máxima ou fixa definida, chamada de *timebox*.

"Os *timeboxes* são importantes, pois evitam o desperdício, limitando o tempo em que um objetivo deve ser alcançado, além de ajudar a criar um ritmo ou uma regularidade no trabalho realizado" (SABBAGH, 2013, p. 41).

As pessoas envolvidas no processo de desenvolvimento são divididas em três papéis fundamentais, o *Product Owner* (PO ou Dono do produto), o *ScrumMaster* e a equipe. Estes papéis fundamentais (envolvidos no time) e os documentos (artefatos) estão melhor descritos nos tópicos seguintes.

2.4.2 Papéis Fundamentais

"Os envolvidos são os componentes necessários para manter a flexibilidade e produtividade do projeto. Para que isso ocorra, deve-se trabalhar em iterações e de maneira auto-organizável." (COUTINHO et al., 2012).

As pessoas envolvidas no processo de desenvolvimento desempenham um dos três papéis fundamentais:

***Product Owner*:** Maior interessado no software, represente quem tem ou teve a necessidade do produto, e por isso, tem a visão do produto. Responsável por priorizar e definir os requisitos que deverão ser entregues em cada sprint;

***ScrumMaster*:** chamado também de facilitador, ele é o profissional responsável por assegurar o entendimento e a realização do processo, assegurando que todas as regras estão sendo aplicadas, removendo, se necessário, os impedimentos para que a sprint seja realizada. O Scrum Master não atua como chefe ou gerente, pois a equipe é auto-organizável.

Time: conjunto de profissionais que trabalha no desenvolvimento do projeto, normalmente, composto de cinco a nove pessoas, *cross-funcional*, isto é, há pessoas com perfis de testadores, desenvolvedores, designers, analistas de negócio e outros. A equipe implementa as funcionalidades que foram selecionadas em cada iteração.

2.4.4 Artefatos

Backlog do Produto: Um conjunto de itens que devem ser implementados, normalmente, representados por histórias de usuários ou casos de uso, que darão origem ao produto, ordenadas por prioridade de implementação, cuja responsabilidade é do *Product Owner*. A finalidade de um backlog é ser um repositório de requisitos que deverão ser desenvolvidos e entregues.

"O Product Backlog evoluirá ao longo de todo o projeto e será frequentemente modificado com a adição, subtração, reordenamento e modificação de seus itens." (SABBAGH, 2013, p. 41).

Backlog da Sprint: basicamente, é o escopo do produto que foi priorizado e selecionado para ser implementado no próximo Sprint, que, ao término da iteração, torna-se-á um incremento entregável do produto.

Burndown: é um gráfico utilizado para medir o progresso da Sprint Backlog ou Product Backlog e dá indicativos do processo de trabalho da equipe. Dessa forma, pode ser feita uma análise do consumo do esforço projetado no tempo.

3. A SENSE API

3.1 ESPECIFICAÇÃO DA API

A Sense API é uma API REST para aplicações m-Health que utiliza sensores vestíveis para obter informação de saúde de pessoas. Além de fornecer um mecanismo responsável, principalmente, por manter e processar os dados gerados pelos sensores, fazendo uso dos serviços de *cloud computing*, também fornece um serviço web cuja finalidade é disponibilizar informações para outras aplicações que não sejam *m-health*.

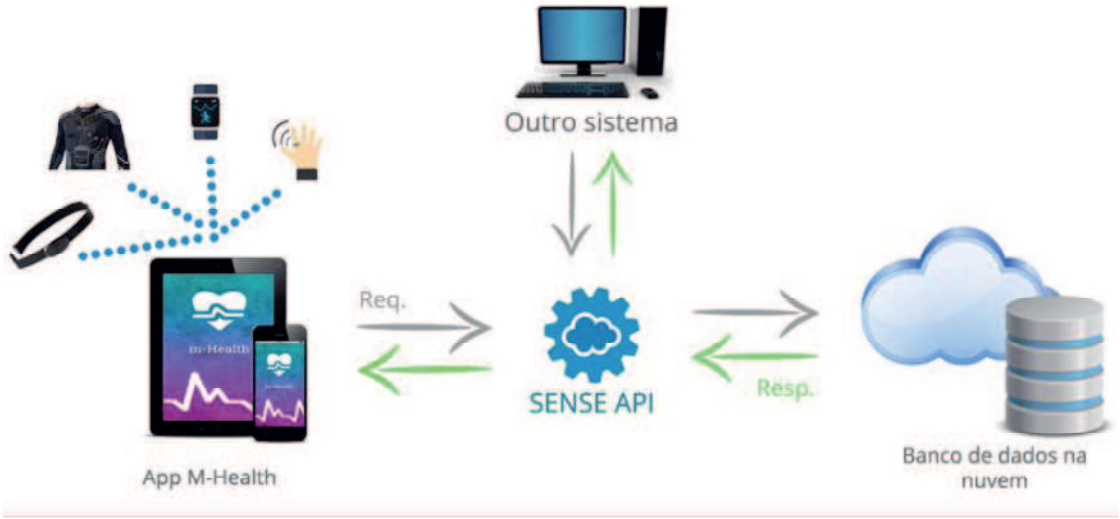
Os serviços web foram criados a fim de que o aplicativo se comunique com a API, enviando requisições e recebendo respostas sobre elas, e ,portanto, a aplicação poderá acessar os recursos localizados nos servidores de armazenamento de dados na nuvem. Estes serviços são acessados por meio da URI (também são chamados de *endpoints).*

A Sense API poderá ser acessada por várias aplicações, pois os serviços REST podem ser utilizados na integração de sistemas, principalmente entre os aplicativos móveis e as aplicações web.

A cada nova tecnologia vestível escolhida, os aplicativos poderão armazenar suas informações no mesmo banco de dados, portanto as informações de saúde de vários dispositivos vestíveis estarão integradas.

A Figura 9 ilustra o cenário de uso da API. Neste cenário, o aplicativo irá fazer uma requisição, em seguida a API se encarrega de processá-la, depois disso, será enviada uma resposta à aplicação informando o estado do recurso criado, alterado ou consultado.

Figura 9 - Cenário de uso da Sense API



Fonte: Própria (2018)

A Sense API faz uso de todos os aspectos do protocolo HTTP. Nesse sentido, as aplicações interagem com os recursos por meio das solicitações, as quais são encapsuladas numa mensagem HTTP, podendo conter a representação do recurso acessado, e, obrigatoriamente, o método escolhido da operação. Após a solicitação ser aceita, elas recebem uma resposta, isto é, quando os requisitos solicitados da interface de comunicação forem respeitados.

A Figura 10 ilustra um esquema de exemplo, no qual um objeto foi encapsulado usando a Notação de Objeto JavaScript – objeto JSON –, depois encapsulado numa mensagem HTTP, após ter sido escolhido um método, e, por fim, segue para a API.

Figura 10 - Esquema de solicitação para a API

Fonte: Própria (2018).

A figura 11 ilustra um esquema de exemplo de uma resposta à solicitação, após ter sido processada e aprovada pela API, na qual a representação do recurso está no formato JSON, e posteriormente, foi encapsulada numa mensagem HTTP. Por fim, segue para o cliente.

Figura 11 - Esquema da resposta da API

Fonte: Própria (2018).

3.2 REQUISITOS DO SOFTWARE

Os requisitos do software foram definidos na forma de User Stories, ou em português, histórias de usuários. Uma User Story ou "história de usuário" é uma descrição concisa de uma necessidade do usuário do produto (ou seja, de um "requisito") sob o ponto de vista desse usuário. Ela busca descrever essa necessidade de uma forma simples e leve.

Para Sabbagh (2013),

> ***User Story*** é a forma preferida de times Ágeis para representar cada um dos itens do *Product Backlog* que tratam de necessidades ou objetivos de negócios, descrevendo-os sob o ponto de vista dos usuários do produto e de uma forma concisa, simples e leve. (p. 41).

Para construí-la é necessário ter um **ator:** O proprietário da User Story (o interessado naquela funcionalidade); **ação:** É o que o ator quer fazer. Utilizando aquela ação ele espera alcançar seu objetivo dentro do sistema; **funcionalidade** – É o que o ator espera que aconteça ao realizar a ação. Ou seja, é o resultado de executar a ação segundo a ótica do ator (justificativa ou motivo).

Além desses requisitos, existem os Critérios de Aceitação, que são representados por uma lista de itens de negócio que expressam formas de usar a funcionalidade implementada em uma História. O objetivo dessa lista é validar se a História foi implementada de acordo com o requisito do *Product Owner*.

Após escrever a história, o *Product Owner* precisa medir o seu valor para o negócio. Uma prática bastante adotada é o jogo do planejamento, *Planning Poker*, que significa Poker do planejamento, em outras palavras, trata-se de uma técnica baseada no consenso para estimar. O valor escolhido (estimativa) corresponde ao número de uma sequência de Fibonacci – 0, 1, 2, 3, 5, 8, 13, 20, 40, 100.

Não diferente do PO, o time de desenvolvimento também estima o esforço necessário para implementar a funcionalidade, fazendo uso dos mesmos números de Fibbonacci, mesmo que seja uma estimativa aproximada, pois ao longo do desenvolvimento do projeto as estimativas ficarão mais aproximadas.

Nesse sentido, seguem os Quadros do número 3 ao 12 descrevendo os requisitos ágeis do projeto:

Quadro 3 - US Criar uma nova atividade de monitoramento

	Valor	20	**Pontos**	5
US - Criar uma nova atividade de monitoramento				
Como desenvolvedor do aplicativo, eu desejo criar uma nova atividade de monitoramento, a fim de que minhas atividades sejam salvas e estejam disponíveis para consulta.				
Testes de Aceitação				
A atividade de monitoramento deverá ter um título obrigatório; A atividade de monitoramento deverá ter uma descrição; A atividade de monitoramento terá uma data de início; A atividade de monitoramento terá uma data de finalização; Se a data de início não for informada, será informada a data atual. A cada nova atividade de monitoramento criada a api deve retornar uma mensagem HTTP informando a localização do recurso criado. Se o recurso não puder ser criado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.				

Fonte: Própria (2018).

Quadro 4 - US Alterar uma atividade de monitoramento

	Valor	20	**Pontos**	5
US - Alterar uma atividade de monitoramento				
Como desenvolvedor do aplicativo, eu desejo alterar uma atividade de monitoramento aberta, a fim de que ela passe para o estado de finalizada.				
Testes de Aceitação				
Para alterar a atividade de monitoramento, a requisição deverá conter todos os dados de identificação de recurso, inclusive, obrigatoriamente, a data e hora de sua finalização; A atividade de monitoramento somente será alterada para finalizada se existir uma atividade aberta. Após atribuir o estado de finalizada, uma mensagem HTTP deverá ser retornada ao aplicativo indicando o novo estado do recurso. Se o recurso não puder ser alterado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.				

Fonte: Própria (2018).

Quadro 5 - US – Buscar uma atividade de monitoramento por sua identificação

	Valor	13	**Pontos**	5
US – Buscar uma atividade de monitoramento por sua identificação				
Como desenvolvedor do aplicativo, eu desejo buscar uma atividade de monitoramento aberta usando sua identificação, a fim de que seja conhecido o seu estado e novos dados dos sensores possam ser enviados.				
Testes de Aceitação				
A atividade será consultada pelo número de identificação do recurso. Se o recurso for encontrado, uma mensagem HTTP, cujo conteúdo contará com um objeto no formato JSON representando o recurso, deve ser retornada ao aplicativo. Se o recurso não puder ser encontrado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.				

Fonte: Própria (2018).

Quadro 6 - US – Excluir uma atividade de monitoramento por sua identificação

Valor	13	**Pontos**	5

US – Excluir uma atividade de monitoramento por sua identificação
Como desenvolvedor do aplicativo, eu desejo excluir uma atividade de monitoramento aberta usando sua identificação, a fim de que ela não esteja mais disponível para receber novos dados dos sensores.
Testes de Aceitação
A identificação do recurso é obrigatória para exclui-lo. Uma mensagem HTTP, cujo *status code* será 204, deve ser retornada ao aplicativo. Se o recurso não puder ser encontrado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.

Fonte: Própria (2018).

Quadro 7 - US - Salvar os dados obtidos do sensor de pressão arterial

Valor	20	**Pontos**	5

US - Salvar os dados obtidos do sensor de pressão arterial
Como desenvolvedor do aplicativo, eu desejo salvar as informações obtidas pelo sensor de pressão arterial, a fim de que elas façam parte de um histórico e estejam disponíveis para consulta.
Testes de Aceitação
A pressão arterial estará atrelada a uma atividade de monitoramento; A requisição deverá conter um valor representando a pressão arterial diastólica; A requisição deverá conter um valor representando a pressão arterial sistólica; A requisição deverá conter data e hora da aferição; A cada nova pressão arterial salva, a API deve retornar uma mensagem HTTP informando a localização do recurso criado. Se o recurso não puder ser criado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.

Fonte: Própria (2018).

Quadro 8 - US – Buscar histórico do sensor de pressão arterial por atividade de monitoramento.

Valor	13	**Pontos**	5

US – Buscar histórico do sensor de pressão arterial por atividade de monitoramento.
Como desenvolvedor do aplicativo, eu desejo buscar o histórico das informações do sensor de pressão arterial por atividade de monitoramento, pois o histórico será útil para prevenir futuras complicações de saúde.
Testes de Aceitação
O histórico será consultado pelo número de identificação da atividade de monitoramento. Se não houver atividade de monitoramento aberta, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida. Se houver registro de pressão arterial, uma mensagem HTTP, cujo conteúdo da mensagem contará com um objeto no formato JSON, o qual representa a lista de recursos armazenados, deve ser retornada ao aplicativo. Se não houver registro, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.

Fonte: Própria (2018).

Quadro 9 - US - Salvar os dados obtidos do sensor de frequência cardíaca

Valor	20	**Pontos**	5

US - Salvar os dados obtidos do sensor de frequência cardíaca
Como desenvolvedor do aplicativo, eu desejo salvar as informações obtidas pelo sensor de frequência cardíaca, a fim de que elas façam parte de um histórico e estejam disponíveis para consulta.
Testes de Aceitação
A frequência cardíaca estará atrelada a uma atividade de monitoramento; A requisição deverá conter um valor maior que zero e não decimal representando a frequência cardíaca; A requisição deverá conter data e hora da aferição; A cada nova frequência cardíaca salva, a API deve retornar uma mensagem HTTP informando a localização do recurso criado.

Se o recurso não puder ser criado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.

Fonte: Própria (2018).

Quadro 10 - US – Buscar histórico do sensor de frequência cardíaca por atividade de monitoramento.

Valor	13	**Pontos**	5
US – Buscar histórico do sensor de frequência cardíaca por atividade de monitoramento.			
Como desenvolvedor do aplicativo, eu desejo buscar o histórico das informações referentes à frequência cardíaca usando a identificação de uma atividade de monitoramento, pois o histórico será útil para prevenir futuras complicações de saúde.			
Testes de Aceitação			
O histórico será consultado pelo número de identificação da atividade de monitoramento. Se não houver atividade de monitoramento aberta, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida. Se houver registro de frequência cardíaca, uma mensagem HTTP, cujo corpo da mensagem contará com um objeto no formato JSON, o qual representa a lista de recursos armazenados, deve ser retornada ao aplicativo. Se não houver registro, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.			

Fonte: Própria (2018).

Quadro 11 - US - Salvar os dados obtidos do sensor de temperatura corporal.

Valor	20	**Pontos**	5
US - Salvar os dados obtidos do sensor de temperatura corporal.			
Como desenvolvedor do aplicativo, eu desejo salvar as informações obtidas pelo sensor de temperatura corporal, a fim de que elas façam parte de um histórico e estejam disponíveis para consulta.			
Testes de Aceitação			
A temperatura estará atrelada a uma atividade de monitoramento;			

A requisição deverá conter um valor inteiro e não decimal representando a temperatura corporal; A requisição deverá conter data e hora da aferição; A cada nova temperatura corporal registrada, a API deve retornar uma mensagem HTTP informando a localização do recurso criado. Se o recurso não puder ser criado, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.

Fonte: Própria (2018).

Quadro 12 - US – Buscar histórico do sensor de temperatura corporal por atividade de monitoramento.

Valor	13	**Pontos**	5
US – Buscar histórico do sensor de temperatura corporal por atividade de monitoramento.			
Como desenvolvedor do aplicativo, eu desejo buscar o histórico referente à temperatura corporal usando a identificação de uma atividade de monitoramento, pois o histórico será útil para prevenir futuras complicações de saúde.			
Testes de Aceitação			
O histórico será consultado pelo número de identificação da atividade de monitoramento. Se não houver atividade de monitoramento aberta, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida. Se houver registro de temperatura corporal, uma mensagem HTTP, cujo corpo da mensagem contará com um objeto no formato JSON, o qual representa a lista de recursos armazenados, deve ser retornada ao aplicativo. Se não houver registro, uma mensagem HTTP deve ser retornada ao aplicativo informando sobre a requisição não atendida.			

Fonte: Própria (2018).

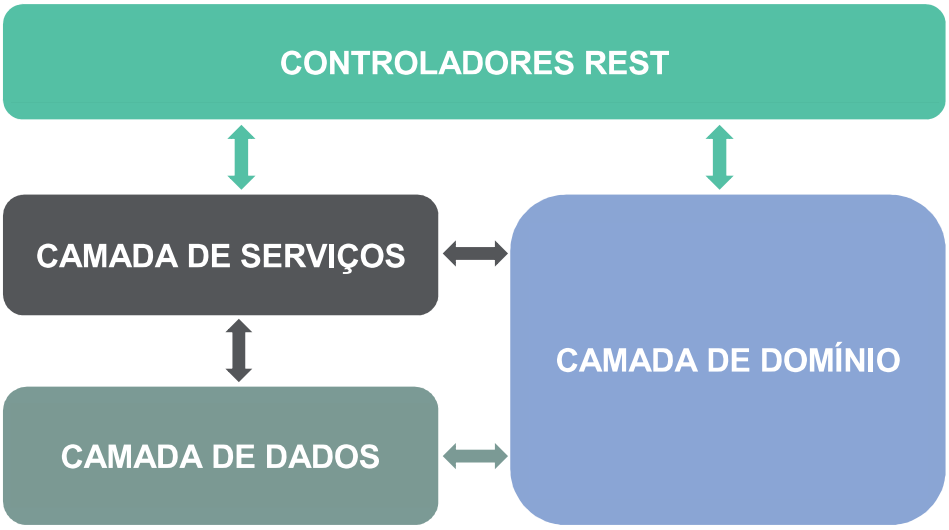
3.3 ARQUITETURA DO SOFTWARE

A API foi projetada em camadas, pois ela permite que regras de negócio, interfaces de acesso ao banco de dados, classes de serviço, classes de domínio da aplicação, classes de

exceções, e outras, estejam separadas por partes, cada uma com suas características e responsabilidades específicas.

A Figura 12 ilustra um esquema de comunicação entre as camadas, ela permite compreender a estrutura e a organização do design do sistema.

Figura 12 - Esquema de comunicação entre as camadas



Fonte: Própria (2018).

Os Controladores REST se comunicam com a camada de serviços e a camada de domínio, bem como as aplicações clientes. São eles que são responsáveis por disponibilizar as rotas da API para que as aplicações clientes acessem os recursos. Dessa forma, as aplicações não podem acessar a camada de acesso aos dados diretamente.

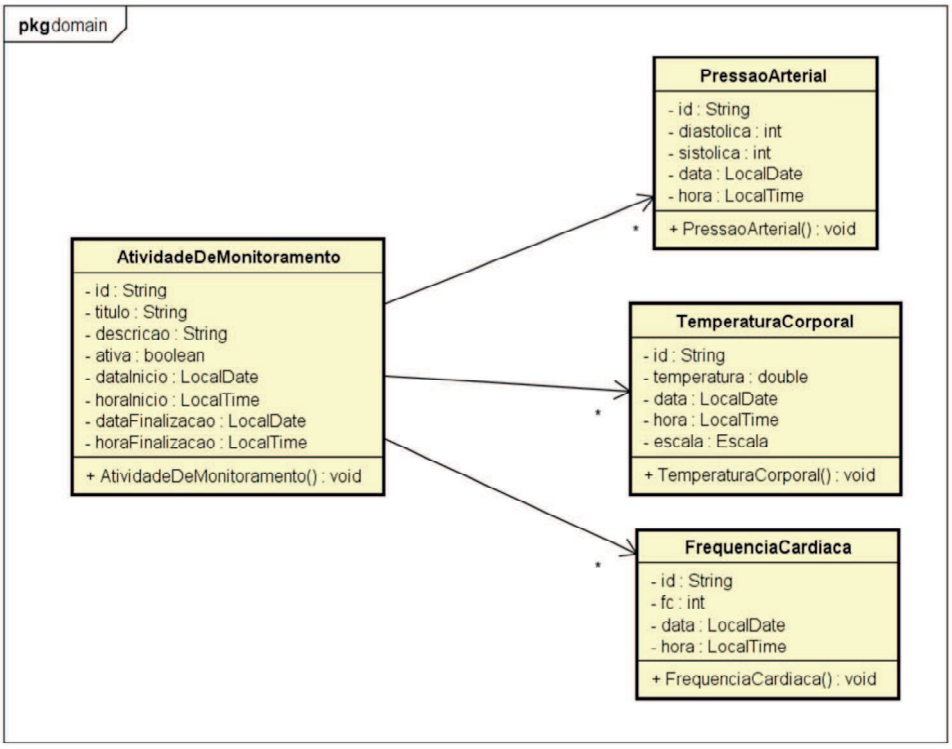
A cama de serviço é responsável por fornecer os serviços de acesso aos dados, tratamento de requisições, como, por exemplo, validadores de requisitos e gerenciamento de acesso às rotas, porventura, serviço de autenticação e autorização.

A camada da direita, a camada de domínio, representa a lógica de negócio da API, nela estão todas as entidades e objetos de valor envolvidos, sendo utilizada pela camada de serviço e os controladores REST.

Por último, a camada de acesso aos dados corresponde ao repositório de acesso aos dados, são esses repositórios que fornecem interfaces de comunicação com o banco de dados, tendo em vista o armazenamento e a recuperação das informações.

O diagrama de classes da figura 13 ilustra as principais classes utilizadas no desenvolvimento da API, elas fazem parte da camada de domínio do software.

Figura 13 - Diagrama das principais classes



Fonte: Própria (2018)

3.4 ENDPOINTS DA API

A lista de endpoints do Quadro 13 descreve quais são os serviços REST disponíveis pela Sense API, com os quais as aplicações clientes podem se comunicar com os recursos da aplicação.

Quadro 13 - Lista de endpoints da API

Método	Endpoint*	Descrição
GET	/api/v1/atividades	Obtém uma lista das atividades de monitoramento.
GET	/api/v1/atividades/{id}	Busca uma atividade de monitoramento usando o id.
POST	/api/v1/atividades	Cria uma nova atividade de monitoramento.

PUT	/api/v1/atividades	Atualiza uma nova atividade de monitoramento.
DELETE	/api/v1/atividades/{id}	Remove uma nova atividade de monitoramento.
GET	/api/v1/atividades/{id}/frequências	Obtém uma lista das atividades de monitoramento.
GET	/api/v1/atividades/{id}/pressões	Obtém uma lista das atividades de monitoramento.
GET	/api/v1/atividades/{id}/temperaturas	Obtém uma lista das atividades de monitoramento.
POST	/api/v1/frequências	Cria um novo recurso frequência cardíaca.
GET	/api/v1/frequências/{id}	Obtém o recurso frequência cardíaca por sua identificação.
GET	/api/v1/frequências/atividade/{id}	Obtém uma lista das frequências cardíacas por atividade de monitoramento.
PUT	/api/v1/frequências/{id}	Atualiza o recurso frequência cardíaca por sua identificação.
DELETE	/api/v1/frequencias/{id}	Remove o recurso frequência cardíaca por sua identificação.
GET	/api/v1/pressoes/{id}	Obtém o recurso pressão arterial por sua identificação.
GET	/api/v1/pressoes/atividade/{id}	Obtém uma lista das pressões arteriais por atividade de monitoramento.
POST	/api/v1/pressões	Cria um novo recurso pressão arterial.
PUT	/api/v1/pressoes/{id}	Atualiza o recurso pressão arterial por sua identificação.
DELETE	/api/v1/pressoes/{id}	Remove o recurso pressão arterial por sua identificação.
GET	/api/v1/temperaturas/{id}	Obtém o recurso temperatura corporal por sua identificação.

GET	/api/v1/temperaturas/atividade/{id}	Obtém uma lista das temperaturas corporais por atividade de monitoramento.
POST	/api/v1/temperaturas	Cria um novo recurso temperatura corporal.
PUT	/api/v1/temperaturas/{id}	Atualiza o recurso temperatura corporal por sua identificação.
DELETE	/api/v1/temperaturas/{id}	Remove o recurso temperatura corporal por sua identificação.

* A URI foi encurtada para facilitar a leitura.

3.5 FERRAMENTAS, TECNOLOGIAS E SOLUÇÕES UTILIZADAS

Este capítulo está destinado a apresentar as ferramentas utilizadas na implementação da API, como ambiente de desenvolvimento integrado – IDE, linguagem de programação adotada, solução de banco de dados na computação em nuvem, plataforma de computação em nuvem para a implantação e a ferramenta para a simulação do funcionamento do aplicativo e testes da API.

Quadro 14 - Ferramentas utilizadas

Ferramentas, tecnologias e outros	Descrição	Pontos positivos	Pontos negativos
Java	Java é uma linguagem de programação orientada a objetos criada desde a década de 90 na empresa Sun Microsystems	Permite portabilidade e segurança. A linguagem mais utilizada em todo o mundo.	Pode levar mais tempo desenvolvendo aplicações.
Spring Boot	Projeto do *Spring Framework*, o qual é um *framework* de código aberto para a plataforma Java, com a finalidade de	Facilitou o processo de configuração e implantação da API.	Não foi identificado.

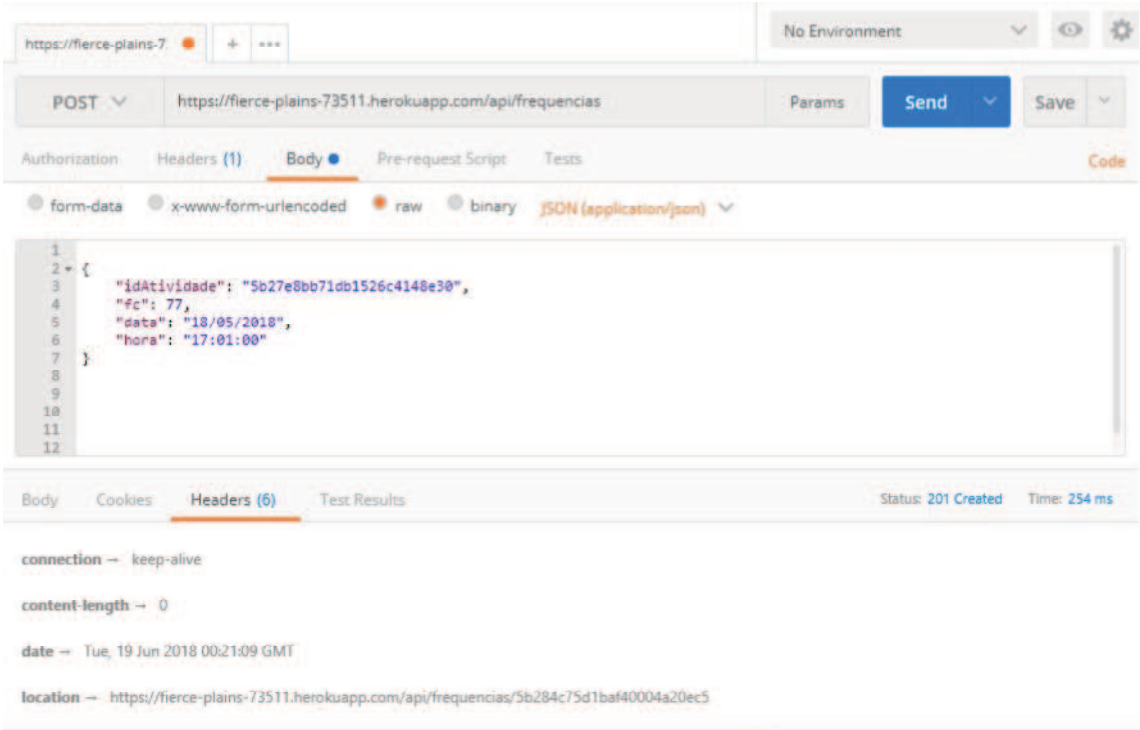
	tornar fácil a criação de aplicações.		
Spring Tools Suite	Uma IDE baseada em Eclipse para facilitar o desenvolvimento de software que utiliza os projetos do ecossistema Spring.	Facilitou a construção dos serviços REST.	Ambiente de desenvolvimento similar ao Eclipse.
Mongo DB Atlas	Um baco de dados não relacional de alta escalabilidade, o qual é disponibilizado como serviço na computação em nuvem.	Não é preciso instalar softwares no ambiente de desenvolvimento. Rápida leitura e escrita de arquivos. Esquema de dados flexível. Baixo custo com manutenção. Monitoramento das operações.	Dependendo das aplicações, os dados precisam ser compactados.
Heroku	Uma plataforma na nuvem que oferece um serviço de deploy da aplicação.	Toda a infraestrutura é fornecida. Facilita configuração de acesso com o balanceamento de carga. Torna a implantação mais rápida.	Com uma conta grátis, pode-se ter baixo poder computacional.

		Não é preciso instalar softwares no ambiente de desenvolvimento.	
		A aplicação passa a ser monitorada.	
Postman	Uma ferramenta que possibilita testar APIs, simulando o funcionamento de uma aplicação.	Simula o cliente de uma API.	
		Não precisa ser programada para realizar testes.	
		Facilita a leitura das solicitações e respostas.	
		Ferramenta gratuita.	

4. RESULTADOS

Esta seção tem por objetivo discutir os resultados obtidos com a API. Esses resultados dizem respeito à performance da comunicação entre a API e a simulação de um aplicativo utilizando a ferramenta Postman. O Postman permite que solicitações HTTP sejam enviadas à API sem a necessidade de ter uma aplicação real, e, portanto, as solicitações e respostas podem ser analisadas.

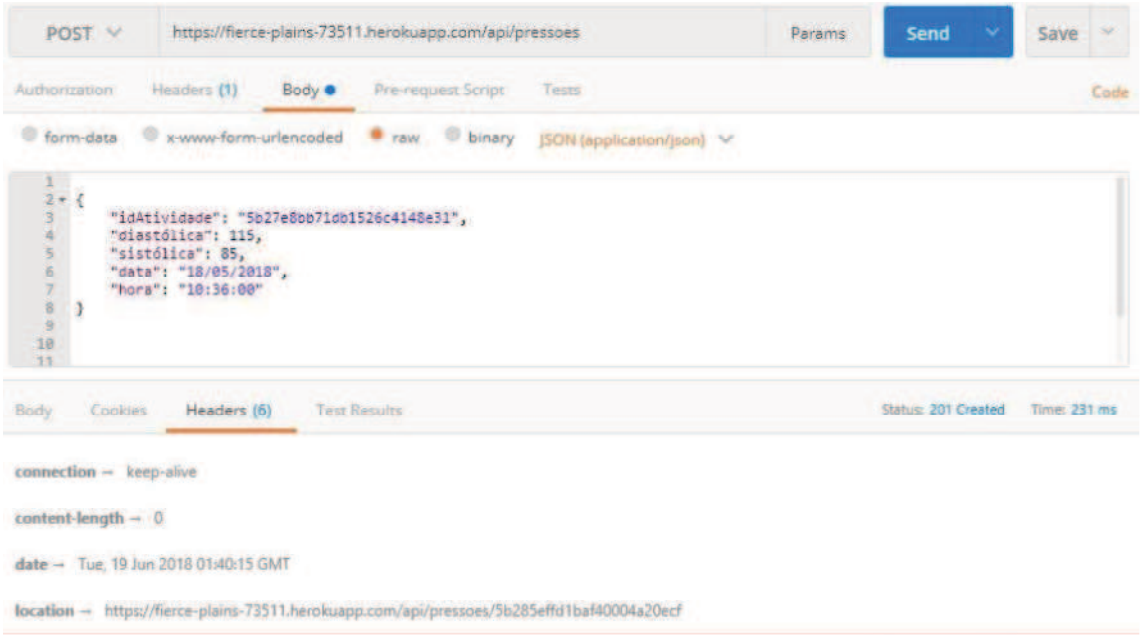
Figura 14 - Solicitação POST para criar recurso frequência cardíaca



Fonte: Elaborada pelo autor (2018).

A figura 14 ilustra uma solicitação enviada à API para criar um recurso frequência cardíaca. O tamanho da requisição foi de 287 bytes e durou 243 milissegundos para ser respondida, o que pode ser considerado quase imperceptível quando se espera uma resposta do provedor do serviço. No entanto, esse tempo pode ser melhorado ao usar um mecanismo de cache. Considerando o envio de três solicitações por segundo, por exemplo, o tempo de resposta será em média de 249,33 milissegundos. O volume armazenado por cada registro foi de 152 bytes, logo, 456 bytes por segundo.

Figura 15 - Solicitação POST para criar recurso pressão arterial

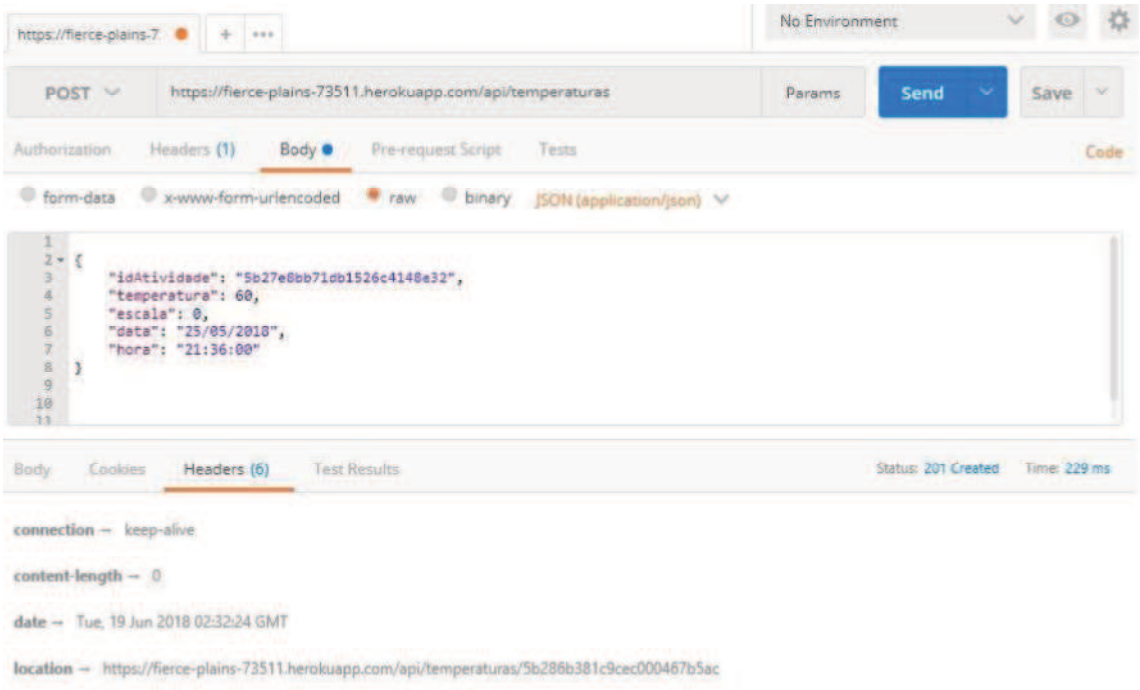


Fonte: Elaborada pelo autor (2018).

A requisição ilustrada pela figura 15 corresponde à solicitação para criar um recurso pressão arterial. O tamanho da requisição foi de 312 bytes e levou 231 milissegundos para ser respondida, o que pode ser quase imperceptível. Dado o envio de três solicitações por segundo, o tempo de resposta será em média de 230,00 milissegundos. O volume armazenado por cada registro foi de 174 bytes e, portanto, 522 bytes por segundo.

Para uma solicitação cuja operação foi a de criar um recurso temperatura corporal, ilustrada na figura 16, teve um tamanho de 11 bytes e levou 320 milissegundos para ser respondida. Considerado o envio de três solicitações por segundo, o tempo de resposta será em média de 251,00 milissegundos. O volume de cada objeto a ser armazenado no banco de dados corresponde a 183 bytes e, portanto, 549 bytes por segundo.

Figura 16 - Solicitação POST para criar recurso temperatura corporal



Fonte: Elaborada pelo autor (2018).

A requisição que teve o menor tamanho corresponde à operação com o método POST para criar um objeto do tipo frequência cardíaca e persisti-lo no banco de dados. Por outro lado, a de maior tamanho foi a da pressão arterial, como pode ser visto na Tabela 1.

Tabela 1 - Tamanho e tempo médio da requisição POST

Recurso:	URI:	Tamanho	Tempo médio:
Frequência Cardíaca	api/frequências	287 bytes	249,33 ms
Pressão Arterial	api/pressões	312 bytes	230,00 ms
Temperatura Corporal	api/temperaturas	311 bytes	251,33 ms

Fonte: Elaborada pelo autor (2018).

Os dados da Tabela 2 foram obtidos por meio do envio de três solicitações do tipo POST para criar cada recurso mencionado. O menor volume armazenado corresponde ao recurso frequência cardíaca, isto aconteceu pelo fato de ter menos valores a serem armazenados. O maior volume armazenado refere-se aos registros da coleção temperatura corporal.

Tabela 2 – Tamanho do volume a ser armazenado por coleção

Recurso:	Por seg.:	Por min.:	Por hora:	Por 24 h:
Frequência Cardíaca	456 bytes	27,36 KB	1,6416 MB	39,3984 MB
Pressão Arterial	522 bytes	31,32 KB	1,8792 MB	45,1008 MB
Temperatura Corporal	558 bytes	33,48 KB	2,0088 MB	48,2112 MB

Fonte: Elaborada pelo autor (2018).

5. CONCLUSÃO

As APIs acoplavam as interfaces do sistema computacional, mas as Web APIs apareceram para solucionar não só este problema, mas outros problemas comuns encontrados no desenvolvimento de softwares, como, por exemplo, simplificar e desacoplar a arquitetura cliente-servidor, a interface do usuário com as regras de negócio e modelo de domínio da aplicação, além de simplificar a comunicação entre o cliente e o provedor do serviço Web, tornando a linguagem mais próxima do entendimento humano.

Uma Web API bem projetada faz uso de todos os aspectos do protocolo HTTP, o que favorece a comunicação das aplicações clientes com ela. Um outro aspecto importante dessas aplicações, diz respeito a elas serem projetadas baseando-se no modelo arquitetural da Web, o REST. O REST não constitui de um conjunto de regras fixas a serem seguidas, mas sim de um tipo de guia contendo recomendações a serem seguidas. Por isso, essas APIs são constantemente desenvolvidas.

As Web APIs tornaram-se parte integrante de modernas aplicações, sejam elas aplicações para dispositivos móveis – aplicativos – ou grandes aplicações Web. Elas são importantes para cenários de integração de programas computacionais em empresas e na computação em nuvem.

Os objetivos do projeto foram cumpridos ao desenhar, implementar e implantar uma Web API seguindo os requisitos elaborados, o modelo arquitetural REST, bem como a utilização da computação em nuvem para a fase de implantação nos provedores de cloud e, também com a utilização de um banco de dados situado na nuvem, como o Mongo DB Atlas. Com a API, é possível usar outro mecanismo de persistência de dados, que não seja o local.

A API poderá produzir melhores resultados quando novos recursos computacionais mais avançados forem contratados, como maior memória RAM no servidor, disco de armazenamento rápido, mais cores de processamento e, talvez, um servidor de hospedagem da aplicação situado no Brasil, haja vista que a aplicação foi implantada no servidor situado nos Estados Unidos da América.

Vale ressaltar, ainda, que os dados enviados na solicitação e na resposta não foram compactados. Neste caso, algoritmos de compactação de texto poderiam ter sido aplicados, e, portanto, poderia diminuir o tamanho da mensagem HTTP, entretanto, poderia aumentar o tempo de respostas às solicitações.

Um ponto negativo no desenvolvimento do projeto deve-se ao fato da realização de testes de software na API de forma manual, no sentido de validar os requisitos ágeis especificados, entretanto, não houve prejuízos ao usar a ferramenta Postman; ferramentas de testes automatizados poderiam ter sido utilizadas.

REFERÊNCIAS

ALLAMARAJU, Subbu. **RESTFUL WEB SERVICES COOKBOOK**: SOLUTIONS FOR IMPROVING SCALABILITY AND SIMPLICITY. [S.L.]: O'reilly Media, 2010. 293 p.

COUTINHO, Micael et al. **Uma visão prática e ferramentas de apoio ao uso do Scrum**. Disponível em: <https://www.devmedia.com.br/uma-visao-pratica-e-ferramentas-de-apoio-ao-uso-do-scrum/26292>. Acesso em: 06/06/2018.

EYSENBACH, G. What is e-health? **Journal Of Medical Internet Research**, [s.l.], v. 3, n. 2, p.20-20, 18 jun. 2001. JMIR Publications Inc. Disponível em: <http://dx.doi.org/10.2196/jmir.3.2.e20>. Acesso em: 01/06/2018.

GOMES, André Farias. Introdução à Metodologias Àgeis. In: GOMES, André Farias. **Agile: Desenvolvimento de software com entregas frequentes e foco no valor de negócio.** São Paulo: Casa do Código, 2014. Cap. 1. p. 1-14.

HUNTER, Kirsten L. **Irresistible APIs:** DESIGNING WEB APIS THAT DEVELOPERS WILL LOVE. Nova York: Manning Publications, 2016. 232 p.

MASSÉ, Mark. **REST API DESIGN RULEBOOK**. 1. ed. Canadá: O'Reilly Media, 2011. 94 p.

OWASP Open Web Application Security Project. **Mobile Top 10 2016-Top 10**. Disponível em: <https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10>. Acesso em: 14/05/2018.

PEREIRA, Alexandre. et al. **Funcionamento do Scrum - Lição 2**. Disponível em: <http://ned.unifenas.br/cursosgratuitos/201302/scrum/funcionamento.html>. Acesso em: 17/05/2018.

SABBAGH, Rafael. **Scrum: Gestão ágil para projetos de sucesso.** São Paulo: Casa do Código, 2014. 297 p.

Scrum Alliance. **Scrum Values**. Disponível em: <https://www.scrumalliance.org/learn-about-scrum/scrum-values>. Acesso em: 01/06/2018.

WHO Global Observatory for eHealth. **MHealth: New Horizons for Health through Mobile Technologies**: Based on the Findings of the Second Global Survey on eHealth. 1. ed.

Genebra: World Health Organization, 2011. 102 p. v. 3. Disponível em: <http://www.who.int/iris/handle/10665/44607>. Acesso em: 23/05/2018.