UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO BACHARELADO EM SISTEMA DE INFORMAÇÃO

Osvaldo Miguel Junior e Robson de Carvalho

GATEWATCH - UM CENTRALIZADOR DE DADOS CAPTURADOS A PARTIR DE PULSEIRAS E RELÓGIOS INTELIGENTES

FLORIANÓPOLIS
2019

Osvaldo Miguel Junior e Robson de Carvalho

GATEWATCH - UM CENTRALIZADOR DE DADOS CAPTURADOS A PARTIR DE PULSEIRAS E RELÓGIOS INTELIGENTES

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do Título de Bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Julio da Silva Dias
Coorientador: Prof. Dr. João Bosco Mangueira Sobral

Florianópolis
2019

Osvaldo Miguel Junior e Robson de Carvalho

GATEWATCH - UM CENTRALIZADOR DE DADOS CAPTURADOS A PARTIR DE PULSEIRAS E RELÓGIOS INTELIGENTES

Trabalho de Conclusão de Curso apresentado junto ao curso Sistemas de Informação da Universidade Federal de Santa Catarina, como requisito parcial para obtenção do título de Bacharel.

Banca Examinadora:

_____

Prof. Dr. Julio da Silva Dias

Orientador

Universidade do Estado de Santa Catarina

_____

Prof. Dr. João Bosco Mangueira Sobral

Coorientador

Universidade Federal de Santa Catarina

_____

Prof. Dr. Julibio David Ardigo

Universidade do Estado de Santa Catarina

RESUMO

Com o constante crescimento da utilização de dispositivos vestíveis produziu-se um ambiente com falta de padrão na coleta e consumo dos dados originários desses equipamentos, sobretudo pelo fato de que cada dispositivo vem com uma solução própria, fornecida pelas fabricantes. Os relógios e pulseiras são, dentre os dispositivos vestíveis, os mais populares e os mais utilizados, com crescente número de vendas anual por diversas marcas que os produzem. Constantemente seguem um fluxo de troca de dados que passa por enviar dados para smartphones para, em seguida, serem enviados e processados em servidores. Porém essa troca não se apresenta de forma tão acessível já que cada relógio ou pulseira possui um modelo de dados e aplicativos próprios para realizar essas trocas dados. Nesse trabalho desenvolvemos um serviço de captura e armazenamento de dados que independe do dispositivo utilizado, de forma a facilitar o acesso a informação para softwares de empresas. Através de um aplicativo fornecido para o usuário final capturamos essas informações enviando-as para um servidor armazená-las e disponibilizá-las para empresas autorizadas.

Palavras-chave: wearable devices. internet of things, mHealth, bluetooth low energy, dispositivos vestíveis inteligentes.

LISTA DE TABELAS

Figura 1 - Os 5V´s no Big Data.....106
Figura 2 - Dinâmica de funcionamento de IoTM.....110
Figura 3 - Modelo atual de funcionamento da captura de dados.....115
Figura 4 - Modelo proposto de funcionamento da captura de dados.....115
Figura 5 - Arquitetura básica do sistema.....116
Figura 6 - Exemplo da ação do Sniffer.....118
Figura 7 - Captura no Wireshark.....118
Figura 8 - Base de arquivos do projeto.....119
Figura 9 – Arquivos de autenticação Barrier.....120
Figura 10 - Arquivos de banco de dados.....120
Figura 11 - Arquivos de controle de evento.....121
Figura 12 - Estrutura de manipulação de dados.....123
Figura 13 - Estrutura restfull do projeto.....124
Figura 14 - Controle de acesso.....125
Figura 15 - Modelagem dos dados para armazenamento.....127
Figura 16 - Modelagem de esquema e sub-esquema.....128
Figura 17 - Base de arquivos do aplicativo.....129
Figura 18 - Tela de login do aplicativo.....130
Figura 19 - Autenticação do usuário e criação de uma conta do usuário. ..131
Figura 20 - Controle de acesso de empresas.....131
Figura 21 - Informações de cadastro e bateria.....132
Figura 22 - Inclusão de novos relógios e pulseiras.....133
Figura 23 - Protótipo das telas de cadastro.....134
Figura 24 - Protótipo do login.....134
Figura 25 - Tela de login desenvolvida.....135
Figura 26 - Tela inicial do administrador.....135
Figura 27 - Interface de cadastro de empresa.....136
Figura 28 - Interface de administração das empresas.....137
Figura 29 - Interface de administração de empresa.....138

LISTA DE ABREVIATURAS E SIGLAS

IoT - Internet of Things

API - Application Programming Interface

IOS – Iphone Operation System

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

NFC - Near Field Communication

JSON - JavaScript Object Notation

DOM - Document Object Model

IOS iPhone Operating System

LABTIC - Laboratório de Tecnologias de Informação e Comunicação Aplicada

ESAG - Escola Superior de Administração e Gerência

UDESC - Universidade do Estado de Santa Catarina

UFSC – Universidade Federal de Santa Catarina

MIT - Instituto de Tecnologia de Massachusetts

BLE - Bluetooth Low Energy

RAM - Random Access Memory

BLOC - Business Logic Component

SUMÁRIO

RESUMO **3**

1 INTRODUÇÃO **101**

1.1 Objetivos 102

1.1.1 Objetivo Geral 102

1.1.2 Objetivos Específicos 102

1.2 Justificativa 102

2 FUNDAMENTAÇÃO TEÓRICA 104

2.1 Internet das Coisas 104

2.2 Big Data 105

2.3 Dispositivos Vestíveis 106

2.4 Bluetooth BLE 108

2.5 Sniffer 108

3 REVISÃO BIBLIOGRÁFICA **110**

4 FERRAMENTAS UTILIZADAS **112**

4.1 JavaScript 112

4.2 Dart 112

4.3 NodeJS 113

4.4 MongoDB 113

4.5 Frameworks 113

4.5.1 VueJS 113

4.5.2 ExpressJS 114

4.5.3 Flutter 114

5 IMPLEMENTAÇÃO 115

5.1 Dispositivos utilizados 117

5.2 Captura dos dados 117

5.3 Arquitetura do sistema 119

5.4 Modelagem dos dados 126

5.5 Desenvolvimento do Aplicativo 128

5.5.1 Arquitetura 128

5.5.2 BLoC 129

5.5.3 Interface do aplicativo 130

5.6 Interfaces de gerenciamento 133

8 BIBLIOGRAFIA 140

1 INTRODUÇÃO

O crescente o uso de dispositivos vestíveis inteligentes (*wearables*) tem chamado a atenção para os volumosos números que poderão ser atingidos em poucos anos, algo em torno de 25 bilhões de dólares em 2019 (YOON; PARK; LEE, 2016). Há um enorme potencial de aplicação para o uso de dispositivos vestíveis, sendo o maior deles o uso para medições e monitoramento da saúde (REEDER; DAVID, 2016). Dentre os dispositivos vestíveis os mais difundidos são os relógios e pulseiras inteligentes, que devido aos avanços tanto na qualidade das baterias, quanto na qualidade de suas interfaces, ou ainda sua integração com outros dispositivos conectados à Internet, estimularam uma popularização notável na última década (YOON; PARK; LEE, 2016). Além disso, sua integração para realizar pagamentos através da tecnologia *Near Fiel Communication* (*NFC*), tem ajudado nesse crescimento.

Os autores, ao prospectar essa tecnologia, perceberam a ausência de padrões para captura e coleta de dados, sobretudo com o surgimento de novas marcas, que substancialmente vem aumentando ainda mais a diversificação de dispositivos existentes no mercado. Levando-se em conta as perspectivas futuras do desenvolvimento da rede 5G (AGYAPONG et al., 2018), o avanço da Internet das Coisas (GUBBI et al., 2013) e a quantidade gigantesca de dados que precisam ser processados e armazenados na era do Big Data (CHEN; MAO; LIU, 2014), a escassez de um padrão para se trabalhar com os dados originários de dispositivos vestíveis, tanto por parte do usuário final, quanto por parte dos desenvolvedores de novas soluções para esses equipamentos, pode causar dificuldade na evolução dessa tecnologia.

Essa falta de padrão do lado do usuário final pode causar problemas, por exemplo, em situações de importação e exportação de dados (CHEN; MAO; LIU, 2014). Cada dispositivo vestível possui um aplicativo para *smartphone*, dessa forma caso o usuário queira trocar de dispositivo ele, muito provavelmente, deverá fazer toda a migração dos dados para utilizar um novo aplicativo. Já da parte dos desenvolvedores, essa excessiva quantidade de

aplicativos gera a falta de um padrão para se trabalhar com os dados, o que com o passar do tempo, com o aumento do número desses dispositivos, pode acabar dificultando o desenvolvimento de soluções.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver um sistema Web, composto por um aplicativo de celular que capture sinais de um relógio ou pulseira inteligente, independente de sua marca e modelo, armazena os dados capturados em um servidor, sobre o qual terceiros possam se cadastrar através de uma interface Web, e com a autorização do usuário, baixar os dados por meio de uma API.

1.1.2 Objetivos Específicos

- Rever a literatura sobre o tema e as tecnologias a serem utilizadas na solução;
- Desenvolver um aplicativo de celular para captura de dados, aderente a diferentes marcas e modelos de *wearables*;
- Desenvolver um servidor Web para armazenar e disponibilizar os dados coletados.

1.2 Justificativa

A tecnologia vem sendo utilizada na área da saúde ao redor do mundo (REHMAN et al., 2017), auxiliando, principalmente, profissionais da saúde no diagnóstico e tratamento de doenças (ANNAMDAS; BHALLA; SOH, 2017). Uma das tecnologias que vão a esse encontro envolve os dispositivos vestíveis, especialmente relógios ou pulseiras inteligentes, na coleta de dados para monitorar e acompanhar da evolução dos pacientes (MOLLOV, 2000).

Assim, como os dispositivos vestíveis são também utilizados no desenvolvimento de soluções para a saúde, esse trabalho se justifica por simplificar o acesso aos dados para o desenvolvimento de soluções.

Como autores deste trabalho, percebemos a dificuldade de acessar os dados de dispositivos vestíveis, ao desenvolver outras aplicações e isso nos motivou a buscar uma solução padronizada para resolver o problema da falta de padronização dos dados.

2 FUNDAMENTAÇÃO TEÓRICA

Quando se fala em relógios e pulseiras inteligentes normalmente os vemos ainda como uma extensão dos *smartphones* para melhorar a experiência dos usuários nestes equipamentos. Porém essa realidade vem mudando e esses novos dispositivos inteligentes estão cada vez mais incorporando elementos que, potencialmente, englobam os *smartphones* em si. Além disso muitos outros equipamentos *inteligentes* estão sendo criados com o surgimento de novas tecnologias, como a *Internet das Coisas*, que possibilitem novos avanços constantes.

Os avanços dos sensores existentes nos *wearables* assim como a melhora dos próprios produtos permitirão que pacientes acompanhem constantemente sua saúde, ao terem acesso aos dados de seus sinais biológicos em tempo real. Idosos poderão ser monitorados a distâncias e com muito mais controle e eficiência da real condição de sua saúde, doentes crônicos terão informações muito precisas para acompanhar suas patologias, e inúmeras outras disfunções poderão ser tratadas de forma mais precisa (HARARI, 2016).

Na sequência dessa fundamentação teórica são apresentados elementos que servem como arcabouço para essa transformação.

2.1 Internet das Coisas

A definição de *Internet* se refere à vasta categoria de aplicações que, utilizando um protocolo comum e interconectadas a uma rede de computadores, se comunicam. A evolução da internet fez o foco mudar em direção a uma integração de pessoas e dispositivos, para integrar o domínio físico com ambientes virtuais construídos pelo ser humano (BUYYA; DASTJERDI, 2016).

O termo Internet das Coisas, foi empregado pela primeira vez no ano de 1999 por Kevin Ashton. De uma forma geral, compreende-se por Internet das Coisas a maneira com que objetos físicos do cotidiano se interconectam na Web, estando muitos destes equipamentos estão incorporados com inteligência

(GUBBI et al., 2013). Os avanços em tecnologias subjacentes permitiram que “coisas” possam ser identificadas, detectadas e controladas remotamente usando sensores e atuadores (XIA; LAURENCE, 2010). Em poucas palavras, nada mais é que uma extensão da Internet atual, a qual possibilita que os objetos do dia-a-dia, com capacidade computacional e de comunicação, se conectem à Internet (SANTOS et al., 2016).

Acredita-se que todo esse ecossistema conectado, conforme a Figura 1, irá proporcionar às pessoas uma melhoria nos processos de tomadas de decisão a partir dos dados disponibilizados bem como delegar pequenas decisões que em outros momentos eram exclusivos do ser humano (LIN et al., 2017). Alguns dados preliminares estimam que atualmente o mundo implementou 5 bilhões de dispositivos inteligentes e a previsão para o ano de 2020 é que haverá 50 bilhões de objetos conectados (ALMEIDA et al., 2016).

2.2 Big Data

Normalmente o termo Big Data é utilizado na definição de grandes volumes de dados, porém sua definição está essencialmente ligada aos dados que não podem ser tratados em bancos de dados tradicionais (NAVATHE&ELMASRI, 2017).

> Big data é um conceito abstrato. Além das massas de dados, ele também possui alguns outros recursos, que determinam a diferença entre si e os “dados massivos” ou “dados muito grandes”[...]. Em geral, Big Data significa os conjuntos de dados que não puderam ser percebidos, adquiridos, gerenciados e processados pelas ferramentas tradicionais de TI e software/ hardware dentro de um tempo tolerável. Devido a diferentes preocupações, empresas científicas e tecnológicas, pesquisadores, analistas de dados e técnicos têm definições diferentes de Big Data.(CHEN; MAO; LIU, 2014).

Inicialmente o conceito de Big Data era definido por 3V´s, *velocidade*, *variedade* e *volume*. Mas, esse conceito foi expandido para os 5V´s (KATAL; WAZID; GOUDAR, 2013):

- Variedade: referente à enorme quantidade de dados originado

- Volume: o termo *Big* diz muito a respeito dessa definição. Inicialmente, os dados existentes eram medidos em gigabytes (10^9 bytes), passaram para zettabytes (10^{21} bytes) e se encaminham para yottabytes(10^{24} bytes).
- Velocidade: Devido à essa massa de dados, o processamento deve ser ágil para gerar as informações necessárias;
- Veracidade: A veracidade está ligada ao quanto uma informação obtida é verdadeira.
- Valor: Este conceito está relacionado com o valor obtido desses dados, ou seja, o quão útil ela é.

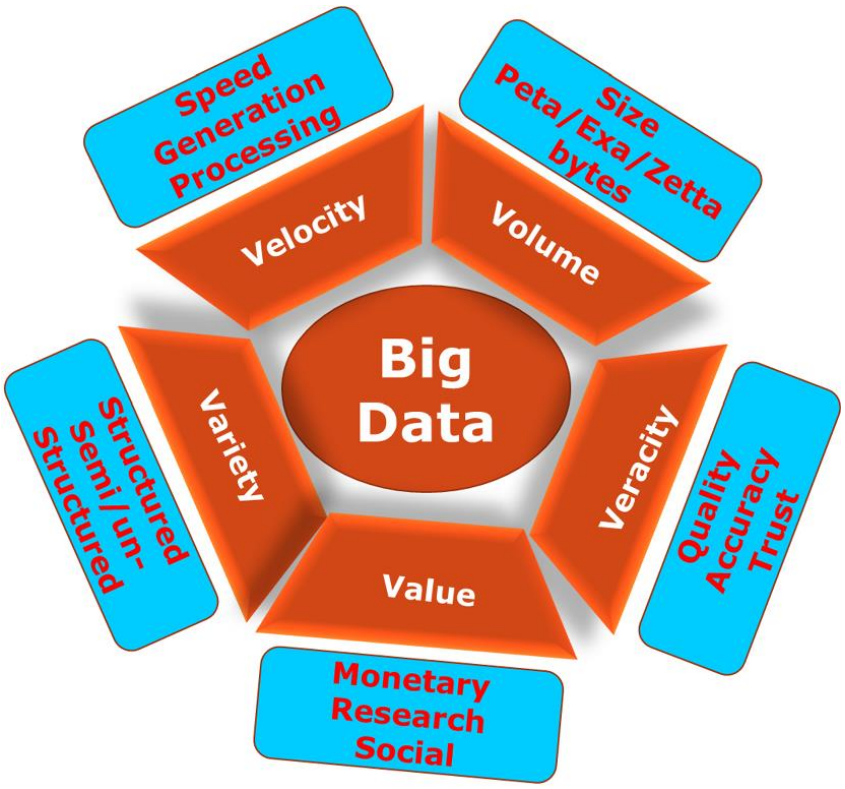

Figura 1 - Os 5V´s no Big Data

2.3 Dispositivos Vestíveis

Os *wearable devices*, também conhecidos como *dispositivos vestíveis*, englobam todos aqueles equipamentos eletrônicos que contém processadores

próprios e que podem ser usados como peças de roupa ou acessórios, permitindo aos usuários a realização normal das atividades cotidianas sem se tornarem invasivos. Parte dos *wearables* dependem de outros dispositivos, como os smartphones, para a conexão ou o processamento de parte dos dados. São várias as opções no mercado além dos relógios e pulseiras inteligentes, como anéis, peças de roupa, capacetes e óculos.

Segundo Rhodes (apud ROLIM, 2016), os dispositivos vestíveis tem as seguintes características definidas:

- **Portátil durante o uso:** pode ser utilizado durante a realização de atividade físicas;
- **Mãos livres:** uso mínimo das mãos, muitas vezes utilizando entradas e saídas de voz;
- **Sensores**: um dispositivo vestível deve ter sensores para capturar dados do ambiente ou do próprio usuário
- **Proativo**: deve comunicar ou transmitir informações mesmo que não esteja sendo diretamente utilizado notificando o usuário
- **Sempre ativo**: estar sempre ligado e sempre funcionando ou outras aplicações

A computação vestível possui diversas áreas de utilização. No esporte pode ajudar na coleta de informações dos atletas através do auxílio de acelerômetros, giroscópios, magnetômetros e outros sensores. Na educação pode ser empregado no auxílio de ensino e também na realização de testes, como no caso do projeto Sistema de Monitoração Contínua, do LABTIC/UDESC, que possibilita a realização de provas a distância, que serviu como base no desenvolvimento desse trabalho. Através da verificação dos batimentos cardíacos dos alunos, um aplicador de prova poderia ser notificado sobre eventuais alterações no nível de estresse do aluno, alertando-o para uma possível tentativa de trapaça desse aluno durante a realização da prova.

Outra área de atuação é a saúde, onde as inovações em dispositivos vestíveis para cuidados de saúde em casa podem melhorar a usabilidade, a eficiência e a popularidade da telemedicina. Esses dispositivos não apenas

permitem o monitoramento de informações fisiológicas a longo prazo, contínuo e sem obstruções, como frequência cardíaca, pressão arterial, saturação de oxigênio no sangue e respiração, mas também pode fornecer uma indicação mais realista do estado de saúde do paciente e informações inacessíveis em contextos clínicos (HUNG; ZHANG; TAI, 2004).

2.4 Bluetooth BLE

Bluetooth Low Energy é uma tecnologia de comunicação de curto alcance que apareceu na versão 4.0 do Bluetooth, em 2010, sendo projetado como uma solução de baixa potência para aplicativos de controle e monitoramento. Ela permite diminuir os níveis de consumo de energia em aparelhos que não precisam transmitir grandes volumes de dados, podendo apresentar um gasto energético de apenas 10% em relação ao Bluetooth clássico (EMERGING; TECHNOLOGY, 2012) Por meio de sua eficiência energética, o BLE rapidamente tornou-se uma das principais opções para as aplicações de Internet das Coisas. Os dispositivos que funcionam via BLE conseguem trabalhar por longos períodos sem que a bateria se esgote, transmitindo dados periodicamente.

E trata-se de um protocolo que exige pouco custo de implementação, a estrutura física necessária para a detecção de uma *tag* BLE não é muito complexa e tem custo reduzido (CHO et al., 2015).

2.5 Sniffer

São programas que capturam pacotes de uma rede de computadores. Seu propósito legal é analisar o tráfego da rede e possibilitar a identificação de potenciais áreas vulneráveis ou problemáticas(QADEER et al., 2010). Em redes locais, os dados trafegam de uma máquina para outra ao longo do cabo de rede em pequenas unidades chamadas frames. Esses frames são divididos em seções que carregam informações específicas. (“Sniffers de Rede”, [s.d.]). Posteriormente esses dados coletados são avaliados com o fim de encontrar possíveis vulnerabilidades dentro de uma rede. No contexto desse trabalho o

sniffer será considerado pois será necessário capturar dados transmitidos pelos dispositivos vestíveis.

3 REVISÃO BIBLIOGRÁFICA

Essa seção contém alguns artigos estudados durante a execução desse trabalho, sendo os resumos contidos nele sendo os mais relevantes no âmbito do trabalho, para tanto os utilizamos como um norte.

Um primeiro trabalho, com o título **DISPOSITIVOS VESTÍVEIS NA INTERNET DAS COISAS MÉDICA: PESQUISA CIENTÍFICA E DISPOSITIVOS COMERCIALMENTE DISPONÍVEIS,** mostra a importância do uso da Internet das Coisas juntamente com o uso de dispositivos vestíveis para a coleta de dados de saúde das pessoas. O avanço na qualidade dos sensores utilizados em relógios e pulseiras inteligentes, juntamente com a eficiência da telemedicina associados à integração desses com aplicativos para smartphone está aprimorando substancialmente o controle e monitoramento da saúde. O processo de criação e controle das informações seguem o fluxo mostrado abaixo.

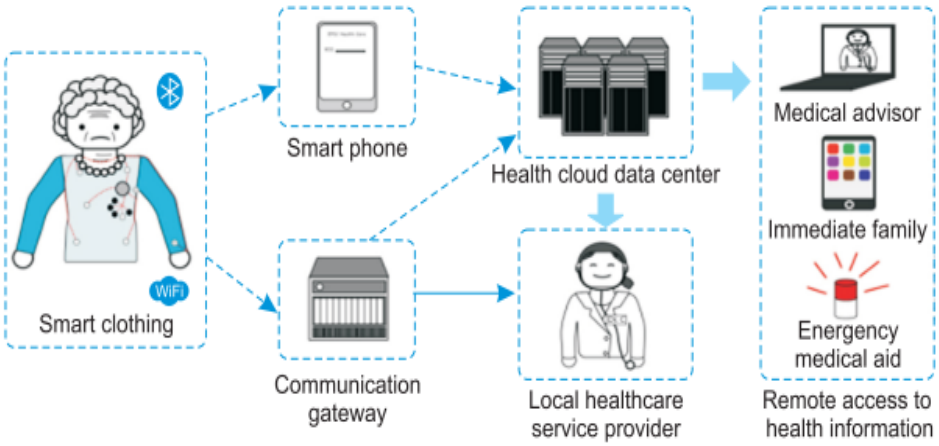

Figura 2 - Dinâmica de funcionamento de IoTM

O segundo trabalho diz respeito, diretamente a aplicação aqui proposta, com título **PULSEIRA INTELIGENTE PARA DETECÇÃO DE ESTRESSE.** Neste trabalho criou um modelo de pulseira inteligente capaz de detectar, a partir de inúmeras informações capturados a partir da própria pulseira, variações que indicam que uma pessoa está estressada ou não. Através de um *wearable*, há diversos indicadores que são capturados para descobrir se uma pessoa está estressada, ou ainda, qual o seu nível de estresse. Alguns desses fatores são: dilatação da pupila, variação do batimento cardíaco, tônus muscular, eletroencefalograma e alguns outros citados pelo autor. Ao final foi demonstrado que o uso de uma pulseira inteligente é capaz de auxiliar na detecção é eficiente.

Já no terceiro e último destaca o **RECONHECIMENTO DE ATIVIDADE DE RELÓGIO INTELIGENTE UTILIZANDO APRENDIZADO ATIVO**. Demonstra que os smartphones conseguem detectar inúmeras atividades executadas por seus usuários através de seus muitos sensores. Porém há um limite de acurácia na identificação de algumas dessas atividades, de forma que os relógios ou pulseiras inteligentes acabam conseguindo melhorar esse desempenho. Um exemplo desse caso é a identificação da de uma pessoa estar bebendo água, enquanto que, em média um relógio ou pulseira possui uma acurácia em torno de 93,3%, um smartphone tem uma acurácia de apenas 77,3%. Esse trabalho mostra que para algumas atividades a eficiência dos relógios e pulseiras são muitos maiores que os dos smartphones.

4 FERRAMENTAS UTILIZADAS

4.1 JavaScript

É uma linguagem de programação usada para desenvolver aplicações, sistemas e serviços de alta complexidade. Surgiu em 1995, e tinha como objetivo manipular validações de entrada deixadas para linguagens do lado do cliente (ZAKAS, 2012).

A linguagem evoluiu e ficou bastante conhecida por seu uso para desenvolvimento de sistemas web, tornou-se uma parte tão importante da web que até navegadores alternativos, incluindo aqueles em telefones celulares e aqueles projetados para usuários com deficiência o suportam. (CECHINEL, 2017). Porém hoje a linguagem não se limita mais apenas aos navegadores, sendo utilizada para criação de aplicativos para celulares, plug-ins para outros sistemas e até para construção de sistema operacional(WISE, 2011). O JavaScript é uma linguagem interpretada, o código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado. Não há a necessidade de transformar o código em algo diferente antes do navegador executá-lo.

4.2 Dart

O JavaScript é a linguagem de programação mais importante quando se fala em desenvolvimento Web. Um do fatores mais importantes é a facilidade de se utilizar uma mesma linguagem para se trabalhar tanto no *front-end* quanto no *back-end* da aplicação(ZAKAS, 2012), baseado nesse princípio a Google, em 2011, criou a linguagem de programação chamada Google Dart.

De acordo com o site do projeto “*a Dart foi desenhada para facilmente escrever ferramentas de desenvolvimento para aplicações web modernas e capacitadas para ambientes de alta performance*” (DARTLANG.DEV, 2019)

As principais características da linguagem são: baseada em compilação de código JavaScript, baseada em classes, orientada a objetos, redigilibilidade baseada em C(DARTLANG.DEV, 2019). É *open-source*,

escalável, contém bibliotecas robustas para desenvolvimento de aplicações web, servidores e para plataformas móveis (DEVMEDIA.COM.BR, 2019)

4.3 NodeJS

Node.js é um interpretador JavaScript assíncrono orientado a eventos, projetado para construir aplicações de rede escaláveis. O NodeJS permite que desenvolvedores utilizem JavaScript para programação no lado do servidor, para gerar dinamicamente o conteúdo de uma página antes de ela ser enviada ao usuário.(Node.js Foundation, a). Construída sobre o motor V8 (JavaScript Engine) do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis e usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos(TILKOV; VINOSKI, 2010). Utiliza o que é chamado modelo de programação orientada a evento. (NODEBR.COM, 2019)

4.4 MongoDB

É um banco de dados orientado a documentos utilizam o conceito de dados e documentos autocontidos e auto descritivos, e isso implica que o documento em si já define como ele deve ser apresentado e qual é o significado dos dados armazenados na sua estrutura(ZHAO et al., 2013).

Foi desenvolvido com foco em aplicações web e visando suportar grande quantidade de dados com grande vazão de leituras e escritas no banco além de várias estratégias a fim de facilitar a replicação. Seu formato de dado utilizado é o BSON que é um conjunto estendido do JSON(WIKTORSKI, 2019).

4.5 Frameworks

4.5.1 VueJS

É um framework progressivo JavaScript que destaca-se pela sua simplicidade em executar as mesmas tarefas dos outros frameworks,

concentrando-se na utilização de componentes reutilizáveis(YOU, [s.d.]). A biblioteca principal é focada exclusivamente na camada visual (*view layer*), sendo fácil adotar e integrar com outras bibliotecas ou projetos existentes. Por outro lado, Vue também é perfeitamente capaz de dar poder a sofisticadas *Single-Page Applications* quando usado em conjunto com ferramentas modernas e bibliotecas de apoio (YOU, [s.d.]).

4.5.2 ExpressJS

Também é conhecido apenas por Express, é um dos *frameworks* mais populares para o *Node*, lançado como software livre e de código aberto sob a Licença MIT. Resolve um leque isolado de problemas comuns em todas aplicações como gerenciamento de rotas, requisição e resposta e views, através de uma estrutura simples disponibiliza uma aplicação em funcionamento com apenas algumas linhas de código (STRONGLOOP, 2019).

4.5.3 Flutter

É um framework para desenvolvimento híbrido de aplicativos utilizando a linguagem Dart como base de criação dos aplicativos. (“FLUTTER”, 2019). Flutter possui os *widgets* que seria a mesma coisa de um componente. Um widget é uma árvore que pode conter um ou mais filhos, e esses filhos são renderizados conforme a construção desta árvore. Muito parecido com o DOM do html.

5 IMPLEMENTAÇÃO

Nesse trabalho fizemos a unificação do acesso aos dados obtidos através das pulseiras e relógios inteligentes em apenas uma API. Isso facilita aos interessados em utilizar esses dados, já que precisarão acessar apenas a uma única *API* ao invés muitas outras. A ***Figura 3*** mostra o modelo atual de acesso aos dados (com inúmeras APIs) onde a obtenção de dados é feita de forma exclusiva através de uma API disponibilizada pela fabricante do dispositivo. Já a ***Figura 4*** mostra a proposta executada nesse trabalho (uma única API), que é a centralização dos dados provenientes dos dispositivos.

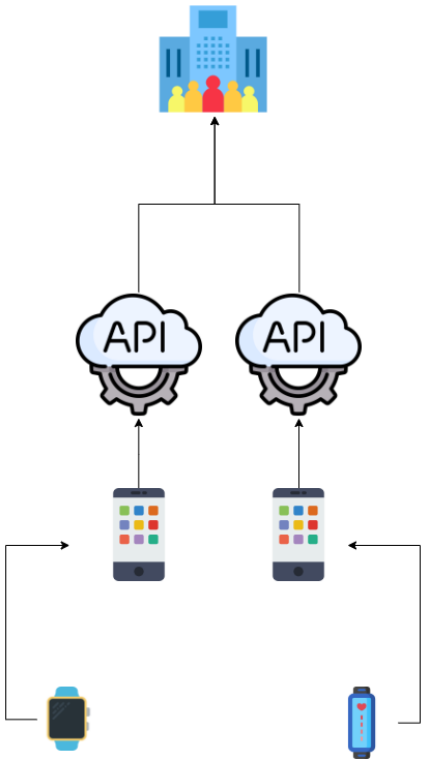

Figura 3 - Modelo atual de funcionamento da captura de dados

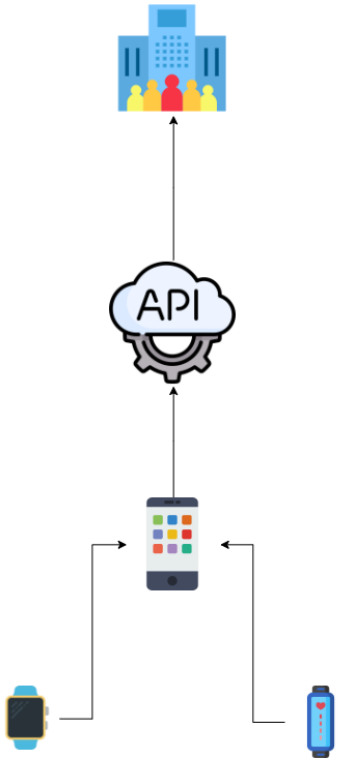

Figura 4 - Modelo proposto de funcionamento da captura de dados

Para realizarmos o trabalho criamos uma solução que, essencialmente, é um grande servidor onde esse recebe informação dos usuários do nosso sistema. O sistema se divide em um aplicativo para smartphone (IOS e Android) e um

sistema web, onde serão as entradas dos dados desse sistema. Após essa entrada, nosso servidor fará o processamento dos dados.

Os usuários do sistema foram divididos em três grupos, cada um tendo um perfil de acesso ao sistema. O primeiro grupo será chamado de Usuário, o segundo grupo Administrador e o terceiro grupo são os Empresa. Cada um desses grupos terá acesso a algumas partes específicas do sistema. A ***Figura 5*** mostra quais partes do sistema será liberado para cada um dos perfis. O Usuário representa todos os utilizadores finais dos dispositivos, pulseira ou relógios inteligentes, os quais terão acesso aos seus dados capturados por meio do aplicativo. O Administrador será alguém vinculado ao sistema que será o intermediador de todo o acesso. Já Empresa são os interessados externos que terão acesso aos dados dos Usuários para promover ou criar soluções.

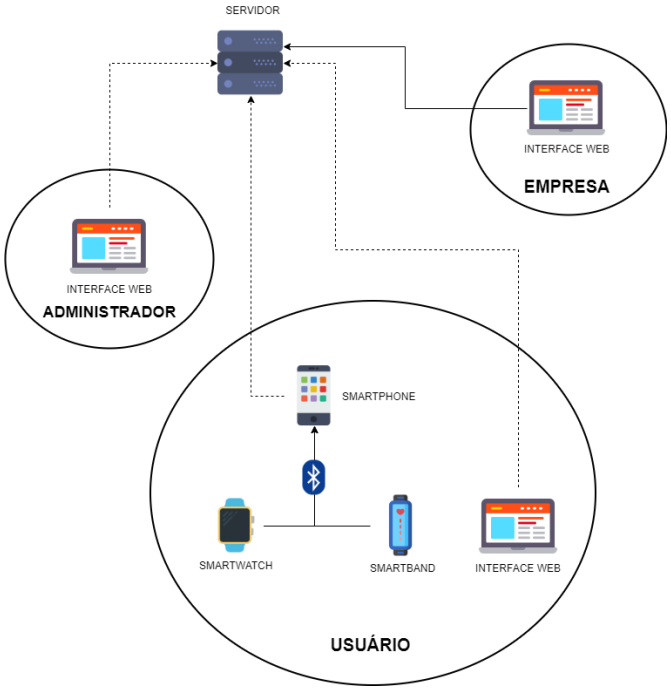

Figura 5 - Arquitetura básica do sistema

5.1Dispositivos utilizados

Para a execução dos testes propostos utilizamos dois dispositivos de mesma marca. Os dispositivos utilizados foram o Xiaomi Amazfit e o Xiaomi Miband 3, sendo utilizados sobretudo pela facilidade de acesso da equipe aos mesmos. Foram iniciados testes com outros dispositivos, porém devido ao escopo do nosso projeto acabamos não incluindo nesse trabalho. Utilizamos também um sniffer para realizar a captura dos dados transmitidos entre os dispositivos vestíveis e os smartphones.

Especificações técnicas dos dispositivos:

- Xiaomi Amazfit Verge:
- Xiaomi Mi band 3
- Nordic BLE Sniffer

5.2 Captura dos dados

A troca de dados entre os relógios ou pulseiras e o smartphone é via Bluetooth, onde os dados são capturados em um desses dispositivos e depois descarregados em um smartphone ficando armazenado inicialmente de forma local e em seguida podendo ser enviados à servidores.

Para realizar a análise do padrão de troca de dados entre esses dispositivos, foi realiza a captura de dados colocando um *sniffer* entre o smartphone e o dispositivo vestível, conforme ***Figura 6***. Nesse processo utilizamos o Wireshark, programa que analisa o tráfego de rede. Através dessa captura conseguimos obter os padrões de transmissão de dados existentes entre os dispositivos.

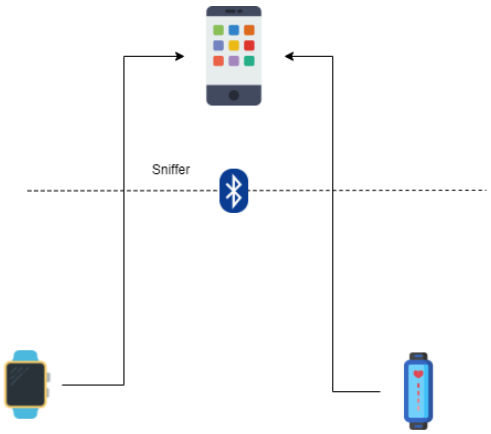

Figura 6 - Exemplo da ação do Sniffer

No que segue, a ***Figura 7*** mostra uma das capturas realizadas no trabalho utilizando o *Nordic sniffer BLE* em conjunto com o software de captura de dados *Wireshark*. Essas capturas possibilitaram o reconhecimento do padrão de troca de dados entre os dispositivos vestíveis utilizados no trabalho e smartphones.

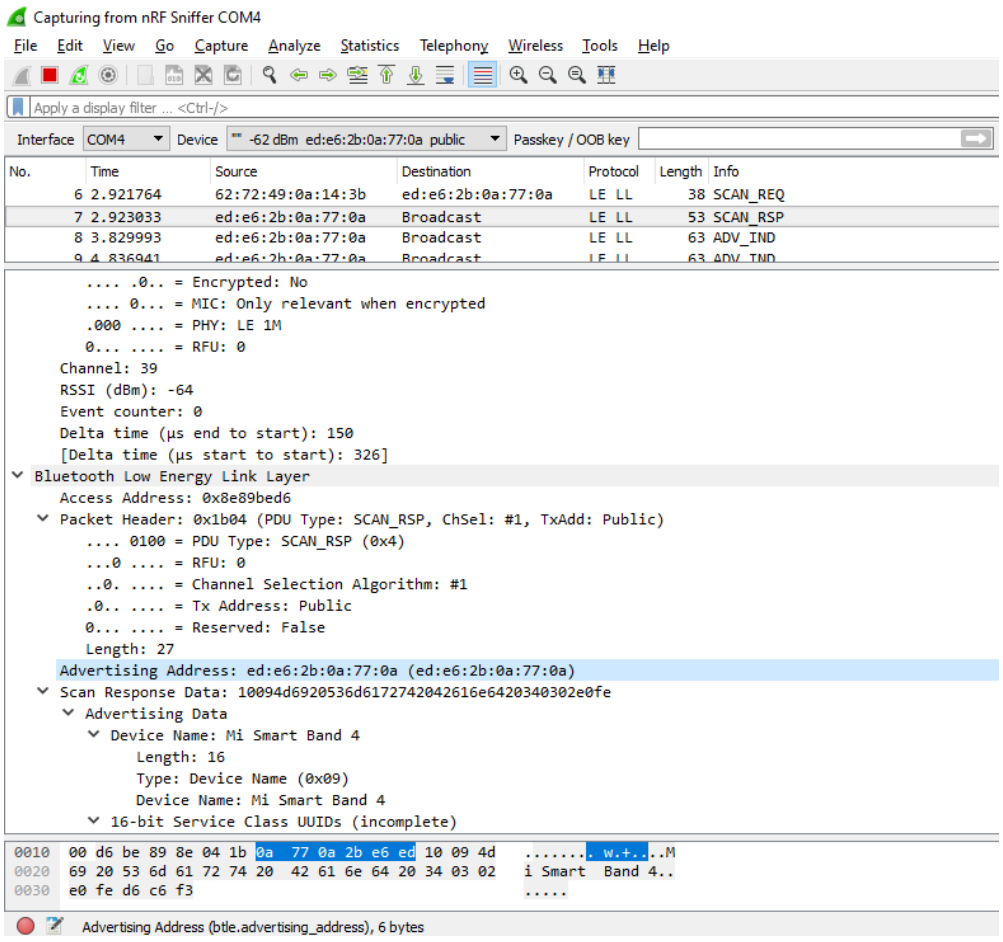

Figura 7 - Captura no Wireshark

5.3 Arquitetura do sistema

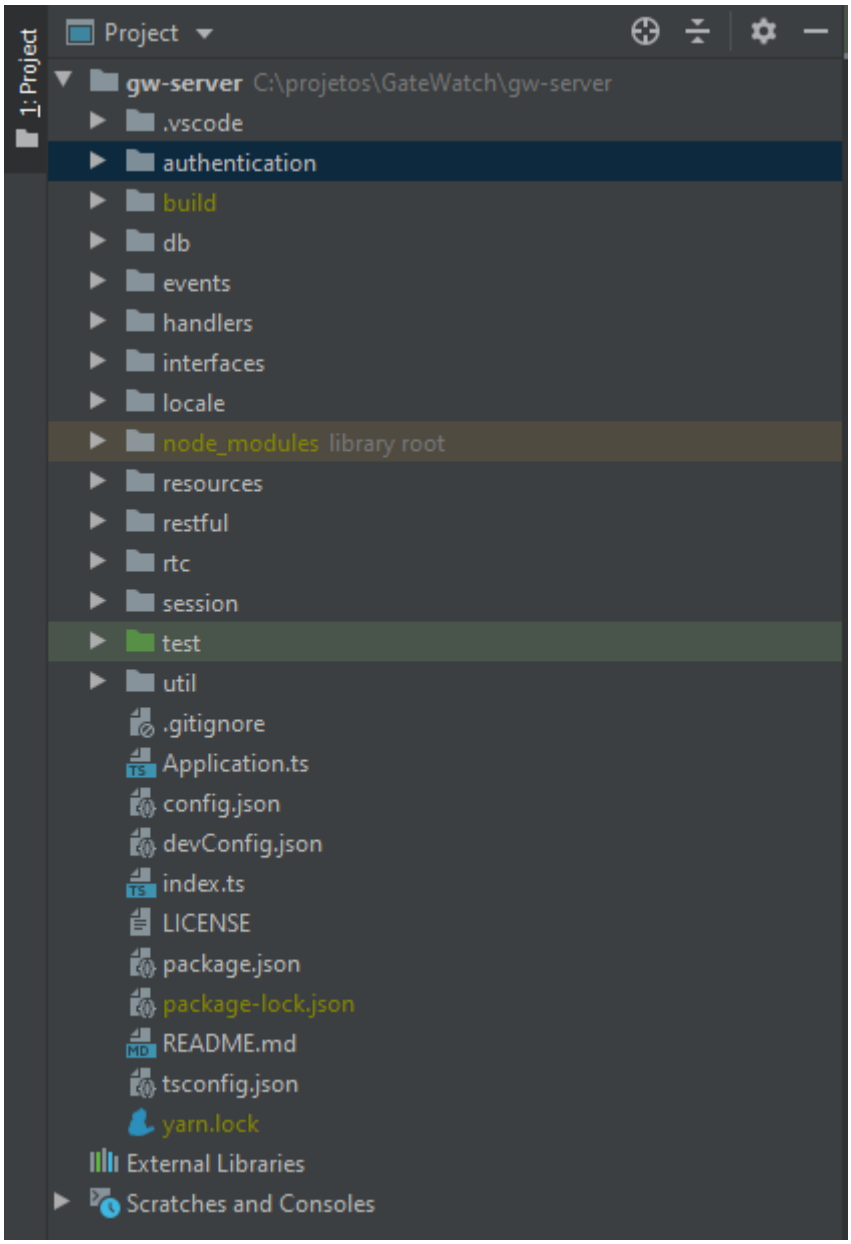

Figura 8 - Base de arquivos do projeto

A ***Figura 8*** mostra a estrutura de pastas adotada para o desenvolvimento da aplicação, essa estrutura segue um padrão estabelecido no LabTIC – ESAG/UDESC, por ser uma estrutura em constante manutenção e evolução, por diversos programadores ao longo da implantação da tecnologia *NodeJS* no local possui uma organização para facilitar a busca por documentos

e criação de novos. Abaixo explicaremos com mais detalhes sobre o conteúdo de cada pasta/arquivo e seus papéis.

- authentication: Essa pasta contém um middleware denominado Barrier.ts e um arquivo de configuração denominado BarrierConfig.json, como na ***Figura 9***.

Figura 9 – Arquivos de autenticação Barrier

- Barrier.ts: Todas as rotas de acesso serão checadas por esse elemento da estrutura, ele é responsável por verificar se essa determinada API é aberta, ou seja, não precisa de autenticação de usuário logado ou se é uma API apenas para usuário logado, se esse for esse o caso verifica se vem na requisição authenticationKey e accesskey, para serem verificadas no session/AccessController.ts posteriormente.

- BarrierConfig.json: Contém os Endpoints de APIs, e descreve se a determinada rota é static, no caso de rotas para leituras de arquivos públicos, quais Endpoints são abertos, onde todos os Paths seguido desses Endpoints estão livres de verificação de chave e APIs especificas que tem a rota livre também.

- build: Essa pasta contém todo o software transcrito em Javascript, é essa pasta que será interpretada pelo NodeJs, conforme ***Figura 10***.

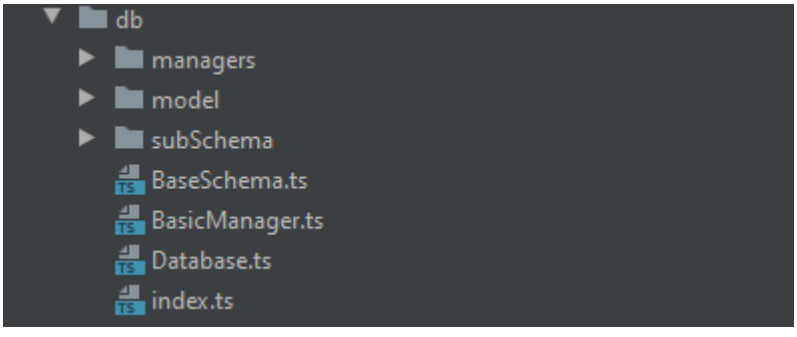

Figura 10 - Arquivos de banco de dados

- db: Essa pasta contém as pastas *managers*, *model* e *subSchema* respectivamente, além dos arquivos BaseSchema.ts, BasicManager.ts, Database.ts e index.ts.
- managers: Nessa pasta contém os arquivos com funções e controles dos *models* do Banco de Dados, eles podem conter funcionalidades especificas de um determinado *model*, ou apenas instanciar e implementar as funcionalidades herdadas do *BasicManager*.
- model: Contém os models (Schemas) que serão inseridos como *collections* no Banco de Dados, esses documentos indicam as especificidades de cada *model* do sistema, indicando quais campos dos documentos são *unique, require*, etc, além de tipos como *Arrays, Strings, Numbers* entre outros.
- subSchema: Pasta que contém *subSchemas* para serem usados por *models* ou *subSchemas*. *subSchemas*, são *subModels* sem gerenciamento, eles são usados para compor os models, pode-se dizer que um *subSchema* é um *Type* não primitivo a ser inserido dentro de um *model*.
- BaseSchema.ts: esse arquivo representa a base dos *models*, nele contém alguns campos e flags obrigatórias nos *models*, todo o *model* herda esse documento.
- BasicManager.ts: Como citado anteriormente, todo *manager* herda o BasicManager.ts, esse documento tem interfaces para todos os métodos CRUD do banco de dados, além de interfaces para aggregates e count.
- Database.ts: Documento responsável pela conexão com o banco de dados, além de iniciar o manager de cada *model* e construir a *collection* no banco.

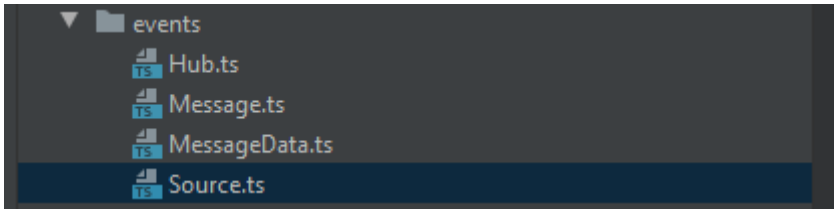

Figura 11 - Arquivos de controle de evento

- events: muitas das comunicações entre os documentos do *backend* são estabelecidas através de eventos e não por meio de *imports* e chamadas diretas de funções como por padrão na maioria das aplicações em *NodeJs*, foi optado trabalhar com eventos para se ter um maior desempenho do sistema em si, isso faz com que nenhuma classe fique ociosa, pois ela recebe uma chamada e dispara um evento quando essa chamada for resolvida por outra classe, essa chama uma função nova para tratar o retorno, isso se dá principalmente nas chamadas feitas ao manager, pois eles são classes únicas do sistema e não podem demorar para responder. Nessa pasta estão contidos documentos como *Hub.ts*, *Message.ts*, *MessageData.ts* e *Source.ts.*
- Hub.ts: Esse documento é uma representação de um hub, como o próprio nome diz, todos os eventos são enviados para ele e as classes estão ligadas e escutando nele, quando um evento é de interesse de uma determinada classe, essa resgata o evento para resolvê-lo e disparar novamente no hub, onde a classe solicitante irá resgatá-lo para consumir e tratar o retorno.
- Message.ts e MessageData.ts: Message.ts é uma estrutura criada para padronizar a comunicação feita pelos eventos, *MessageData.ts* é um padrão de data inserido dentro da *Message.ts.*
- Source.ts: Para uma classe ter acesso a estrutura de eventos ela precisa herdar o documento *Source.ts*, ele possui toda a estrutura necessária para usar eventos, é através de métodos implementados nela que as classes disparam e consomem eventos relevantes.

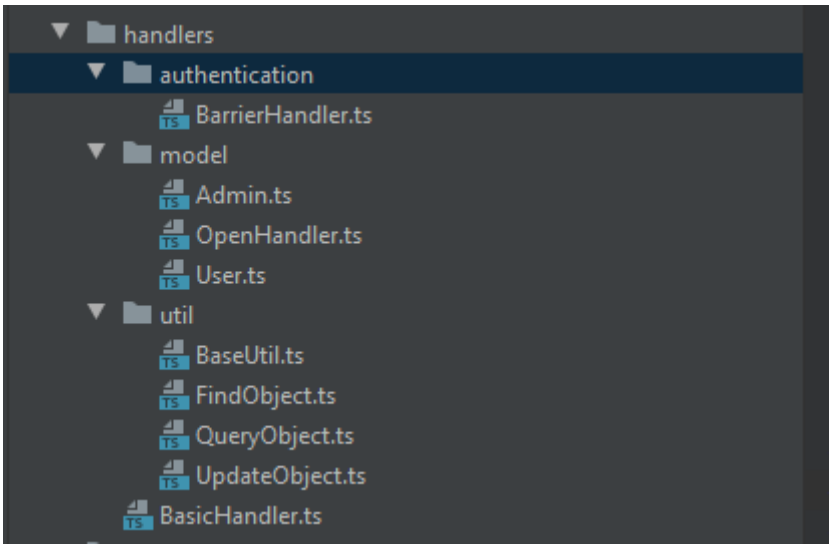

Figura 12 - Estrutura de manipulação de dados

- handlers: Além de todas as especificidades dessa estrutura, temos os *handlers*, nessa pasta contém toda a lógica de negócio da aplicação, essa parte da aplicação é baseada no *BLoC Pattern*, ela foi criada para que as classes responsáveis por receber as requisições através do acesso pela API ou eventos através do *Websocket* não tenham nenhuma lógica de negócio e apenas tratem os dados enviados através dessas interfaces e enviem direto para o *handler* responsável, isso possibilita o trabalho com qualquer uma das interfaces e uma melhor manutenção do software.

- authentication e BarrierHandler.ts: Nessa pasta contém o arquivo *BarrierHandler.ts*, esse arquivo importado pela classe *authentication/Barrier.ts*, ele é utilizado sempre que algum usuário tenta acessar uma rota da API que não é aberta, assim esse documento solicita ao documento *Session/AccessController.ts*, para verificar se o usuário está realmente logado e se a chave de sessão dele ainda está válida
- model: Essa pasta contém todas as classes responsáveis por tratar requisições de clientes, ela é dividida em restrições de usuário e *Endpoints*, podendo conter documentos para tratar

requisições de administradores, usuário comuns, *companies* e rotas abertas no caso do *OpenHandler.ts*.

- util: Contém classes utilizadas pelos *handlers*, são classes que padronizam dados a serem inseridos dentro das *MessageData.ts*.
- BasicHandler.ts: Documento herdado por todos os *handlers*, contém algumas funções essenciais para tratamento de dados e comunicação com outras classes.
- interfaces: São interfaces para serem usadas como tipos em algumas classes e funções no sistema, a necessidade dessa pasta está acabando, sendo que as interfaces estão sendo criadas no final de suas classes com uso privado.
- locale: Possibilita a internacionalização do sistema.
- resources: É nessa pasta que ficam os arquivos que podem ser consumidos pelos clientes da aplicação.

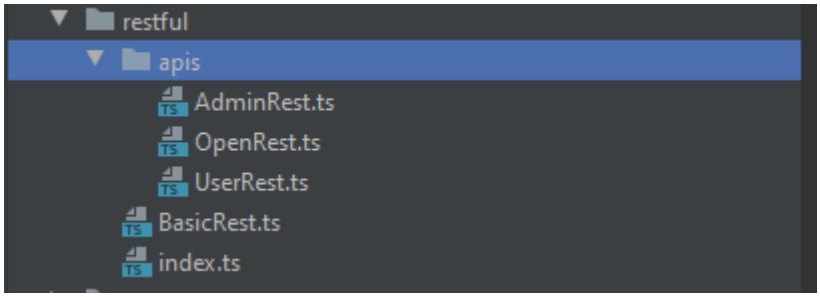

Figura 13 - Estrutura restfull do projeto

- restful: Contém todas as APIs e tratamentos de dados para serem enviados para os *handlers* responsáveis. A organização dessa pasta se baseia na estrutura dos *handlers*, cada uma das classes possui um Endpoint específico para aquela rota e seus *paths*.

- apis: Contém todos os Endpoint e *paths*, tratam os dados enviados pelos clientes e redirecionam para serem tratados para os *handlers*.
- BasicRest.ts: Todas as classes dentro de APIs devem herdar essa classe, só assim para serem reconhecidas como uma classe responsável por tratar as requisições oriundas de APIs.
- index.ts: Inicia e disponibiliza todas as APIs, essa classe é chamada pela *application*.
- rtc: Contém toda a estrutura necessária para se trabalhar com *websocket* na aplicação, não será abordado com detalhes por não ter sido utilizado na implementação.

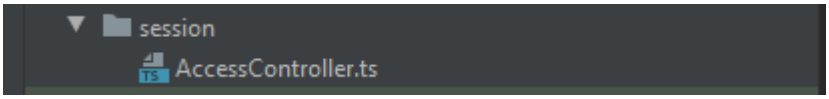

Figura 14 - Controle de acesso

- session e AccessController.ts: A *session* contém o documento *AccessController.ts*, esse documento é responsável por gerar e verificar a validade da chave de acesso do cliente. É importante ressaltar que não está sendo utilizado nenhum tipo de criptografia aqui, porém com a estrutura fornecida pode-se facilmente adicionar essa característica.
- test: Aqui ficam todos os teste unitários das APIs e *Websockets*, os testes são feitos utilizando a biblioteca mocha.
- util: Possui documentos com validadores, *enuns*, chaves para testar *https* em servidor local, entre outras utilidades.
- Application.ts: Inicia todas as classes criadas ao longo do desenvolvimento, também inicia o *Express.js* e disponibiliza, através deste, a porta de acesso aos serviços.

- config.json e devConfig.json: Possui algumas configurações como, porta *HTTP*, porta *HTTPS*, nome do bando de teste e do banco de produção entre outras configurações.
- index.ts: Essa classe verifica se o sistema está rodando em modo de desenvolvimento ou em modo de produção e inicia a classe *Application.ts* passando os parâmetros adequados.
- package.json: Possui informações sobre a aplicação, versão atual, alguns scripts *CLI*, bibliotecas que serão usadas em produção e também bibliotecas que serão utilizadas apenas em desenvolvimento, além das versões de cada uma delas.
- tsconfig.json: Configuração para fazer a transcrição de *TypeScript* para *JavaScript*, são passadas regras como versão do *JavaScript* a ser gerado, espaçamento, se possuirá *map* ou não (importante para fazer o *debug*), local onde será armazenado o *JavaScript* gerado, entre outras.

5.4 Modelagem dos dados

Uma modelagem dos dados é aqui apresentada como mostra a Figura 15.

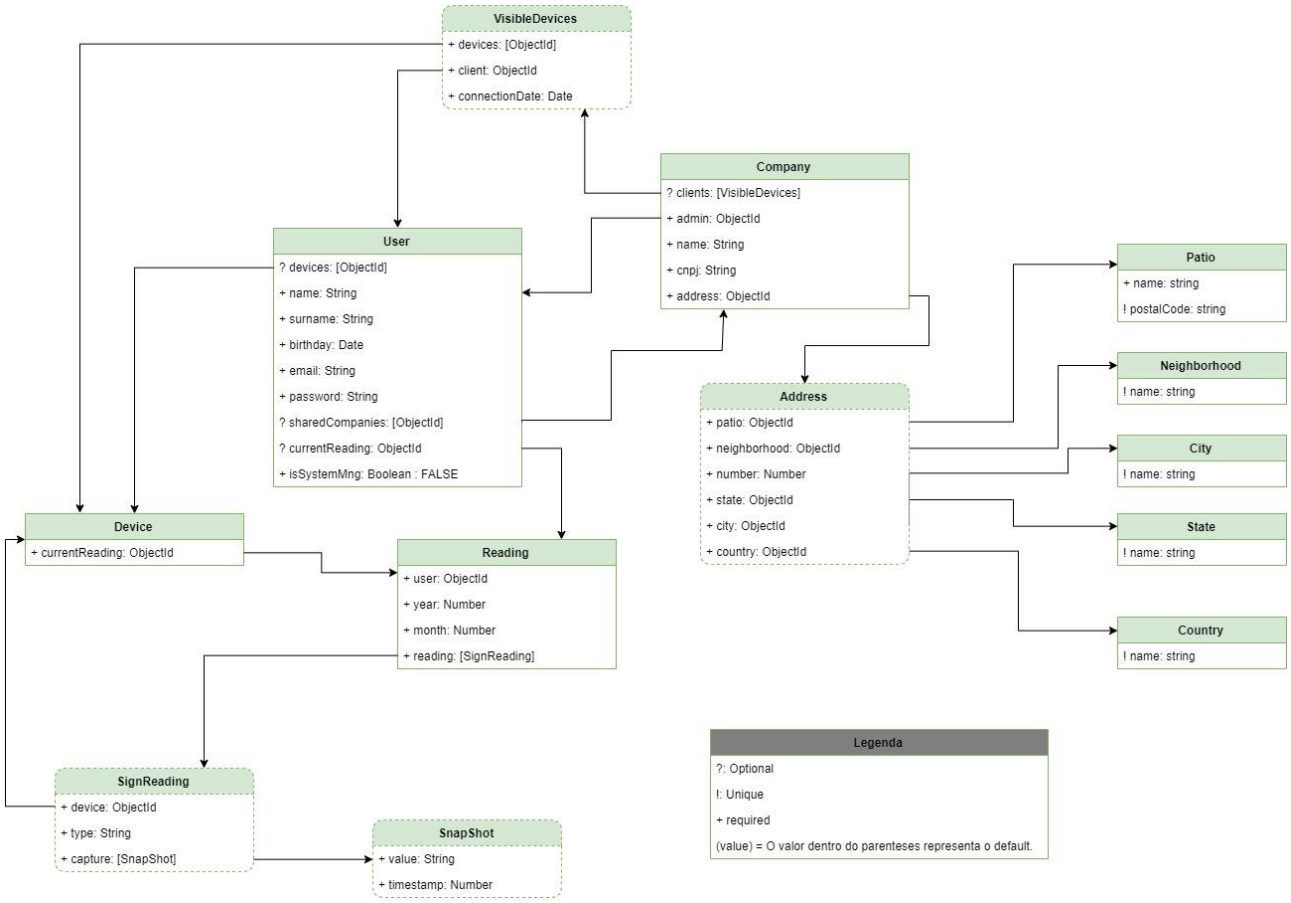

Figura 15 - Modelagem dos dados para armazenamento no banco de dados

Esse modelo de dados foi criado pelo LabTIC/UDESC para a representação da estrutura dos dados a serem armazenadas no banco de dados não relacional *MongoDB*. Nesse modelo as coleções são representadas em retângulos com linhas continuas e os *subSchemas* são representados com retângulos de bordas arredondadas e linhas tracejadas. Cada chave dentro de uma coleção ou *subSchema* vem acompanhada com alguma restrição (descritas na legenda do modelo), o nome da chave e o seu tipo, como mostrado na Figura 16.

Figura 16 - Modelagem de esquema e sub-esquema

O exemplo acima representa que a chave denominada *currentReading* é *required* e o seu valor é do tipo *ObjectId*.

As setas representam que uma coleção ou *subSchema* está contido em outro, onde a ponta da seta informa quem é o elemento contido. Os colchetes representam que o lvalor daquela chave é um Array.

5.5 Desenvolvimento do Aplicativo

A criação do aplicativo foi feita com o framework *Flutter*. A escolha se deu devido a facilidade na concepção de interfaces nativas de alta qualidade no iOS e no Android de forma ágil. O *Flutter* trabalha com código existente, é usado por desenvolvedores e organizações em todo o mundo, é gratuito e de código aberto.

5.5.1 Arquitetura

A arquitetura base usada foi fornecida pelo próprio framework, e todo o código gerado para o desenvolvimento do aplicativo é armazenado na pasta lib. Fica a critério do desenvolvedor organizar os componentes de telas. A ***Figura 17*** mostra como estão organizados os componentes de tela adotados nesse trabalho.

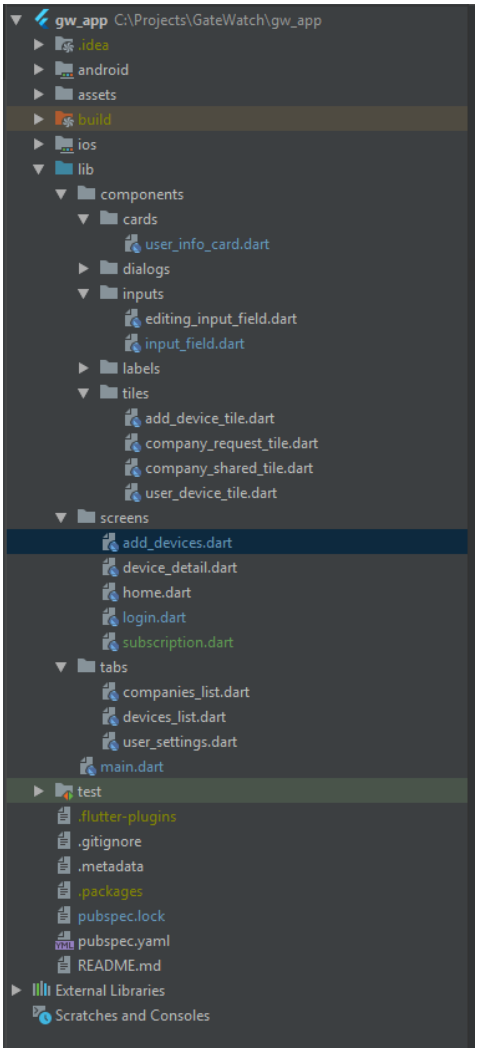

Figura 17 - Base de arquivos do aplicativo

5.5.2 BLoC

As interfaces do aplicativo foram separadas da lógica usando o padrão de projeto BLoC (Business Logic Component). Esse padrão permite que o desenvolvimento da lógica fique separado do desenvolvimento da interface, possibilitando uma fácil manutenção dessas partes, sem interferência na outra.

No Flutter, o *BLoC* funciona através de eventos permitindo que uma classe lógica seja usada por vários widgets, possibilitando também o gerenciamento de estado. Seguindo o padrão de comunicação *Stream*, eventos são disparados por widgets para serem consumidos por outros *widgets*.

5.5.3 Interface do aplicativo

Na Figura 18 são mostradas as imagens e descrições das telas implementadas.

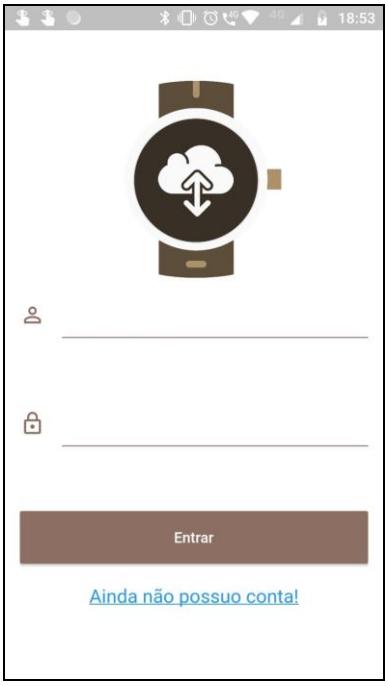

Figura 18 - Tela de login do aplicativo

No primeiro acesso ao aplicativo será apresentada a tela de autenticação, nesta o usuário poderá se autenticar, caso já tenha uma conta criada, ou optar por criar uma conta, como mostrado na *Figura 19*.

Figura 19 - Autenticação do usuário e criação de uma conta do usuário.

Caso o usuário queira criar uma conta no sistema, ele precisará preencher o formulário da *Figura 19*. Assim que a conta for criada, o usuário será autenticado e redirecionado para a tela principal.

Figura 20 - Controle de acesso de empresas

Essa é a página inicial do aplicativo, mostrada na Figura 21. Nela é mostrada todas as empresas que tem acesso aos dados gerados pelos dispositivos vestíveis do usuário. O usuário tem controle para bloquear a empresa, impossibilitando-a de acessar os seus dados.

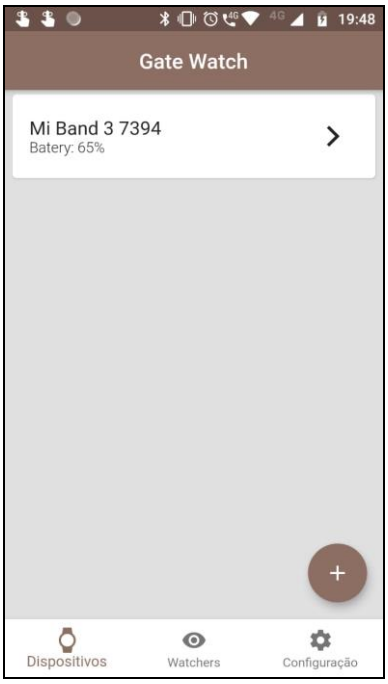

Figura 21 - Informações de cadastro e bateria

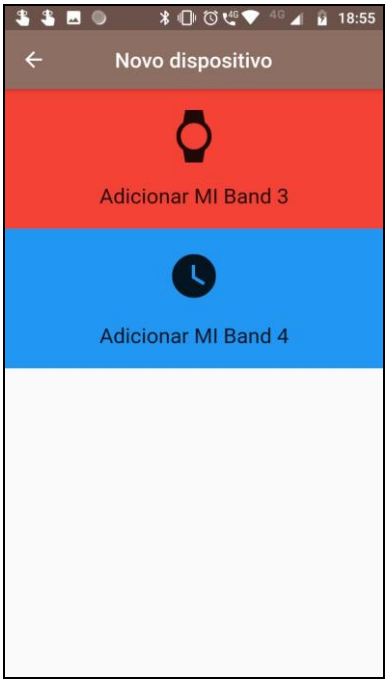

Figura 22 - Inclusão de novos relógios e pulseiras

Essas duas interfaces mostram os dispositivos vestíveis cadastrados e os modelos aceitos. O usuário tem a possibilidade de alterar/atualizar seus dados na interface de configuração.

5.6 Interfaces de gerenciamento

A interface do sistema foi desenvolvida com *JavaScript* utilizando o framework *Vue.JS*. Dois perfis são aptos a utilizarem a parte web do sistema, o perfil administrador e o perfil terceiros.

Inicialmente para a concepção da interface de interação foram criados protótipos de alta fidelidade, como mostrado na Figura 23. Esses protótipos foram utilizados para se ter uma ideia conceitual da interação dos usuários nesse sistema, dessa forma não nos comprometemos em criar exatamente o que está demonstrado nele.

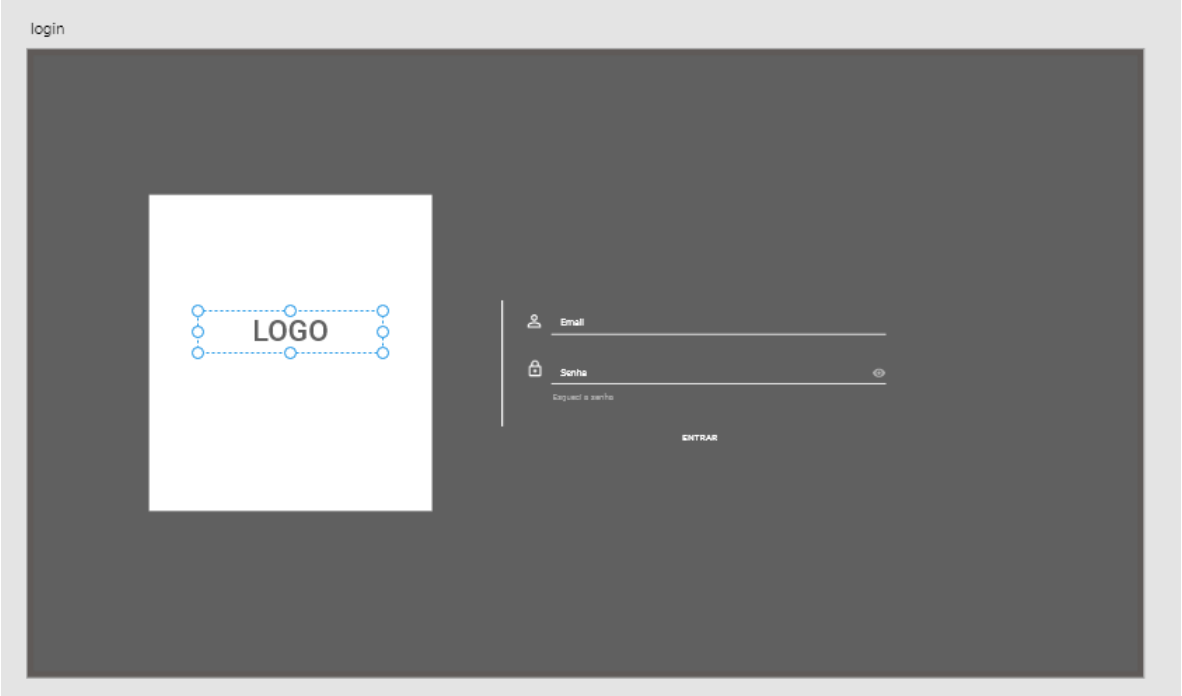

Figura 24 - Protótipo do login

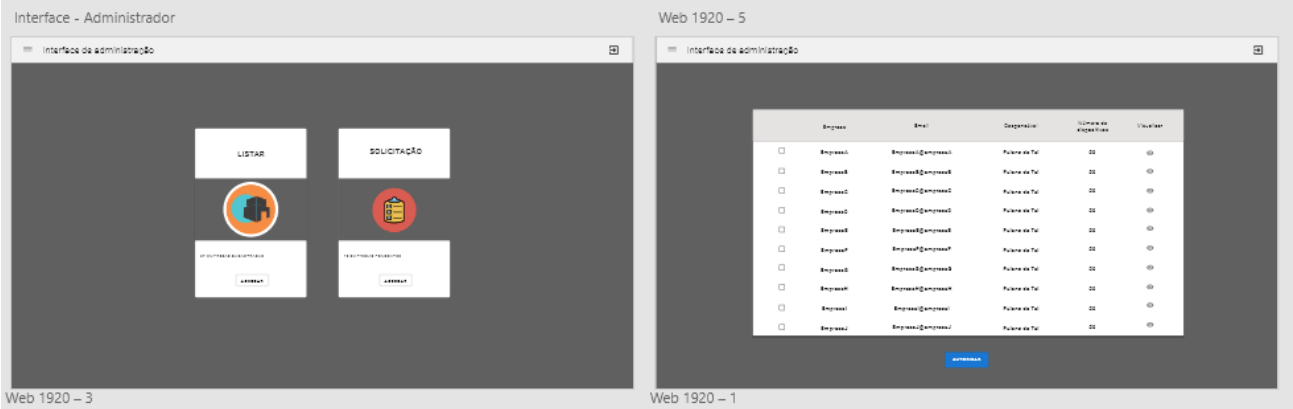

Figura 23 - Protótipo das telas de cadastro

Os protótipos foram criados através do software Adobe XD, em sua versão gratuita. Através do uso desse software de prototipação, além de ter noção de como ficaria interface gráfica do sistema, também pudemos simular as interações ocorridas entre elas.

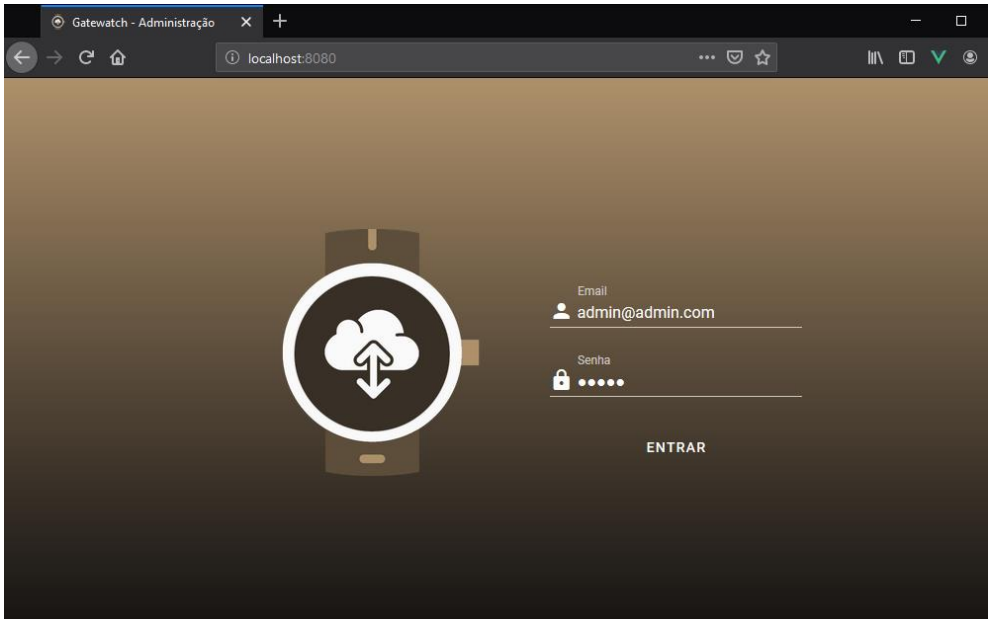

Figura 25 - Tela de login desenvolvida

A autenticação será efetuada, tanto por Administrador quanto por Empresa, através da mesma interface, sendo direcionado subsequentemente para as devidas interfaces através de rotas.

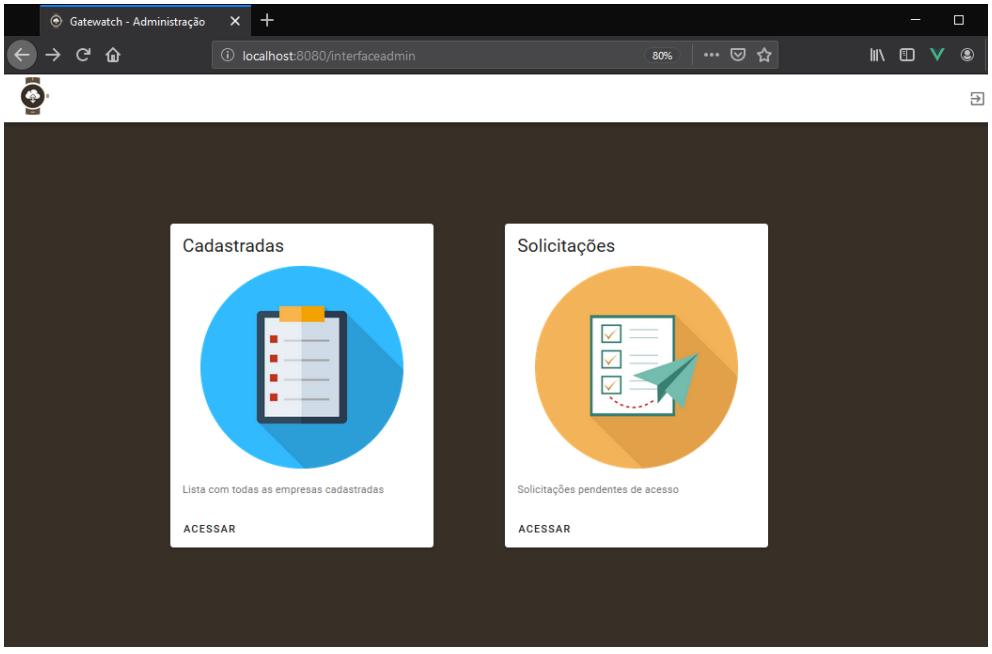

Figura 26 - Tela inicial do administrador

Ao acessar a parte *“Cadastradas”,* o Administrador terá acesso à lista completa com as principais informações das empresas que atualmente tem acesso ao sistema. Poderá ainda realizar alterações no cadastro e realizar novos cadastros de empresas.

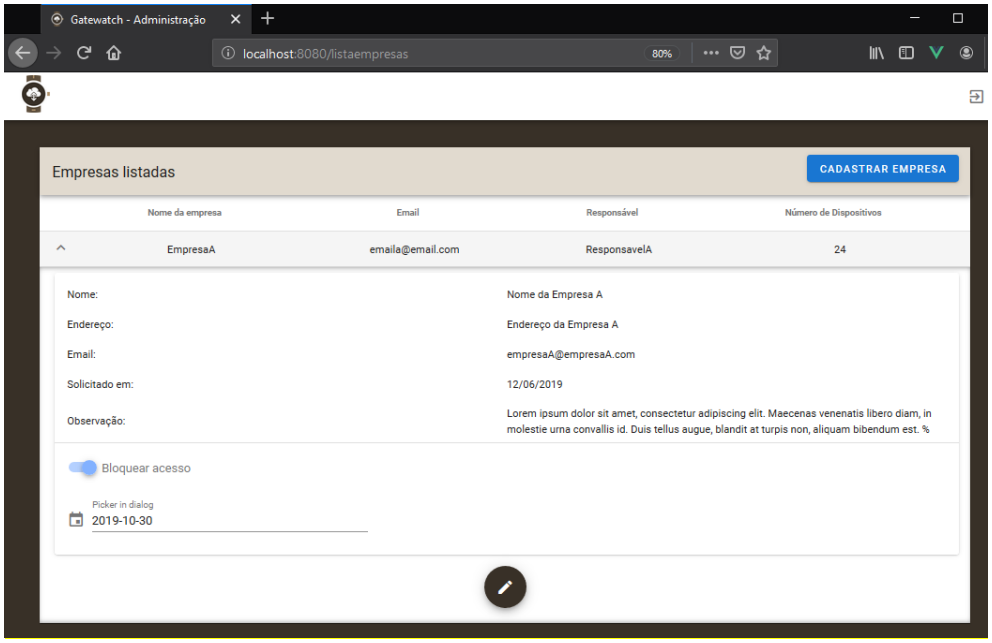

Figura 27 - Interface de cadastro de empresa

O Administrador é responsável por gerenciar usuários e empresas cadastradas, além de autorizar a utilização da plataforma analisando as solicitações de uso das empresas.

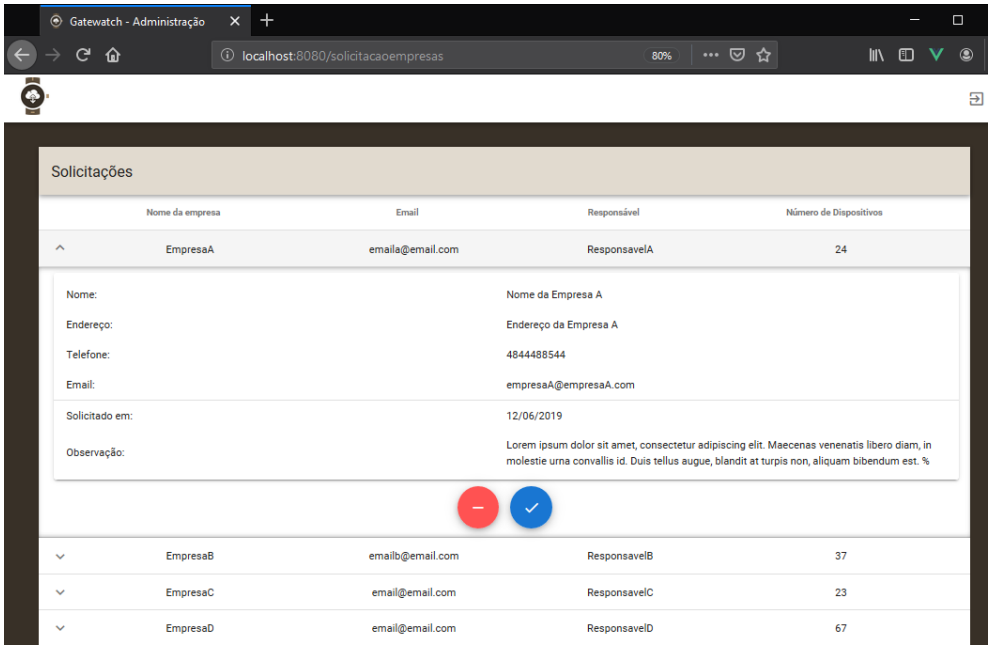

Figura 28 - Interface de administração das empresas

Já na parte de solicitações, haverá a lista com todos pedidos de acesso vindos da empresa interessadas nas informações de alguns dos dispositivos cadastrados na plataforma.

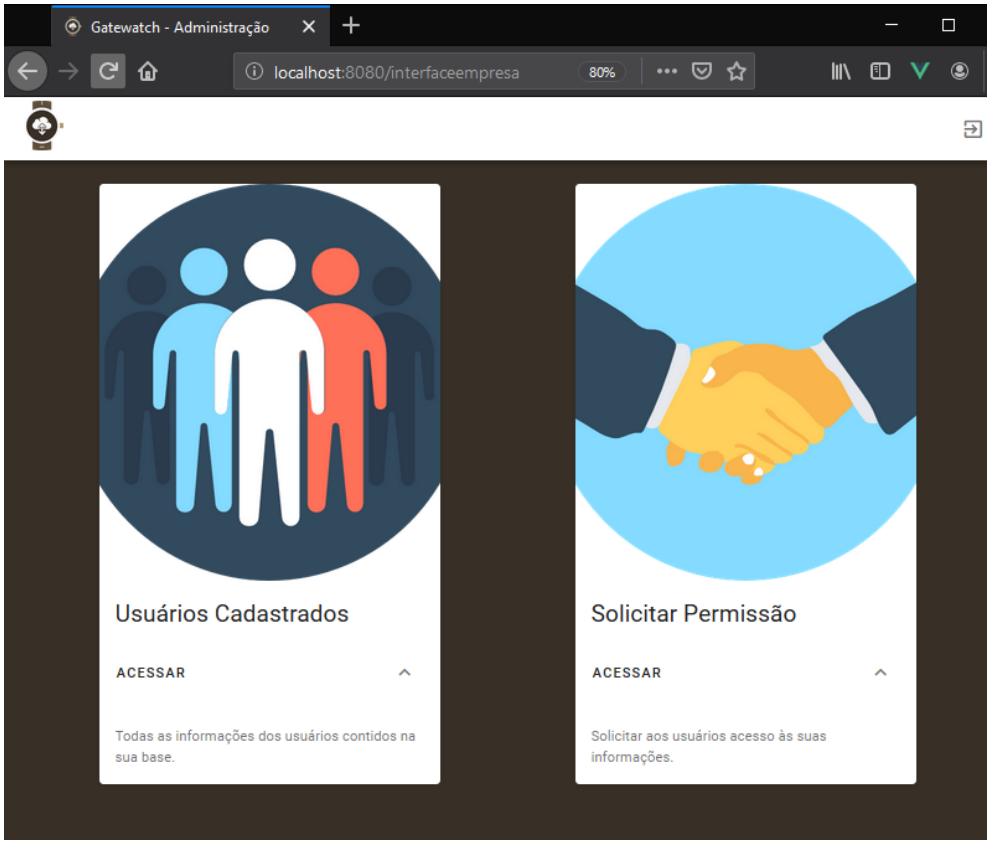

Figura 29 - Interface de administração de empresa

O perfil Empresa poderá gerenciar seus usuários, sem poder alterar ou atualizar informações de perfil destes. E também solicitar acesso a informações de outros usuários já cadastrados na plataforma.

6 CONCLUSÃO

Conseguimos executar parte da proposta inicialmente definida, que seria a criação de um modelo de captura e coleta de dados de relógios e pulseiras inteligentes que visa facilitar o acesso de empresas interessadas em utilizar tais dados. Inicialmente o modelo serviria para inúmeros modelos de dispositivos, porém durante a execução desse projeto acabou sendo executado apenas para um deles, ou seja, como trabalho futuro esse seria um ponto a ser melhorado.

Verificamos que o modelo de uma construção genérica de sistema para captura de dados proposto é viável, porém demanda tempo e um esforço maior que o empregado até o momento. As limitações mais notáveis durante a execução do projeto foram relacionadas às capturas de dados transmitidos entre os relógios e pulseiras e os smartphones, pois a decodificação e entendimento dos dados levaram muito tempo. Por fim, definimos como as partes mais interessantes a serem exploradas futuramente no capítulo posterior.

7 TRABALHOS FUTUROS

Com base nas sugestões fornecidas aos alunos durante a pesquisa e desenvolvimento desse trabalho, levando em conta a evolução tecnológica e necessidades emergentes no decorrer de todo o projeto, os seguintes trabalhos futuros foram identificados:

- Incluir novos modelos de dispositivos
- Melhorar a autenticação de usuários
- Incluir alertas e outras funcionalidades no aplicativo

8 BIBLIOGRAFIA

AGYAPONG, P. K. et al. 5G Is Not the Answer For Rural Broadband. IEEE Communications Magazine, v. 12, n. July, p. 56, 2018.

ALMEIDA, G. et al. Internet of Things (IoT): Um Cenário Guiado por Patentes Industriais Internet of Things (IoT): A Scenario Guided by Industrial Patents.

GESTÃO.Org - Revista Eletrônica de Gestão Organizacional, v. 13, p. 271–281, 2016.

ANNAMDAS, V. G. M.; BHALLA, S.; SOH, C. K. Applications of structural health monitoring technology in Asia. Structural Health Monitoring, v. 16, n. 3, p. 324–346, 2017.

CECHINEL, A. Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web. v. 0, p. 77, 2017.

CHEN, M.; MAO, S.; LIU, Y. Big data: A survey. Mobile Networks and Applications, v. 19, n. 2, p. 171–209, 2014.

CHO, K. et al. Analysis of latency performance of bluetooth low energy (BLE) networks. Sensors (Switzerland), v. 15, n. 1, p. 59–78, 2015.

DARTLANG.DEV. Dartlang.

EMERGING, A.; TECHNOLOGY, L. W. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. p. 11734–11753, 2012.

GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, v. 29, n. 7, p. 1645–1660, 2013.

HARARI, Y. N. 21 LESSONS FOR THE 21ST CENTURY. [s.l: s.n.].

HUNG, K.; ZHANG, Y. T.; TAI, B. Wearable medical devices for tele-home healthcare. Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings, v. 26 VII, p. 5384–5387, 2004.

KATAL, A.; WAZID, M.; GOUDAR, R. H. Big data: Issues, challenges, tools and Good practices. 2013 6th International Conference on Contemporary Computing, IC3 2013, p. 404–409, 2013.

LIN, J. et al. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. IEEE Internet of Things Journal, v. 4, n. 5, p. 1125–1142, 2017.

MOLLOV, M. BEN. Smart Wearables for Remote Health Monitoring, From Prevention To Rehabilitation: Current R&D, Future Challenges. Proc of the 4th Annual IEEE Conf on Information Technology Applications in Biomedicine. UK, v. 7, n. 1, p. 25–4888, 2000.

NODEBR.COM. NODEBR.

QADEER, M. A. et al. Network traffic analysis and intrusion detection using packet sniffer. 2nd International Conference on Communication Software and Networks, ICCSN 2010, p. 313–317, 2010.

REEDER, B.; DAVID, A. Health at hand: A systematic review of smart watch uses for health and wellness. Journal of Biomedical Informatics, v. 63, p. 269–276, 2016.

REHMAN, H. et al. Mobile Health (mHealth) Technology for the Management of Hypertension and Hyperlipidemia: Slow Start but Loads of Potential. Current Atherosclerosis Reports, v. 19, n. 3, 2017.

ROLIM, M. P. Saúde Monitorada: O Uso de Vestíveis para o Acompanhamento à Distância de Sinais do Usuário, 2016.

SANTOS, B. P. et al. Internet das Coisas: da Teoria à Prática. Minicursos SBRC-Simp, p. 50, 2016.

STRONGLOOP. ExpressJS.

TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, v. 14, n. 6, p. 80–83, 2010.

WIKTORSKI, T. NOSQL databases. Advanced Information and Knowledge Processing, p. 77–84, 2019.

WISE, A. Interactive learning aided by JavaScript. Research in Learning Technology, v. 7, n. 2, 2011.

XIA, F.; LAURENCE, Y. Internet of Things. International Journal of Communication Systems, v. 23, n. 5, p. 633–652, 2010.

YOON, H.; PARK, S. H.; LEE, K. T. Lightful user interaction on smart wearables. Personal and Ubiquitous Computing, v. 20, n. 6, p. 973–984, 2016.

YOU, E. VUEJS.

ZAKAS, N. Professional JavaScript for Web Developers. [s.l: s.n.]. v. 3ª edição

ZHAO, G. et al. Modeling MongoDB with relational model. Proceedings - 4th

International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2013, p. 115–121, 2013.

NAVATHE&ELMASRI. Fundamentals of Database-Systems 7th Edition. 7th. ed. Arlington: Person, 2017.

ANEXO A – Artigo

Gatewatch, um centralizador de dados capturados a partir de pulseiras e relógios inteligentes

Osvaldo Miguel Junior, Robson de Carvalho

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) - Florianópolis – SC – Brasil

juniordepalmas@ufsc.br, robson.carvalho@ufsc.br

Abstract. *The constant growth of the use of wearable devices has produced a non-standard environment in the collection and consumption of data originating from these devices, especially because each device comes with its own solution, provided by manufacturers. Smartwatch and smartband are, among wearable devices, the most popular and most used, with increasing annual sales by various brands that produce them. They constantly follow a data exchange stream that goes from sending data to smartphones to being sent and processed on servers. But this exchange is not as accessible as each smartwatch and smartband has its own data model and applications to perform these data exchanges. In this work we develop a data capture and storage service that is independent of the device used, in order to facilitate access to information for corporate software. Through an application provided to the end user we capture this information by sending it to a server, storing it and making it available to authorized companies.*

Resumo. *Com o constante crescimento da utilização de dispositivos vestíveis produziu-se um ambiente com falta de padrão na coleta e consumo dos dados originários desses equipamentos, sobretudo pelo fato de que cada dispositivo vem com uma solução própria, fornecida pelas fabricantes. Os relógios e pulseiras são, dentre os dispositivos vestíveis, os mais populares e os mais utilizados, com crescente número de vendas anual por diversas marcas que os produzem. Constantemente seguem um fluxo de troca de dados que passa por enviar dados para smartphones para, em seguida, serem enviados e processados em servidores. Porém essa troca não se apresenta de forma tão acessível já que cada relógio ou pulseira possui um modelo de dados e aplicativos próprios para realizar essas trocas dados. Nesse trabalho desenvolvemos um serviço de captura e armazenamento de dados que independe do dispositivo utilizado, de forma a facilitar o acesso a informação para softwares de empresas. Através de um aplicativo fornecido para o usuário final capturamos essas informações enviando-as para um servidor armazená-las e disponibilizá-las para empresas autorizadas.*

1.INTRODUÇÃO

O crescente o uso de dispositivos vestíveis inteligentes (wearables) tem chamado a atenção para os volumosos números que poderão ser atingidos em poucos

anos, algo em torno de 25 bilhões de dólares em 2019 (YOON; PARK; LEE, 2016). Há um enorme potencial de aplicação para o uso de dispositivos vestíveis, sendo o maior deles o uso para medições e monitoramento da saúde (REEDER; DAVID, 2016). Dentre os dispositivos vestíveis os mais difundidos são os relógios e pulseiras inteligentes, que devido aos avanços tanto na qualidade das baterias, quanto na qualidade de suas interfaces, ou ainda sua integração com outros dispositivos conectados à Internet, estimularam uma popularização notável na última década (YOON; PARK; LEE, 2016). Além disso, sua integração para realizar pagamentos através da tecnologia Near Fiel Communication (NFC), tem ajudado nesse crescimento.

Os autores, ao prospectar essa tecnologia, perceberam a ausência de padrões para captura e coleta de dados, sobretudo com o surgimento de novas marcas, que substancialmente vem aumentando ainda mais a diversificação de dispositivos existentes no mercado. Levando-se em conta as perspectivas futuras do desenvolvimento da rede 5G (AGYAPONG et al., 2018), o avanço da Internet das Coisas (GUBBI et al., 2013) e a quantidade gigantesca de dados que precisam ser processados e armazenados na era do Big Data (CHEN; MAO; LIU, 2014), a escassez de um padrão para se trabalhar com os dados originários de dispositivos vestíveis, tanto por parte do usuário final, quanto por parte dos desenvolvedores de novas soluções para esses equipamentos, pode causar dificuldade na evolução dessa tecnologia.

Essa falta de padrão do lado do usuário final pode causar problemas, por exemplo, em situações de importação e exportação de dados (CHEN; MAO; LIU, 2014). Cada dispositivo vestível possui um aplicativo para smartphone, dessa forma caso o usuário queira trocar de dispositivo ele, muito provavelmente, deverá fazer toda a migração dos dados para utilizar um novo aplicativo. Já da parte dos desenvolvedores, essa excessiva quantidade de aplicativos gera a falta de um padrão para se trabalhar com os dados, o que com o passar do tempo, com o aumento do número desses dispositivos, pode acabar dificultando o desenvolvimento de soluções.

2. PROPOSTA

Desenvolver um sistema Web, composto por um aplicativo de celular que capture sinais de um relógio ou pulseira inteligente, independente de sua marca e modelo, armazena os dados capturados em um servidor, sobre o qual terceiros possam

se cadastrar através de uma interface Web, e com a autorização do usuário, baixar os dados por meio de uma API.

3. IMPLEMENTAÇÃO

Nesse trabalho fizemos a unificação do acesso aos dados obtidos através das pulseiras e relógios inteligentes em apenas uma API. Isso facilita aos interessados em utilizar esses dados, já que precisarão acessar apenas a uma única API ao invés muitas outras. A Figura 3 mostra o modelo atual de acesso aos dados (com inúmeras APIs) onde a obtenção de dados é feita de forma exclusiva através de uma API disponibilizada pela fabricante do dispositivo. Já a Figura 4 mostra a proposta executada nesse trabalho (uma única API), que é a centralização dos dados provenientes dos dispositivos.

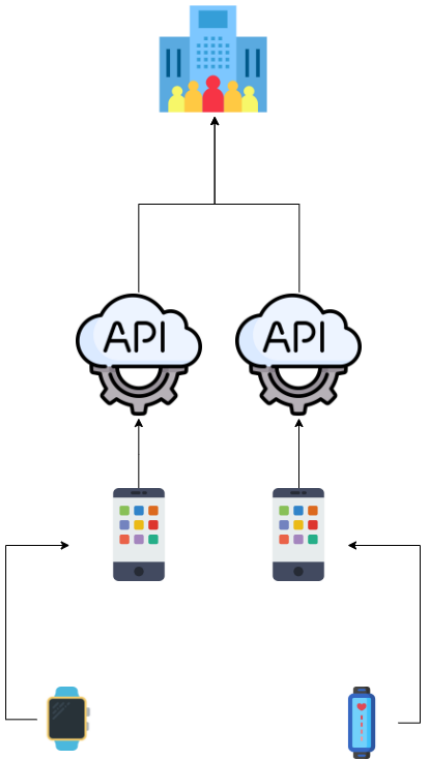

Figura 30 - Modelo atual de funcionamento da captura de dados

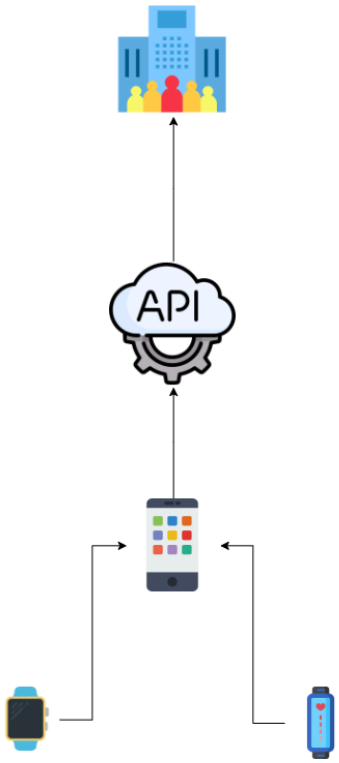

Figura 31 - Modelo proposto de funcionamento da captura de dados

Para realizarmos o trabalho criamos uma solução que, essencialmente, é um grande servidor onde esse recebe informação dos usuários do nosso sistema. O sistema se

divide em um aplicativo para smartphone (IOS e Android) e um sistema web, onde serão as entradas dos dados desse sistema. Após essa entrada, nosso servidor fará o processamento dos dados.

Os usuários do sistema foram divididos em três grupos, cada um tendo um perfil de acesso ao sistema. O primeiro grupo será chamado de Usuário, o segundo grupo Administrador e o terceiro grupo são os Empresa. Cada um desses grupos terá acesso a algumas partes específicas do sistema. A Figura 5 mostra quais partes do sistema será liberado para cada um dos perfis. O Usuário representa todos os utilizadores finais dos dispositivos, pulseira ou relógios inteligentes, os quais terão acesso aos seus dados capturados por meio do aplicativo. O Administrador será alguém vinculado ao sistema que será o intermediador de todo o acesso. Já Empresa são os interessados externos que terão acesso aos dados dos Usuários para promover ou criar soluções.

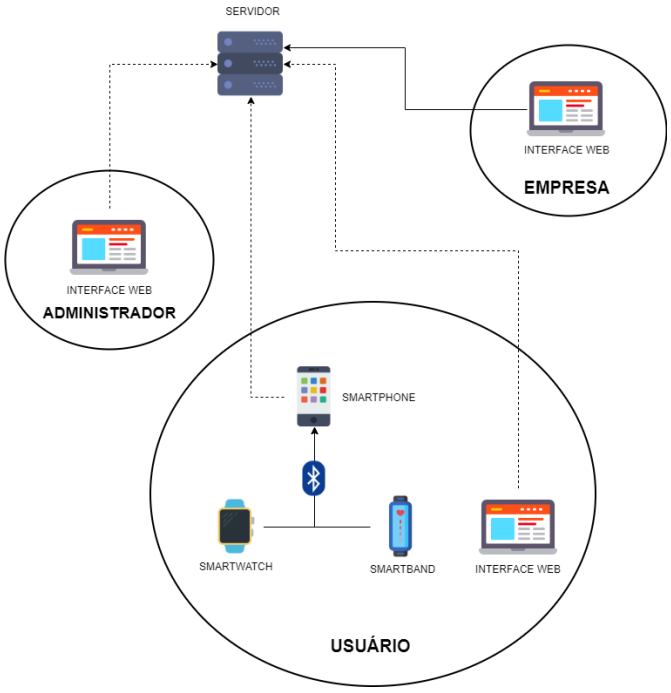

Figura 32 - Arquitetura básica do sistema

4. DISPOSTIVOS UTILIZADOS

Para a execução dos testes propostos utilizamos dois dispositivos de mesma marca. Os dispositivos utilizados foram o Xiaomi Amazfit e o Xiaomi Miband 3, sendo utilizados sobretudo pela facilidade de acesso da equipe aos mesmos. Foram iniciados testes com outros dispositivos, porém devido ao escopo do nosso projeto acabamos não incluindo nesse trabalho. Utilizamos também um sniffer para realizar a captura dos dados transmitidos entre os dispositivos vestíveis e os smartphones.

Especificações técnicas dos dispositivos:

Xiaomi Amazfit Verge:

Xiaomi Mi band 3

Nordic BLE Sniffer

5. MODELAGEM DOS DADOS

Uma modelagem dos dados é aqui apresentada como mostra a Figura 15.

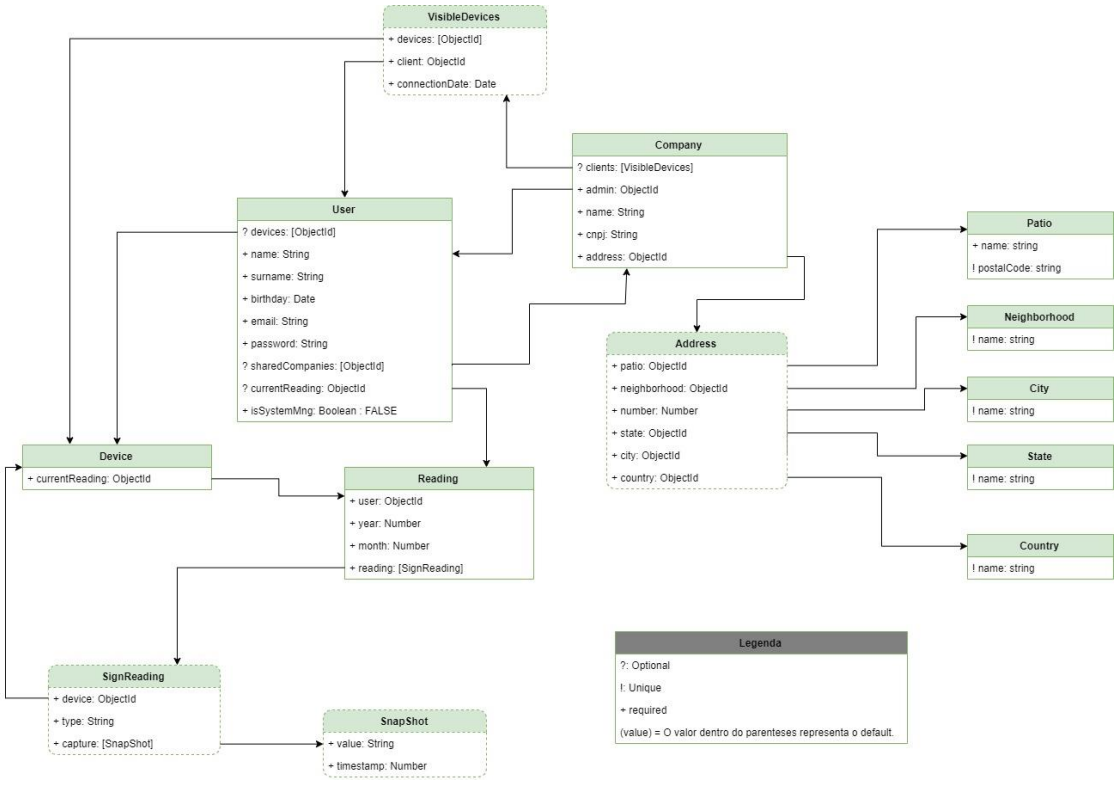

Figura 33 - Modelagem dos dados para armazenamento no banco de dados

Esse modelo de dados foi criado pelo LabTIC/UDESC para a representação da estrutura dos dados a serem armazenadas no banco de dados não relacional MongoDB. Nesse modelo as coleções são representadas em retângulos com linhas continuas e os subSchemas são representados com retângulos de bordas arredondadas e linhas tracejadas. Cada chave dentro de uma coleção ou subSchema vem acompanhada com alguma restrição (descritas na legenda do modelo), o nome da chave e o seu tipo, como mostrado na Figura 16.

Figura 34 - Modelagem de esquema e sub-esquema

O exemplo acima representa que a chave denominada currentReading é required e o seu valor é do tipo ObjectId. As setas representam que uma coleção ou subSchema está contido em outro, onde a ponta da seta informa quem é o elemento contido. Os colchetes representam que o valor daquela chave é um Array.

6. PROJETO

Na Figura 18 são mostradas as imagens e descrições das telas implementadas.

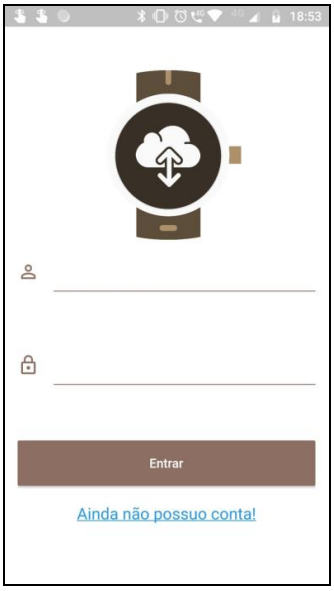

Figura 35 - Tela de login do aplicativo

No primeiro acesso ao aplicativo será apresentada a tela de autenticação, nesta o usuário poderá se autenticar, caso já tenha uma conta criada, ou optar por criar uma conta, como mostrado na ***Figura 19***.

Figura 36 - Autenticação do usuário e criação de uma conta do usuário.

Caso o usuário queira criar uma conta no sistema, ele precisará preencher o formulário da *Figura 19*. Assim que a conta for criada, o usuário será autenticado e redirecionado para a tela principal.

Figura 37 - Controle de acesso de empresas

Essa é a página inicial do aplicativo, mostrada na Figura 21. Nela é mostrada todas as empresas que tem acesso aos dados gerados pelos dispositivos vestíveis do usuário. O usuário tem controle para bloquear a empresa, impossibilitando-a de acessar os seus dados.

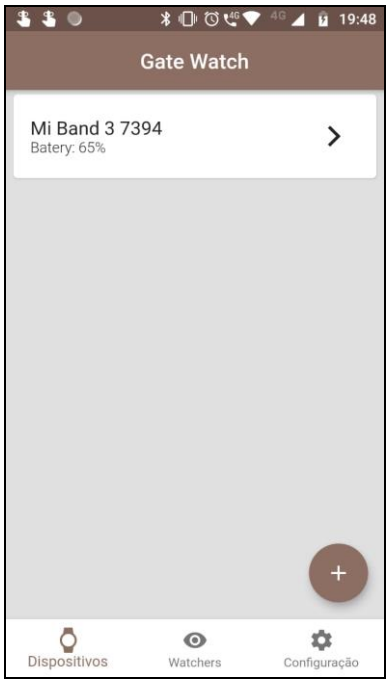

Figura 38 - Informações de cadastro e bateria

Figura 39 - Inclusão de novos relógios e pulseiras

Essas duas interfaces mostram os dispositivos vestíveis cadastrados e os modelos aceitos. O usuário tem a possibilidade de alterar/atualizar seus dados na interface de configuração.

Interfaces de gerenciamento

A interface do sistema foi desenvolvida com *JavaScript* utilizando o framework *Vue.JS*. Dois perfis são aptos a utilizarem a parte web do sistema, o perfil administrador e o perfil terceiros. Inicialmente para a concepção da interface de interação foram criados protótipos de alta fidelidade, como mostrado na Figura 23. Esses protótipos foram utilizados para se ter uma ideia conceitual da interação dos

usuários nesse sistema, dessa forma não nos comprometemos em criar exatamente o que está demonstrado nele.

Os protótipos foram criados através do software Adobe XD, em sua versão gratuita. Através do uso desse software de prototipação, além de ter noção de como ficaria interface gráfica do sistema, também pudemos simular as interações ocorridas entre elas.

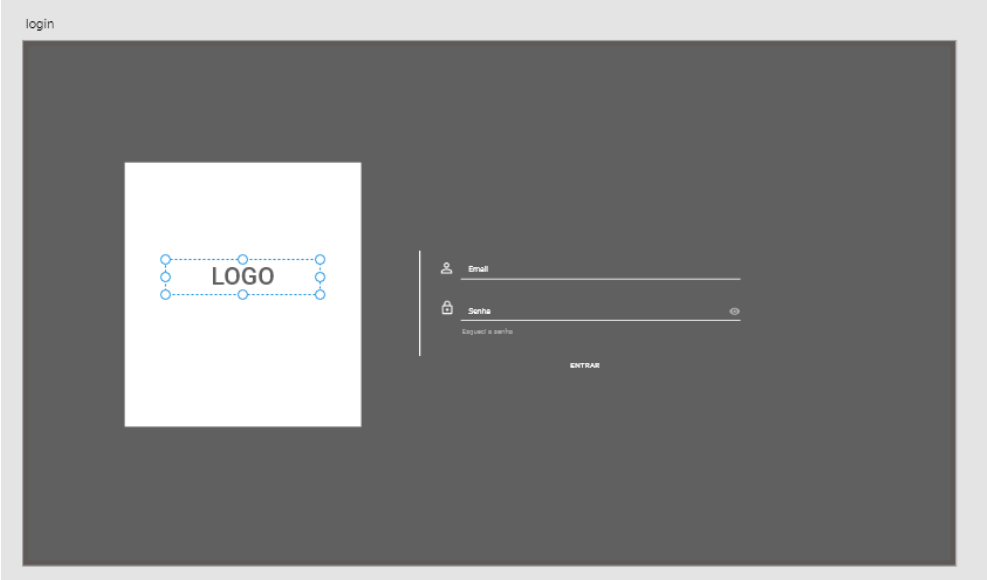

Figura 40 - Protótipo do login

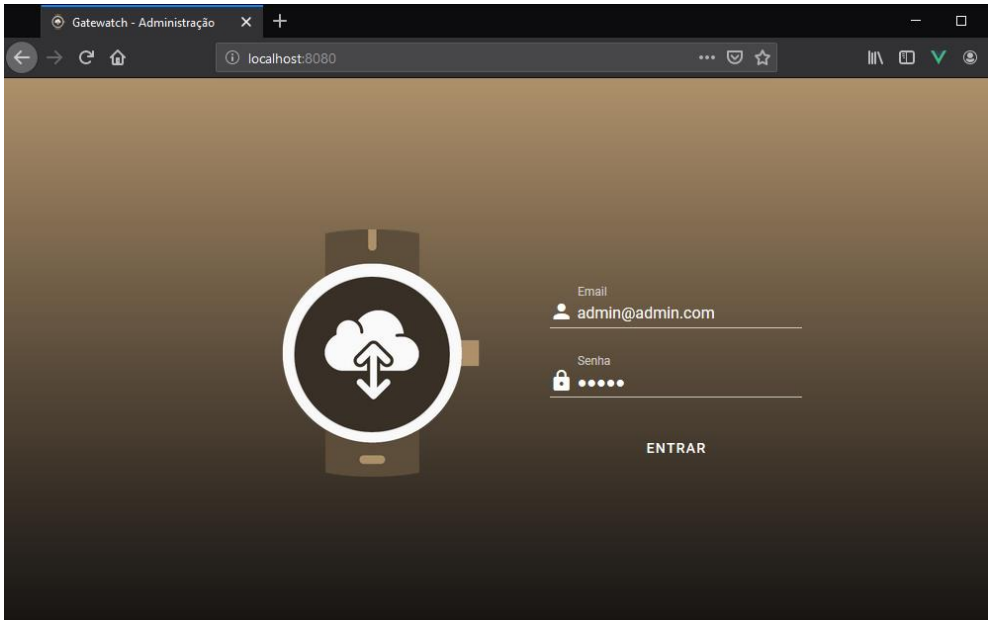

Figura 41 - Tela de login desenvolvida

A autenticação será efetuada, tanto por Administrador quanto por Empresa, através da mesma interface, sendo direcionado subsequentemente para as devidas interfaces através de rotas.

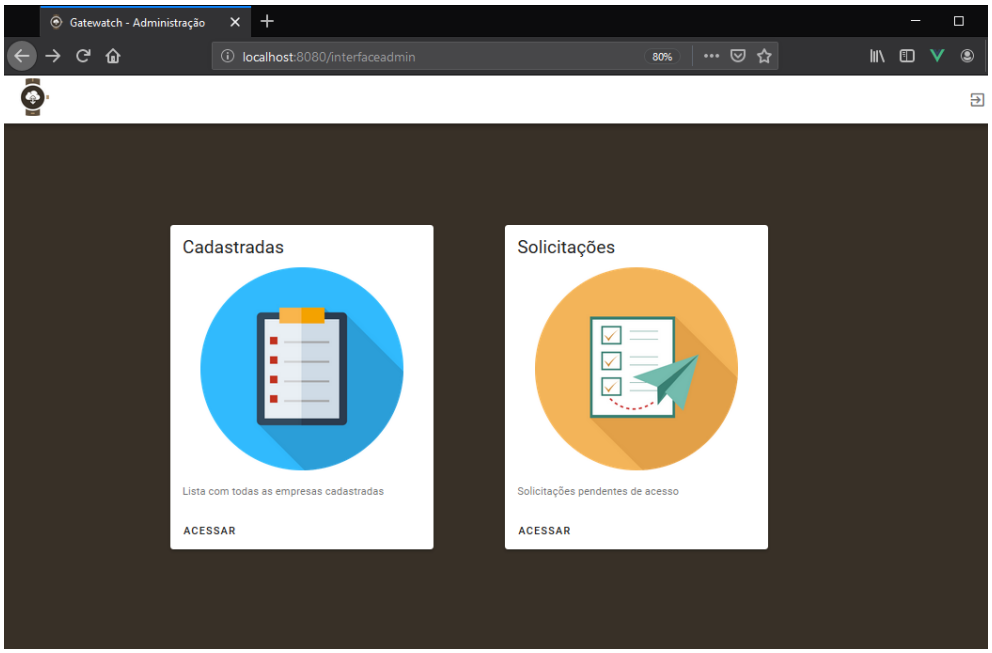

Figura 42 - Tela inicial do administrador

Ao acessar a parte *"Cadastradas"*, o Administrador terá acesso à lista completa com as principais informações das empresas que atualmente tem acesso ao sistema. Poderá ainda realizar alterações no cadastro e realizar novos cadastros de empresas.

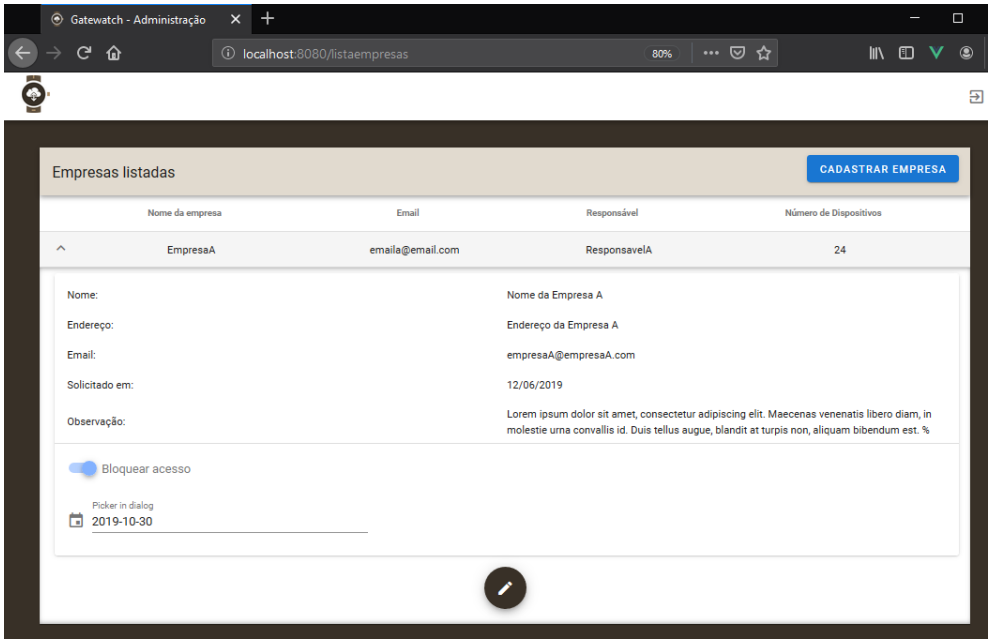

Figura 43 - Interface de cadastro de empresa

O Administrador é responsável por gerenciar usuários e empresas cadastradas, além de autorizar a utilização da plataforma analisando as solicitações de uso das empresas.

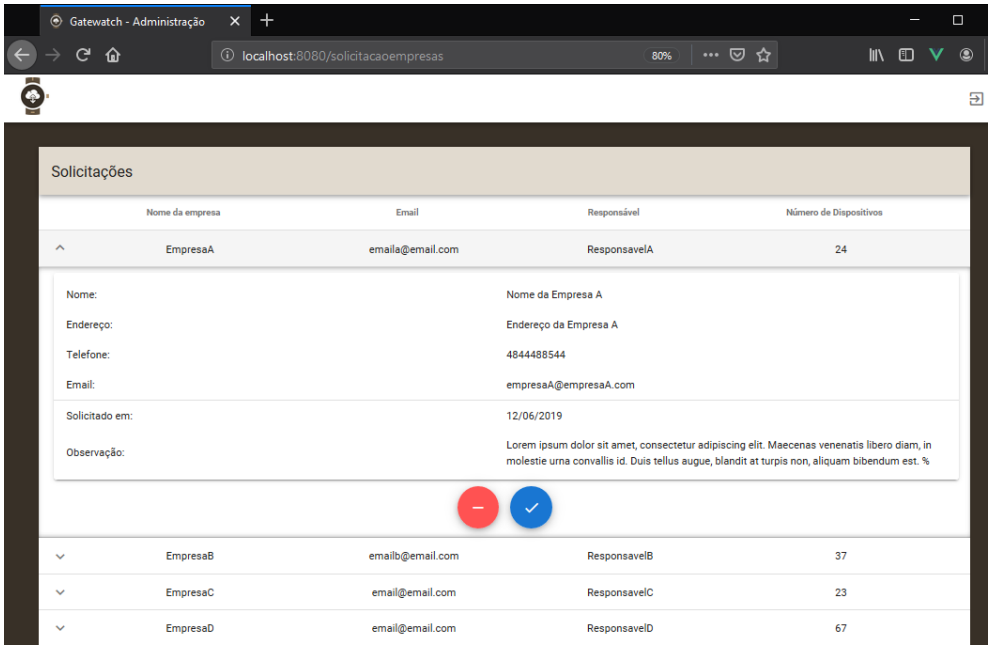

Figura 44 - Interface de administração das empresas

Já na parte de solicitações, haverá a lista com todos pedidos de acesso vindos da empresa interessadas nas informações de alguns dos dispositivos cadastrados na plataforma.

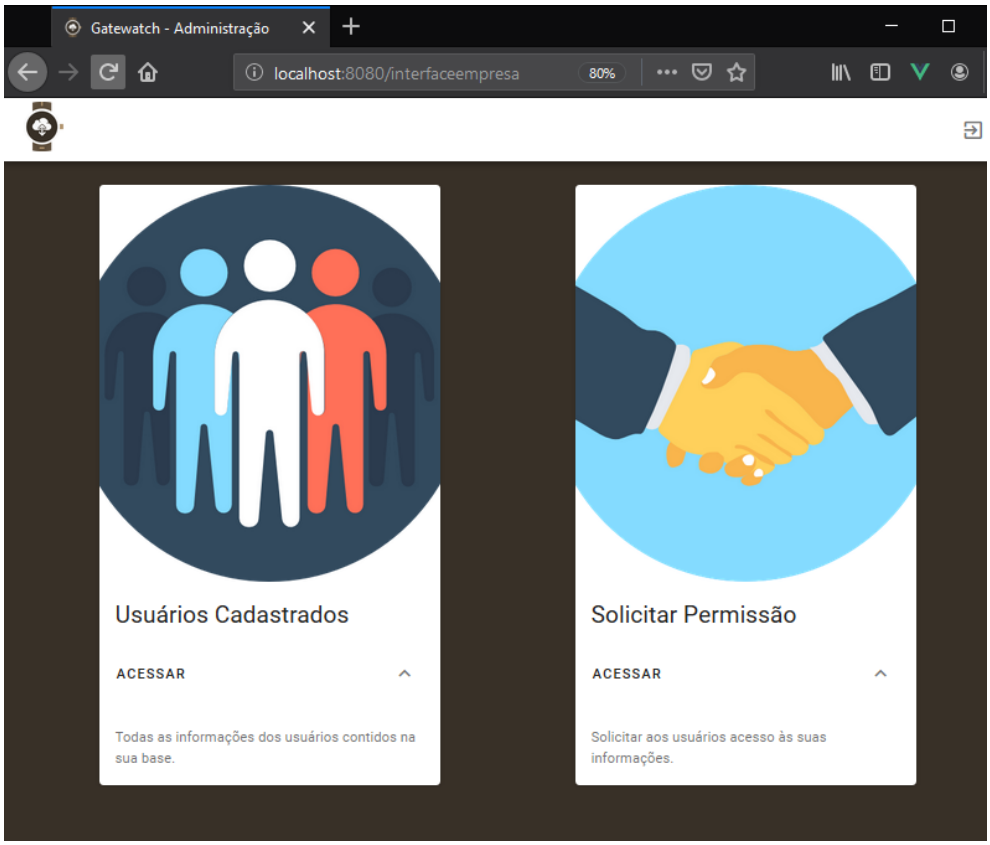

Figura 45 - Interface de administração de empresa

O perfil Empresa poderá gerenciar seus usuários, sem poder alterar ou atualizar informações de perfil destes. E também solicitar acesso a informações de outros usuários já cadastrados na plataforma.

7. CONCLUSÃO

Conseguimos executar parte da proposta inicialmente definida, que seria a criação de um modelo de captura e coleta de dados de relógios e pulseiras inteligentes que visa facilitar o acesso de empresas interessadas em utilizar tais dados. Inicialmente o modelo serviria para inúmeros modelos de dispositivos, porém durante a execução desse projeto acabou sendo executado apenas para um deles, ou seja, como trabalho futuro esse seria um ponto a ser melhorado.

Verificamos que o modelo de uma construção genérica de sistema para captura de dados proposto é viável, porém demanda tempo e um esforço maior que o empregado até o momento. As limitações mais notáveis durante a execução do projeto foram relacionadas às capturas de dados transmitidos entre os relógios e pulseiras e os smartphones, pois a decodificação e entendimento dos dados levaram muito tempo. Por fim, definimos como as partes mais interessantes a serem exploradas futuramente no capítulo posterior.

REFERENCES

AGYAPONG, P. K. et al. 5G Is Not the Answer For Rural Broadband. IEEE Communications Magazine, v. 12, n. July, p. 56, 2018.

ALMEIDA, G. et al. Internet of Things (IoT): Um Cenário Guiado por Patentes Industriais Internet of Things (IoT): A Scenario Guided by Industrial Patents.

GESTÃO.Org - Revista Eletrônica de Gestão Organizacional, v. 13, p. 271–281, 2016.

ANNAMDAS, V. G. M.; BHALLA, S.; SOH, C. K. Applications of structural health monitoring technology in Asia. Structural Health Monitoring, v. 16, n. 3, p. 324–346, 2017.

CECHINEL, A. Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web. v. 0, p. 77, 2017.

CHEN, M.; MAO, S.; LIU, Y. Big data: A survey. Mobile Networks and Applications, v. 19, n. 2, p. 171–209, 2014.

CHO, K. et al. Analysis of latency performance of bluetooth low energy (BLE) networks. Sensors (Switzerland), v. 15, n. 1, p. 59–78, 2015.

DARTLANG.DEV. Dartlang.

EMERGING, A.; TECHNOLOGY, L. W. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. p. 11734–11753, 2012.

GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, v. 29, n. 7, p. 1645–1660, 2013.

HARARI, Y. N. 21 LESSONS FOR THE 21ST CENTURY. [s.l: s.n.].

HUNG, K.; ZHANG, Y. T.; TAI, B. Wearable medical devices for tele-home healthcare. Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings, v. 26 VII, p. 5384–5387, 2004.

KATAL, A.; WAZID, M.; GOUDAR, R. H. Big data: Issues, challenges, tools and Good practices. 2013 6th International Conference on Contemporary Computing, IC3 2013, p. 404–409, 2013.

LIN, J. et al. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. IEEE Internet of Things Journal, v. 4, n. 5, p. 1125–1142, 2017.

MOLLOV, M. BEN. Smart Wearables for Remote Health Monitoring, From Prevention To Rehabilitation: Current R&D, Future Challenges. Proc of the 4th Annual IEEE Conf on Information Technology Applications in Biomedicine. UK, v. 7, n. 1, p. 25–4888, 2000.

NODEBR.COM. NODEBR.

QADEER, M. A. et al. Network traffic analysis and intrusion detection using packet sniffer. 2nd International Conference on Communication Software and Networks, ICCSN 2010, p. 313–317, 2010.

REEDER, B.; DAVID, A. Health at hand: A systematic review of smart watch uses for health and wellness. Journal of Biomedical Informatics, v. 63, p. 269–276, 2016.

REHMAN, H. et al. Mobile Health (mHealth) Technology for the Management of Hypertension and Hyperlipidemia: Slow Start but Loads of Potential. Current Atherosclerosis Reports, v. 19, n. 3, 2017.

ROLIM, M. P. Saúde Monitorada: O Uso de Vestíveis para o Acompanhamento à Distância de Sinais do Usuário, 2016.

SANTOS, B. P. et al. Internet das Coisas: da Teoria à Prática. Minicursos SBRC-Simp, p. 50, 2016.

STRONGLOOP. ExpressJS.

TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, v. 14, n. 6, p. 80–83, 2010.

WIKTORSKI, T. NOSQL databases. Advanced Information and Knowledge Processing, p. 77–84, 2019.

WISE, A. Interactive learning aided by JavaScript. Research in Learning Technology, v. 7, n. 2, 2011.

XIA, F.; LAURENCE, Y. Internet of Things. International Journal of Communication Systems, v. 23, n. 5, p. 633–652, 2010.

YOON, H.; PARK, S. H.; LEE, K. T. Lightful user interaction on smart wearables. Personal and Ubiquitous Computing, v. 20, n. 6, p. 973–984, 2016.

YOU, E. VUEJS.

ZAKAS, N. Professional JavaScript for Web Developers. [s.l: s.n.]. v. 3a edição

ZHAO, G. et al. Modeling MongoDB with relational model. Proceedings - 4th International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2013, p. 115–121, 2013.

NAVATHE&ELMASRI. Fundamentals of Database-Systems 7th Edition. 7th. ed. Arlington: Person, 2017.

Anexo B - Código Fonte

App.vue

```
<template>
  <v-app>
    <v-app-bar v-if="user.authenticationKey" app>
      <v-toolbar-title class="headline text-uppercase">
        <button>
          <v-img src="./assets/logo-gate.png" width="50px"></v-img>
        </button>
      </v-toolbar-title>
      <v-spacer></v-spacer>
      <v-toolbar-items>
        <v-btn icon @click="doLogout">
          <v-icon>mdi-exit-to-app</v-icon>
        </v-btn>
      </v-toolbar-items>
    </v-app-bar>
    <v-content>
      <router-view/>
    </v-content>
  </v-app>
</template>

<script>
 import {mapState} from 'vuex'
 import {
  REMOVE_USER
 } from './store/mutationTypes'
 import {GwRequest} from './request'

 export default {
  name: 'App',
```

```
    data: () => ({
      //
    }),
    computed: {
      ...mapState({
        user: state => state.main.user,
      }),
    },
    methods: {
      async doLogout() {
        try {
          let ret = await GwRequest.doLogout();
          if (ret.success) {
            this.$store.commit(REMOVE_USER);
            await this.$router.replace('/');
          }
        } catch (e) {
          console.error(e);
        }
      },
    }
  };
</script>
<style>
    .v-content {
        background-color: #383027;
    }

    footer {
        clear: both;
        position: relative;
        height: 200px;
```

```
    margin-top: -200px;
  }
</style>
```

Login.vue

```
<template>
  <v-container class="fundo" fluid fill-height>
    <v-layout row wrap align-center justify-center>
      <v-flex sm8 md4>
        <div class="layout column align-center">
          <img src="../assets/logo-gate.png" alt="Logo Gatewatch"
            height="250px"/>
        </div>
      </v-flex>
      <v-flex xs6 md3>
        <v-form class="login" @submit.prevent="doLogin">
          <v-text-field
              required v-model="user.login"
              prepend-inner-icon="mdi-account"
              dark
              name="login"
              label="Email"
              type="email"
          ></v-text-field>
          <v-text-field
              required v-model="user.password"
              prepend-inner-icon="mdi-lock"
              dark
              name="password"
              label="Senha"
              id="password"
```

```
                    type="password"
                ></v-text-field>
                <v-btn type="submit" dark text small>
                    ENTRAR
                </v-btn>
            </v-form>
        </v-flex>
    </v-layout>
</v-container>

</template>

<script>
    import {
        GwRequest
    } from '../request/'
    import {
        SET_USER
    } from '../store/mutationTypes'

    export default {
        data: () => ({
            loading: false,
            user: {
                login: "admin@admin.com",
                password: "admin",
            },
        }),
        methods: {
            async doLogin() {
                try {
```

```
                    let ret = await GwRequest.doLogin(this.user.login,
this.user.password);
                    if (!ret.success) return console.error('error', ret.data);
                    this.$store.commit(SET_USER, ret.data);
                    await this.$router.replace('/interfaceadmin');
                  } catch (e) {
                    console.error(e);
                  }
                }
              }
            }
          </script>
          <style scoped>
            .split-bg {
              height: 50%;
              width: 100%;
              position: absolute;
              top: 0;
              left: 0;
              content: "";
              z-index: 0;
            }

            .v-form {
              text-align: center;
              margin-top: 40px;
            }

            .v-btn {
              margin-top: 15px;
            }
```

```
.fundo {
    background-image: linear-gradient(to bottom, #ae916b, #856f53, #5d4e3c, #383027, #161412);
}

.vl {
    border-left: 3px solid #856F53;
    height: 500px;
}

</style>
```

InterfaceAdministrador.vue

```
<template>
    <v-container fill-height fluid>
        <v-layout align-center justify-center row fill-height>
            <v-flex sm4>
                <v-card max-width="350" >
                    <v-list-item>
                        <v-list-item-content>
                            <v-list-item-title  class="headline">Cadastradas</v-list-item-title>
                        </v-list-item-content>
                    </v-list-item>

                    <v-img  src="../../assets/acesso1.png"  width="270px"  class="mx-auto"></v-img>

                    <v-card-text >
                        Lista com todas as empresas cadastradas
                    </v-card-text>
```

```
                    <v-card-actions>
                      <v-btn to="/listaempresas" text>
                        ACESSAR
                      </v-btn>
                    </v-card-actions>
                  </v-card>
                </v-flex>
                <v-flex sm4>
                  <v-card  max-width="350">
                    <v-list-item>
                      <v-list-item-content>
                        <v-list-item-title   class="headline">Solicitações</v-list-item-
title>
                      </v-list-item-content>
                    </v-list-item>

                    <v-img  src="../../assets/acesso2.png"  width="270px"  class="mx-
auto"></v-img>

                    <v-card-text >
                      Solicitações pendentes de acesso
                    </v-card-text>

                    <v-card-actions >
                      <v-btn to="/solicitacaoempresas" text>
                        ACESSAR
                      </v-btn>
                    </v-card-actions>
                  </v-card>
                </v-flex>
                <v-flex>
```

```
                    <v-card  max-width="350">
                        <v-list-item>
                            <v-list-item-content>
                                <v-list-item-title        class="headline">Empresa</v-list-item-
title>
                            </v-list-item-content>
                        </v-list-item>

                        <v-card-text >
                            Solicitações pendentes de acesso
                        </v-card-text>

                        <v-card-actions >
                            <v-btn to="/interfaceempresa" text>
                                ACESSAR
                            </v-btn>
                        </v-card-actions>
                    </v-card>
                </v-flex>
            </v-layout>
        </v-container>
    </template>

    <script>
        export default {
            name: "InterfaceAcesso",
            data: () => ({
                show: false,
                show2: false,
                fotos: [
                    {foto: "TESTE"},
                    {foto: "TESTE2"}
```

```
        ]
      })
  }
</script>

<style scoped>

</style>
```

ListaEmpresa.vue

```
<template>
  <v-container>
    <v-layout mt-6>
      <v-flex>
        <v-data-table
            :headers="headers"
            :items="empresasCadastradas"
            :single-expand="singleExpand"
            :expanded.sync="expanded"
            hide-default-footer
            item-key="name"
            show-expand
            class="elevation-1">
          <template v-slot:top>
            <v-toolbar color="#e1dacf">
              <v-toolbar-title>{{$t('companies.list')}}</v-toolbar-title>
              <v-spacer></v-spacer>
              <v-dialog v-model="dialogCreate" max-width="800px">
                <template v-slot:activator="{ on }">
                  <v-btn color="primary" dark class="mb-2"
                      v-on="on">{{$t('companies.register')}}
```

```
                </v-btn>
            </template>
            <v-card>
                <v-card-title>
                    <span class="headline">Nova empresa</span>
                </v-card-title>

                <v-card-text>
                    <v-container>
                        <v-row justify="center"
                               align="center">
                            <v-col cols="12">
                                <v-row>
                                    <v-col cols="10">
                                        <v-text-field
                                                v-
model="companyData.address.patio.postalCode"
                                                label="Digite o CEP"
                                                outlined
                                        ></v-text-field>
                                    </v-col>
                                    <v-col cols="2">
                                        <v-btn
                                                :loading="buscaCEPBTNload"
                                                @click="getAddressByCEP"
                                                block
                                        >
                                            <v-icon color="primary">
                                                mdi-cloud-search-outline
                                            </v-icon>
                                        </v-btn>
                                    </v-col>
```

```
                    </v-row>
                </v-col>
                <v-col cols="12">
                    <v-row>
                        <v-col cols="9">
                            <v-text-field
                                v-
model="companyData.address.patio.name"
                                label="Rua"
                                outlined
                            ></v-text-field>
                        </v-col>
                        <v-col cols="3">
                            <v-text-field
                                v-
model="companyData.address.number"
                                label="Nº"
                                outlined
                            ></v-text-field>
                        </v-col>
                    </v-row>
                </v-col>
                <v-col cols="12">
                    <v-text-field
                        v-
model="companyData.address.neighborhood.name"
                        label="Bairro"
                        outlined
                    ></v-text-field>
                </v-col>
                <v-col cols="12">
                    <v-row>
```

```
                                        <v-col cols="9">
                                            <v-text-field
                                                v-
model="companyData.address.city.name"
                                                label="Cidade"
                                                outlined
                                            ></v-text-field>
                                        </v-col>
                                        <v-col cols="3">
                                            <v-text-field
                                                v-
model="companyData.address.state.name"
                                                label="Estado"
                                                outlined
                                            ></v-text-field>
                                        </v-col>
                                    </v-row>
                                </v-col>
                                <v-col cols="12">
                                    <v-text-field
                                        v-
model="companyData.address.country.name"
                                        label="País"
                                        outlined
                                    ></v-text-field>
                                </v-col>
                                <v-col cols="12">
                                    <v-row>
                                        <v-col cols="8">
                                            <v-text-field
                                                v-
model="companyData.company.name"
```

```
                                    label="Razão social"
                                    outlined
                                ></v-text-field>
                            </v-col>
                            <v-col cols="4">
                                <v-text-field
                                    v-
model="companyData.company.cnpj"
                                    label="cnpj"
                                    outlined
                                ></v-text-field>
                            </v-col>
                        </v-row>
                    </v-col>
                    <v-col cols="12">
                        <v-row>
                            <v-col cols="10">
                                <v-text-field
                                    v-model="userEmail"
                                    label="Buscar por email"
                                    outlined
                                ></v-text-field>
                            </v-col>
                            <v-col cols="2">
                                <v-btn
                                    :loading="searchUser"
                                    @click="getUserByEmail"
                                    block
                                >
                                    <v-icon color="primary">
                                    mdi-account-search-outline
                                    </v-icon>
```

```
                    </v-btn>
                  </v-col>
                </v-row>
              </v-col>
              <v-col cols="12"
                  v-if="companyAdmin">
                <v-card-text>
                  {{companyAdmin.name}}
                  {{companyAdmin.surname}}
                </v-card-text>
                <v-card-text>
                  {{companyAdmin.email}}
                </v-card-text>
                <v-card-text>
                  {{companyAdmin.birthday}}
                </v-card-text>
              </v-col>
            </v-row>
          </v-container>
        </v-card-text>

        <v-card-actions>
          <v-spacer></v-spacer>
          <v-btn color="blue darken-1" text
              @click="close">Cancel
          </v-btn>
          <v-btn color="blue darken-1" text
              @click="companySave">Save
          </v-btn>
        </v-card-actions>
      </v-card>
    </v-dialog>
```

```
    </v-toolbar>
</template>
<template v-slot:expanded-item="{ headers }">
    <td :colspan="headers.length">
        <v-flex>
            <v-data-iterator
                :items="items"
                :items-per-page.sync="itemsPerPage"
                hide-default-footer
            >
                <template v-slot:default="props">
                    <v-layout mt-2 mb-3 wrap>
                        <v-flex
                            v-for="item in props.items"
                            :key="item.name"
                            ml-1
                        >
                            <v-card>
                                <v-list dense>
                                    <v-list-item>
                                        <v-list-item-content>
                                            Nome:
                                        </v-list-item-content>
                                        <v-list-item-content
                                            class="align-end">
                                            {{ item.name }}
                                        </v-list-item-content>
                                    </v-list-item>
                                    <v-list-item>
                                        <v-list-item-content>
                                            Endereço:
                                        </v-list-item-content>
```

```
    <v-list-item-content
          class="align-end">
      {{ item.patio }}
    </v-list-item-content>
  </v-list-item>
  <v-list-item>
    <v-list-item-content>
      Email:
    </v-list-item-content>
    <v-list-item-content
          class="align-end">
      {{ item.email }}
    </v-list-item-content>
  </v-list-item>
  <v-list-item>
    <v-list-item-content>
      Solicitado em:
    </v-list-item-content>
    <v-list-item-content
          class="align-end">
      {{ item.data }}
    </v-list-item-content>
  </v-list-item>
  <v-list-item>
    <v-list-item-content>
      Observação:
    </v-list-item-content>
    <v-list-item-content
          class="align-end">
      {{ item.observacao }}
    </v-list-item-content>
  </v-list-item>
```

```
<v-divider></v-divider>
<v-list-item>
   <v-switch
         v-model="switch1"
         :label="`Bloquear acesso`"
   ></v-switch>
</v-list-item>
<v-flex ml-4 xs12 sm6
      md4>
   <v-dialog
         ref="dialog"
         v-model="modal"
         :return-value.sync="date"
         persistent
         full-width
         width="290px"
   >
      <template
            v-slot:activator="{ on }">
         <v-text-field
               v-model="date"
               label="Picker in dialog"
               prepend-icon="mdi-calendar"
               readonly
               v-on="on"
         ></v-text-field>
      </template>
      <v-date-picker
            v-model="date"
            scrollable>
         <div class="flex-grow-1"></div>
         <v-btn text
```

```
                                color="primary"
                                @click="modal = false">
                            Cancel
                        </v-btn>
                        <v-btn text
                                color="primary"

@click="$refs.dialog.save(date)">
                            OK
                        </v-btn>
                    </v-date-picker>
                </v-dialog>
            </v-flex>
        </v-list>
    </v-card>
</v-flex>
<v-layout mt-2 justify-center>
    <v-dialog v-model="dialog"
            persistent
            max-width="600px">
        <template
                v-slot:activator="{ on }">
            <v-layout align-center
                    justify-center>
                <div class="my-2">
                    <v-btn color="#383027"
                        v-on="on"
                        fab dark>
                        <v-icon>
                            mdi-pencil
                        </v-icon>
                    </v-btn>
```

```
                    </div>
                </v-layout>
            </template>
            <v-card>
                <v-card-title>
                    <span class="headline">Alteração de
cadastro</span>
                </v-card-title>
                <v-card-text>
                    <v-container
                        grid-list-md>
                        <v-layout wrap>
                            <v-flex xs12>
                                <v-text-field
label="Empresa"></v-text-field>
                            </v-flex>
                            <v-flex xs12>
                                <v-text-field
label="Endereço"></v-text-field>
                            </v-flex>
                            <v-flex xs12>
                                <v-text-field label="Email"></v-
text-field>
                            </v-flex>
                            <v-flex xs12>
                                <v-text-field
label="Observação"></v-text-field>
                            </v-flex>
                        </v-layout>
                    </v-container>
                </v-card-text>
                <v-card-actions>
```

```
                                    <v-spacer></v-spacer>
                                    <v-btn color="blue darken-1"
                                        text
                                        @click="dialog = false">
                                      Fechar
                                    </v-btn>
                                    <v-btn color="blue darken-1"
                                        text
                                        @click="dialog = false">
                                      Alterar
                                    </v-btn>
                                  </v-card-actions>
                                </v-card>
                              </v-dialog>
                            </v-layout>
                          </v-layout>
                        </template>
                      </v-data-iterator>
                    </v-flex>
                  </td>
                </template>
              </v-data-table>
            </v-flex>
          </v-layout>
        </v-container>
</template>

<script>

  import buscaCEP from "../../services/buscaCEP";
  import {GwRequest} from '../../request';
  import i18n from "../../plugins/i18n/pt_BR";
```

```
import i18n_us from "../../plugins/i18n/en_US";

export default {
  name: "ListaEmpresas",
  data() {
    return {
      date: new Date().toISOString().substr(0, 10),
      isActive: false,
      menu: false,
      modal: false,
      menu2: false,
      switch1: true,
      expanded: [],
      singleExpand: false,
      headers: [
        {
          text: 'Company name',
          align: 'center',
          value: 'name',
        },
        {
          text: 'Email',
          align: 'center',
          value: 'email'
        },
        {
          text: 'Responsible',
          align: 'center',
          value: 'responsavel'
        },
        {
          text: 'Device number',
```

```
    align: 'center',
    value: 'dispositivos'
  },
],
empresasCadastradas: [
  {
    name: 'EmpresaA',
    email: "emaila@email.com",
    responsavel: "ResponsavelA",
    dispositivos: 24,
  },
],

itemsPerPageOptions: [4, 8, 12],
itemsPerPage: 4,
items: [
  {
    informacoes: 'Informações',
    name: "Nome da Empresa A",
    patio: "Endereço da Empresa A",
    email: "empresaA@empresaA.com",
    data: "12/06/2019",
    observacao: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas venenatis libero diam, in molestie urna convallis id. Duis tellus augue, blandit at turpis non, aliquam bibendum est. %',
  },
],
dialog: false,

dialogCreate: false,
buscaCEPBTNload: false,
searchUser: false,
```

```
    companyData: {
      company: {
        name: 'company ltda',
        cnpj: '78787878787878',
        admin: null,
      },
      address: {
        number: 179,
        patio: {
          name: null,
          postalCode: '88102300',
        },
        neighborhood: {
          name: null,
        },
        city: {
          name: null,
        },
        state: {
          name: null,
        },
        country: {
          name: 'Brasil',
        },
      },
    },
    userEmail: 'juniorsin2012@gmail.com',
    companyAdmin: null,
  }
},
computed: {
  save() {
```

```
    return this.formatDate(this.date)
  }
 },
 methods: {
  formatDate(date) {
    if (!date) return null;

    const [year, month, day] = date.split('-')
    return `${month}/${day}/${year}`
  },
  parseDate(date) {
    if (!date) return null

    const [month, day, year] = date.split('/')
    return `${year}-${month.padStart(2, '0')}-${day.padStart(2, '0')}`
  },
  async getUserByEmail() {
    this.searchUser = true;
    try {
      let ret = await GwRequest.getUserByEmail(this.userEmail);
      if (!ret.data.success) return console.error(ret.data.data);
      this.companyAdmin = ret.data.data;
      this.companyData.company.admin = this.companyAdmin.id;
    } catch (e) {
      console.error('error', e);
    }
    this.searchUser = false;
  },
  async getAddressByCEP() {
    this.buscaCEPBTNload = true;
    try {
```

```
            let ret = await buscaCEP.getAddressByCEP(this.companyData.address.patio.postalCode);
            this.companyData.address.patio.name = ret.logradouro;
            this.companyData.address.neighborhood.name = ret.bairro;
            this.companyData.address.city.name = ret.localidade;
            this.companyData.address.state.name = ret.uf;
          } catch (e) {
            console.error(e);
          }
          this.buscaCEPBTNload = false;
        },
        close() {
          this.dialogCreate = false;
          setTimeout(() => {
            this.editedItem = Object.assign({}, this.defaultItem);
            this.editedIndex = -1
          }, 300)
        },
        async companySave() {
          console.log('salvar', JSON.parse(JSON.stringify(this.companyData)));
          try {
            let ret = await GwRequest.createCompany(this.companyData);
            console.log('ret', ret);
          } catch (e) {
            console.error(e);
          }
          this.close();
        },
      },
      async created() {
      }
    }
```

```
</script>

<style scoped>

</style>
```

SolicitacaoEmpresa.vue

```
<template>
  <v-container>
    <v-layout mt-5>
      <v-flex align-center>
        <v-data-table
            :headers="headers"
            :items="empresasCadastradas"
            :single-expand="singleExpand"
            :expanded.sync="expanded"
            item-key="name"
            show-expand
            class="elevation-1"
            >
          <template v-slot:top>
            <v-toolbar color="#e1dacf">
              <v-toolbar-title>Solicitações</v-toolbar-title>
              <v-spacer></v-spacer>
            </v-toolbar>
          </template>
          <template v-slot:expanded-item="{ headers }">
            <td :colspan="headers.length">
              <v-flex align-self-center>
                <v-data-iterator
                    :items="items"
```

```
                    :items-per-page.sync="itemsPerPage"
                    hide-default-footer
                >
                    <template v-slot:default="props">
                        <v-layout mt-2 mb-3 wrap>
                            <v-flex
                                v-for="item in props.items"
                                :key="item.name"
                                ml-1
                            >
                                <v-card>
                                    <v-list dense>
                                        <v-list-item>
                                            <v-list-item-content>Nome:</v-list-item-
content>
                                            <v-list-item-content class="align-
center">{{ item.nome }}</v-list-item-content>
                                        </v-list-item>
                                        <v-list-item>
                                            <v-list-item-content>Endereço:</v-list-
item-content>
                                            <v-list-item-content class="align-
center">{{ item.endereco }}</v-list-item-content>
                                        </v-list-item>
                                        <v-list-item>
                                            <v-list-item-content>Telefone:</v-list-
item-content>
                                            <v-list-item-content class="align-
center">{{ item.telefone }}</v-list-item-content>
                                        </v-list-item>
                                        <v-list-item>
```

```
                                        <v-list-item-content>Email:</v-list-item-
content>
                                        <v-list-item-content class="align-
center">{{ item.email }}</v-list-item-content>
                                    </v-list-item>
                                    <v-divider></v-divider>
                                    <v-list-item>
                                        <v-list-item-content>Solicitado em:</v-
list-item-content>
                                        <v-list-item-content class="align-
center">{{ item.data }}</v-list-item-content>
                                    </v-list-item>
                                    <v-list-item>
                                        <v-list-item-content>Observação:</v-list-
item-content>
                                        <v-list-item-content class="align-
center">{{ item.observacao }}</v-list-item-content>
                                    </v-list-item>
                                </v-list>
                            </v-card>
                        </v-flex>
                        <v-layout mt-2 justify-center>
                            <v-dialog v-model="dialog" persistent max-
width="600px">
                                <template v-slot:activator="{ on }">
                                    <v-layout align-center justify-center>
                                        <div>
                                            <v-btn class="button-reject"
color="error" v-on="on" fab dark>
                                                <v-icon>mdi-minus</v-icon>
                                            </v-btn>
                                        </div>
```

```
                    <div>
                        <v-btn              class="button-accept"
color="primary" v-on="on" fab dark>
                            <v-icon>mdi-check</v-icon>
                        </v-btn>
                    </div>
                </v-layout>
            </template>
            <v-card>
                <v-card-title class="headline">Aceitação da
solicitação</v-card-title>

                <v-card-text>
                    Após aceitar essa liberação de acesso, seu
cliente terá acesso às informações dos dispositivos.
                    Tem certeza dessa ação?
                </v-card-text>

                <v-card-actions>
                    <v-spacer></v-spacer>

                    <v-btn
                        color="red darken-1"
                        text
                        @click="dialog = false"
                    >
                        Rejeitar
                    </v-btn>

                    <v-btn
                        color="green darken-1"
                        text
```

```
                                        @click="dialog = false"
                                    >
                                        Aceitar
                                    </v-btn>
                                </v-card-actions>
                            </v-card>
                        </v-dialog>
                    </v-layout>
                </v-layout>
            </template>
        </v-data-iterator>
    </v-flex>
</td>
</template>
</v-data-table>
</v-flex>
</v-layout>
</v-container>
</template>

<script>
    export default {
        name: "SolicitacaoEmpresas",
        data () {
            return {
                expanded: [],
                singleExpand: false,
                headers: [
                    {
                        text: 'Nome da empresa',
                        align: 'center',
                        value: 'name',
```

```
    },
    {
      text: 'Email',
      align: 'center',
      value: 'email'
    },
    {
      text: 'Responsável',
      align: 'center',
      value: 'responsavel'
    },
    {
      text: 'Número de Dispositivos',
      align: 'center',
      value: 'dispositivos'
    },
  ],
  empresasCadastradas: [
    {
      name: 'EmpresaA',
      email: "emaila@email.com",
      responsavel: "ResponsavelA",
      dispositivos: 24,
    },
    {
      name: 'EmpresaB',
      email: "emailb@email.com",
      responsavel: "ResponsavelB",
      dispositivos: 37,
    },
    {
      name: 'EmpresaC',
```

```
    email: "email@email.com",
    responsavel: "ResponsavelC",
    dispositivos: 23,
},
{
    name: 'EmpresaD',
    email: "email@email.com",
    responsavel: "ResponsavelD",
    dispositivos: 67,
},
{
    name: 'EmpresaE',
    email: "email@email.com",
    responsavel: "ResponsavelE",
    dispositivos: 49,
},
{
    name: 'EmpresaF',
    email: "email@email.com",
    responsavel: "ResponsavelE",
    dispositivos: 94,
},
{
    name: 'EmpresaG',
    email: "email@email.com",
    responsavel: "ResponsavelE",
    dispositivos: 98,
},
{
    name: 'EmpresaH',
    email: "email@email.com",
    responsavel: "ResponsavelE",
```

```
            dispositivos: 87,
        },
        {
            name: 'EmpresaI',
            email: "email@email.com",
            responsavel: "ResponsavelE",
            dispositivos: 51,
        },
        {
            name: 'EmpresaJ',
            email: "email@email.com",
            responsavel: "ResponsavelE",
            dispositivos: 65,
        },
    ],

    itemsPerPageOptions: [4, 8, 12],
    itemsPerPage: 4,
    items: [
        {
            informacoes: 'Informações',
            nome: "Nome da Empresa A",
            endereco: "Endereço da Empresa A",
            telefone: 4844488544,
            email: "empresaA@empresaA.com",
            data: "12/06/2019",
            observacao: 'Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Maecenas venenatis libero diam, in molestie urna convallis id. Duis tellus augue,
blandit at turpis non, aliquam bibendum est. %',
        },
    ],
```

```
            dialog: false,
        }

    },
}
</script>

<style scoped>

    .button-reject{
        margin-right: 15px;
    }

</style>
```

```
                    let schema_options = {
  timestamps: true,
  toObject: {
    virtuals: true,
    transform: function (doc, ret) {
      delete ret._id;
      delete ret.__v;
      return ret;
    }
  },
  toJSON: {
    virtuals: true,
    transform: function (doc, ret) {
      delete ret._id;
      delete ret.__v;
      return ret;
    }
  }
};

let schema = new Schema(Object.assign({
  name: {
    type: Schema.Types.String,
    trim: true,
    required: [true, 'nameRequired'],
    unique: true,
  },
}, BaseSchema), schema_options);

let modelSchema = model("city", schema);
export {modelSchema as Model};
```

```
import {BasicHandler} from "../BasicHandler";
import {FindObject} from "../util/FindObject";
import {UpdateObject} from "../util/UpdateObject";

export class Admin extends BasicHandler {

  public async createCompany (param: companyCreate) {
    let required = this.attributeValidator([
      'auth', "address", ["number", "patio", ["name", "postalCode"],
        "neighborhood", ["name"], "city", ["name"], "state", ["name"],
        "country", ["name"]], "company", ["admin", "name", "cnpj"],
    ], param);
    if(!required.success)
      return await this.getErrorAttributeRequired(required.error);
    try {
      await this.isAdmin(param.auth);
      let address = await this.createAddress(param.address);
      let ret = await this.sendToServer('db.company.create', {
        admin: param.company.admin,
        name: param.company.name,
        cnpj: param.company.cnpj,
        address: {
          number: param.address.number,
          patio: address.patio.id,
          neighborhood: address.neighborhood.id,
          city: address.city.id,
          state: address.state.id,
          country: address.country.id,
        }
      });
      return await this.returnHandler({
        model: 'user',
```

```
      data: ret.data,
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
      data: {error: e.message || e},
    })
  }
}

public async getUsers (param: defaultParam) {
  let required = this.attributeValidator([
    'auth'
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  try {
    await this.isAdmin(param.auth);
    let ret = await this.sendToServer('db.user.read',
      new FindObject({
        query: {
          isSystemMng: false,
        },
        select: 'name surname birthday email id',
      }));
    return await this.returnHandler({
      model: 'user',
      data: ret.data,
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
```

```
      data: {error: e.message || e},
    })
  }
}

public async getUserByEmail (param: userByEmail) {
  let required = this.attributeValidator([
    'auth', 'email'
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  try {
    await this.isAdmin(param.auth);
    let ret = await this.sendToServer('db.user.read',
      new FindObject({
        findOne: true,
        query: {
          email: param.email,
        },
        select: 'name surname birthday email id',
      }));
    return await this.returnHandler({
      model: 'user',
      data: ret.data,
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
      data: {error: e.message || e},
    })
  }
}
```

```
public async updateUser (param: userUpdate) {
  let required = this.attributeValidator([
    'auth', 'toUpdate', ['id', 'update', '$or', ['name', 'surname', 'birthday', 'email']]
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  try {
    await this.isAdmin(param.auth);
    let ret = await this.updateUserData(param.toUpdate);
    return await this.returnHandler({
      model: 'user',
      data: ret.data,
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
      data: {error: e.message || e},
    })
  }
}

public async updateUserPassword (param: userUpdatePassword) {
  let required = this.attributeValidator([
    'auth', 'toUpdate', ['id', 'update']
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  try {
    await this.isAdmin(param.auth);
    let ret = await this.updateUserPasswordData(param.toUpdate);
    if(!ret.data.success || ret.data.error) return await this.returnHandler({
```

```
      model: 'user',
      data: {error: 'weCantUpdateThisPassword'},
    });
    return await this.returnHandler({
      model: 'user',
      data: {success: true},
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
      data: {error: e.message || e},
    })
  }
}

public async deleteUser (param: userById) {
  let required = this.attributeValidator([
    'auth', 'id'
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  try {
    await this.isAdmin(param.auth);
    let ret = await this.sendToServer('db.user.update',
      new UpdateObject({
        query: param.id,
        update: {
          removed: true,
        },
      }),
    );
    if(!ret.data.success || ret.data.error)
```

```
      return await this.returnHandler({
        model: 'user',
        data: {
          error: 'weCantRemoveUser',
        }
      });
    return await this.returnHandler({
      model: 'user',
      data: {
        success: {
          removed: ret.data.success.removed,
        },
      },
    });
  } catch (e) {
    return await this.returnHandler({
      model: 'user',
      data: {error: e.message || e},
    })
  }
}

private async createAddress (param: addressCreate) {
  try {
    let readPromises = await Promise.all([
      this.sendToServer('db.patio.read', new FindObject({
        findOne: true,
        query: {
          postalCode: param.patio.postalCode,
        },
        select: 'id',
      })),
```

```
            this.getComponentAddressIdByName('neighborhood',
param.neighborhood.name),
            this.getComponentAddressIdByName('city', param.city.name),
            this.getComponentAddressIdByName('state', param.state.name),
            this.getComponentAddressIdByName('country', param.country.name),
          ]);
          let toCheck = {
            patio: readPromises[0].data.success,
            neighborhood: readPromises[1].data.success,
            city: readPromises[2].data.success,
            state: readPromises[3].data.success,
            country: readPromises[4].data.success,
          };
          await this.createIfNull(toCheck, param);
          for (let att in toCheck) {
            if(toCheck.hasOwnProperty(att) && !toCheck[att]) {
              throw new Error(`${att}CreateError`);
            }
          }
          return toCheck;
        } catch (e) {
          return e;
        }
      }

      private async createIfNull (param: { patio, neighborhood, city, state, country
}, address: addressCreate) {
        let promises = [];
        for (let attr in param) {
          if(param.hasOwnProperty(attr) && !param[attr]) {
            promises.push(this.sendToServer(`db.${attr}.create`, address[attr]));
          }
```

```
        }
        if(!promises.length) return;
        let created = await Promise.all(promises);
        for (let i = 0; i < created.length; i++) {
          let event = created[i].event;
          param[event.slice(event.indexOf('.') + 1, event.lastIndexOf('.'))] = created[i].data.success.length ? created[i].data.success[0] : null;
        }
      }

      private async getComponentAddressIdByName (componentName: string, name: string) {
        if(!componentName &&
          componentName !== 'neighborhood' &&
          componentName !== 'city' &&
          componentName !== 'state' &&
          componentName !== 'country'
        ) return;
        return this.sendToServer(`db.${componentName}.read`, new FindObject({
          findOne: true,
          query: {
            name,
          }
        }));
      }

      private async isAdmin (authenticationKey) {
        return new Promise(async (resolve, reject) => {
          let ret = await this.sendToServer('db.user.read', new FindObject({
            findOne: true,
            query: {
              authenticationKey,
```

```
        }
      }));
      if(!ret.data.success || ret.data.error) return reject('userValidationFailed');
      if(!ret.data.success.isSystemMng) return reject('invalidUser');
      resolve(ret.data.success);
    });
  }

}

export default new Admin();

interface defaultParam {
  auth: string,
}

interface addressCreate {
  number: number,
  patio: {
    name: string,
    postalCode: string,
  },
  neighborhood: {
    name: string,
  },
  city: {
    name: string,
  },
  state: {
    name: string,
  },
  country: {
```

```
    name: string,
  },
}

interface companyCreate extends defaultParam {
  address: addressCreate,
  company: {
    admin: string,
    name: string,
    cnpj: string,
  },
}

interface userByEmail extends defaultParam {
  email: string,
}

interface userById extends defaultParam {
  id: string,
}

interface userUpdate extends defaultParam {
  toUpdate: {
    id: string,
    update: {
      name?: string,
      surname?: string,
      birthday?: number,
      email?: string,
    }
  },
}
```

```
interface userUpdatePassword extends defaultParam {
  toUpdate: {
    id: string,
    update: {
      password: string,
    }
  }
}

import {BasicHandler} from "../BasicHandler";
import {FindObject} from '../util/FindObject';
import {Application} from '../../Application';
import * as path from 'path';

export class OpenHandler extends BasicHandler {

  /**
   * * Return model data, if he has a account or access error if he don't has a account.
   *
   * @param data
   */
  public async login (data: loginData): Promise<returnData> {
    let required = this.attributeValidator(['login', 'password'], data);
    if(!required.success) return await this.getErrorAttributeRequired(required.error);
    let ret = await this.sendToServer('db.user.login', {
      password: data.password,
      queryObject: new FindObject({
        query: {
          email: data.login,
```

```
            removed: false,
          },
          select: 'id name surname email password type authenticationKey isSystemMng',
        }),
      });
      if(ret.data.error) return await this.returnHandler({
        model: 'user',
        data: ret.data,
      });
      let accessKey = await this.sendToServer('accessSession.create', ret.data.success.authenticationKey);
      if(accessKey.data.error) return await this.returnHandler({
        model: 'user',
        data: accessKey.data,
      });
      ret.data.success.accessKey = accessKey.data.success;
      return await this.returnHandler({
        model: 'user',
        data: ret.data,
      });
    }

    public async getLocale (data) {
      let dataLocale = Application.getPathI18N();
      let i18n = data.i18n ? data.i18n : dataLocale.defaultI18N;
      let ret = await this.getI18N({path: path.resolve(`${dataLocale.mainPath}/${i18n}/${dataLocale.i18n}`)});
      return await this.returnHandler({
        model: 'global',
        data: ret
      })
```

```
  }

  public async createUser (param: userCreate) {
    let required = this.attributeValidator([
      'name', 'surname', 'birthday', 'email', 'password'
    ], param);
    if(!required.success)
      return await this.getErrorAttributeRequired(required.error);
    try {
      let ret = await this.sendToServer('db.user.create', {
        name: param.name,
        surname: param.surname,
        birthday: param.birthday,
        email: param.email,
        password: param.password,
      });
      return await this.returnHandler({
        model: 'user',
        data: ret.data,
      });
    } catch (e) {
      return await this.returnHandler({
        model: 'user',
        data: {error: e.message || e},
      })
    }
  }

}

interface returnData {
  success: boolean,
```

```
  data: any
}

interface loginData {
  login: string,
  password: string
}

interface userCreate {
  name: string,
  surname: string,
  birthday: number,
  email: string,
  password: string,
}

export default new OpenHandler();

import {BasicHandler} from "../BasicHandler";
import {FindObject} from "../util/FindObject";
import {UpdateObject} from "../util/UpdateObject";

export class User extends BasicHandler {

  public async logout (param: logoutParam) {
    let required = this.attributeValidator(['auth', 'aKey'], param);
    if(!required.success) return await this.getErrorAttributeRequired(required.error);
    let ret = await this.sendToServer('accessSession.removeKey', {
      authenticationKey: param.auth,
      accessKey: param.aKey,
    });
```

```
  return await this.returnHandler({
    model: 'user',
    data: ret.data,
  });
}

public async updateMe (param: userUpdate) {
  let required = this.attributeValidator([
    'id', 'update', '$or', ['name', 'surname', 'birthday', 'email'],
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  let ret = await this.updateUserData(param);
  return await this.returnHandler({
    model: 'user',
    data: ret.data,
  });
}

public async updateMePassword (param: userUpdatePassword) {
  let required = this.attributeValidator([
    'id', 'update', ['password', 'currentPassword'],
  ], param);
  if(!required.success)
    return await this.getErrorAttributeRequired(required.error);
  let checkPassword = await this.sendToServer('db.user.verifyPassword', {
    userId: param.id,
    password: param.update.currentPassword,
  });
  if(checkPassword.data.error) return await this.returnHandler({
    model: 'user',
    data: {error: 'weCantUpdatePassword'}
```

```
    });
    if(!checkPassword.data.success) return await this.returnHandler({
      model: 'user',
      data: {error: 'wrongPassword'}
    });
    let ret = await this.updateUserPasswordData(param);
    if(!ret.data.success || ret.data.error) return await this.returnHandler({
      model: 'user',
      data: {error: 'weCantUpdateThisPassword'},
    });
    return await this.returnHandler({
      model: 'user',
      data: {success: true},
    });
  }

  public async readMe (param: defaultParam) {
    let required = this.attributeValidator(['auth'], param);
    if(!required.success) return await this.getErrorAttributeRequired(required.error);
    let ret = await this.sendToServer('db.user.read', new FindObject({
      findOne: true,
      query: {
        authenticationKey: param.auth,
      },
      select: 'isSystemMng name surname birthday email id authenticationKey'
    }));
    if(!ret.data.success || ret.data.error) return await this.returnHandler({
      model: 'user',
      data: {error: 'cantReadUserInfos'},
    });
```

```
            let companies = await this.sendToServer('db.company.read', new
FindObject({
              query: {
                admin: ret.data.success.id,
              },
              select: 'name',
            }));
            if(companies.data.error) return await this.returnHandler({
              model: 'user',
              data: {error: 'serverError'}
            });
            ret.data.success['companies'] = companies.data.success;
            return await this.returnHandler({
              model: 'user',
              data: ret.data,
            });
          }

          public async deleteMe (param: defaultUpdate) {
            let required = this.attributeValidator(['id'], param);
            if(!required.success) return await
this.getErrorAttributeRequired(required.error);
            let ret = await this.sendToServer('db.user.update', new UpdateObject({
              query: param.id,
              update: {
                removed: true,
              },
            }));
            await this.sendToServer('db.user.update',
              new UpdateObject({
                query: param.id,
                update: {
```

```
          removed: true,
        },
      }),
    );
    if(!ret.data.success || ret.data.error)
      return await this.returnHandler({
        model: 'user',
        data: {
          error: 'weCantRemoveUser',
        }
      });
    return await this.returnHandler({
      model: 'user',
      data: {
        success: {
          removed: ret.data.success.removed,
        },
      },
    });
  }

}

export default new User();

interface defaultParam {
  auth: string,
}

interface logoutParam extends defaultParam {
  aKey: string,
}
```

```
interface userUpdate {
  id: string,
  update: {
    name?: string,
    surname?: string,
    birthday?: number,
    email?: string,
  }
}

interface defaultUpdate {
  id: string,
}

interface userUpdatePassword extends defaultUpdate{
  update: {
    password: string,
    currentPassword: string,
  }
}

import * as BBPromise from "bluebird";
import * as fs from 'fs';
import * as path from 'path';
import {Source} from "../events/Source";
import {Util} from "../util/Util";
import {UpdateObject} from "./util/UpdateObject";

export class BasicHandler extends Source {

  constructor () {
```

```
    super();
  }

  protected sendToServer (event, dado): BBPromise<any> {
    return this.hub.send(this, event, {success: dado, error: null,}).promise;
  }

  /**
   * Verifica os erros de validacao e retorna o correspondente.
   *
   * @param model
   * @param errors
   * @returns {Promise<[any , any , any , any , any , any , any , any , any ,
any]>}
   */
  private async getErrorsValidation (model, errors) {
    let errorsArray = [];
    for (let attr in errors) {
      if(errors.hasOwnProperty(attr) && !errors[attr].errors) {
        errorsArray.push(Util.getErrorByLocale('pt-Br',                    model,
errors[attr].message));
      }
    }
    return await Promise.all(errorsArray);
  }

  private async getErrorsDuplicationKey (model, msgError) {
    let key = `duplicated.${msgError.slice(msgError.indexOf('index:') + 7,
msgError.indexOf('_1 dup'))}`;
    return await Util.getErrorByLocale('pt-Br', model, key);
  }
```

```
private async getErrorsByLocale (model, msgError) {
  return await Util.getErrorByLocale("pt-Br", model, msgError);
}

/**
 * Verifica o tipo de erro e pega o padrao de erro correspondente.
 *
 * @param {string} model
 * @param error
 * @returns {Promise<any>}
 */
private async getError (model: string, error: any) {
  if(typeof error === 'string') {
    return await this.getErrorsByLocale(model, error);
  } else if(typeof error === 'object') {
    if(error.hasOwnProperty('name')) {
      if(error.name === "ValidationError") {
        return await this.getErrorsValidation(model, error.errors);
      } else if(error.name === 'MongoError') {
        if(error.code && error.code === 11000) {
          return await this.getErrorsDuplicationKey(model, error.errmsg);
        }
      } else if(error.name === 'CastError') {
        if(!error.reason) return await this.getErrorsByLocale(model, `${error.name}.${error.path}.${error.kind}`);
        return await this.getErrorsByLocale(model, `${error.reason.name}.${error.reason.path}.${error.reason.kind}`);
      }
    } else if(error.hasOwnProperty('index') && error.hasOwnProperty('msg')) {
      let errorReturn = await this.getErrorsByLocale(model, error.msg);
      errorReturn.description = errorReturn.description + error.index;
      return errorReturn;
```

```
    } else if(error.type === "attributeRequired") {
      let errorReturn = await this.getErrorsByLocale(model, error.type);
      errorReturn.description = `${error.errorMessage} is required`;
      return errorReturn;
    }
  }
}

/**
 * Funcao responsave por fazer tratamento de retornos, antes de serem
 * enviados para o cliente.
 * Se conter erro, busca o erro correspondente.
 *
 * @returns {Promise<{success, data}>}
 * @param ret
 */
async returnHandler (ret: { model: string, data: any }) {
  if(ret.data.error) {
    return {
      success: false,
      data: await this.getError(ret.model, ret.data.error),
    };
  }
  return {
    success: true,
    data: ret.data.success
  };
}

async updateValidator (data) {
  if(!data.id) return await this.returnHandler({
    model: 'global',
```

```
        data: {
          error: 'update.idRequired'
        }
      });
      if(!data.update || !Object.keys(data.update).length) return await this.returnHandler({
        model: 'global',
        data: {
          error: 'update.updateRequire'
        }
      });
      return {
        success: true,
      };
    }

    /**
     *
     * @param attributes
     * @returns {Promise<{success: boolean; data: errorMessage | any[]} | {success: boolean; data: any}>}
     *
     * Chama a funcao para retornar o erro correto.
     */
    protected async getErrorAttributeRequired (attributes) {
      return this.returnHandler({
        model: 'global',
        data: {
          error: {
            type: "attributeRequired",
            errorMessage: attributes,
          }
```

```
    }
   })
  }

  protected async getI18N (data: { path: string }) {
   let required = this.attributeValidator(['path'], data);
   if(!required.success) return await
this.getErrorAttributeRequired(required.error);
   try {
    let files = fs.readdirSync(data.path);
    let success = {};
    for (let i = 0; i < files.length; i++) {
     success[files[i].split('.')[0]] = require(path.join(data.path, files[i]));
    }
    return {success};
   } catch (error) {
    return {error};
   }
  }

  protected getUpdateObject (allowedAttributes: string[], data: object) {
   let update = {};
   let objectKeys = Object.keys(data);
   let allowedSet = new Set(allowedAttributes);
   for (let i = 0; i < objectKeys.length; i++) {
    if(allowedSet.has(objectKeys[i])) update[objectKeys[i]] =
data[objectKeys[i]];
   }
   return update;
  }

  protected updateUserData (param: userUpdate) {
```

```
    const allowedAttributes = ['name', 'surname', 'birthday', 'email'];
    return this.sendToServer('db.user.update',
      new UpdateObject({
        query: param.id,
        update: this.getUpdateObject(allowedAttributes, param.update),
        select: allowedAttributes,
      }));
  }

  protected updateUserPasswordData (param: userUpdatePassword) {
    return this.sendToServer('db.user.update',
      new UpdateObject({
        query: param.id,
        update: this.getUpdateObject(["password"], param.update),
      }));
  }

}

interface userUpdate {
  id: string,
  update: {
    name?: string,
    surname?: string,
    birthday?: number,
    email?: string,
  }
}

interface userUpdatePassword {
  id: string,
  update: {
```

```
    password,
  }
}


import {EventEmitter2} from "eventemitter2";
import {Message} from "./Message";
import * as BBPromise from "bluebird";
import {Source} from "./Source";
import * as path from 'path';

export class Hub {
                        eventemitter: EventEmitter2;
                        private static instance: Hub;

                        private constructor() {
    let config:any = require(path.resolve("config.json"));
    this.eventemitter = new EventEmitter2(config.eventConfig);
                        }

                        /**
                         * Adds a listener to the end of the listeners array for the
specified event.
                         * @param event
                         * @param listener
                         */
                        public on(event: string, listener: Function): Hub {
                        this.eventemitter.on(event, listener);
                        return this;
                        }

                        /**
```

```
     * Adds a one time listener for the event.
     * The listener is invoked only the first time the event is fired, after which it is removed.
     * @param event
     * @param listener
     */
    public once(event: string, listener: Function): Hub {
    this.eventemitter.once(event, listener);
    return this;
    }

    /**
     * Remove a listener from the listener array for the specified event.
     * Caution: changes array indices in the listener array behind the listener.
     * @param event
     * @param listener
     */
    public un(event: string, listener: Function): Hub {
    this.eventemitter.off(event, listener);
    return this;
    }

    /**
     * Send a message into the HUB.
     *
     * @param source
     * @param event
     * @param data
     * @param previous
     * @return {Message}
```

```
                    */
                    public send(source: Source, event: string, data, previous?:
string): Message {
                    // @ts-ignore
                    if (!source || (typeof source !== "object" && !(source
instanceof Source))) {
                    throw new Error("To send message through Hub, the
source is required " +
                    "and must extend Source");
                    }

                    let msg = new Message(source.id, event, data, previous);
                    process.nextTick(() => {
                    try {
                    this.eventemitter.emit(event, msg);
                    } catch (e) {
                    console.error("Tentando enviar dados para um HUB
Destruido", event, source, e);
                    }
                    });

                    return msg;
                    }

                    public destroy(): BBPromise<boolean> {
                    return new BBPromise<boolean>((resolve) => {
                    Hub.instance = null;
                    this.eventemitter.removeAllListeners();
                    this.eventemitter = null;
                    resolve(true);
                    });
                    }
```

```
                public static getInstance(): Hub {
                if (!Hub.instance) {
                Hub.instance = new Hub();
                }

                return Hub.instance;
                }
        }

        import {MessageData} from "./MessageData";
        import * as BBPromise from "bluebird";
        import {v4} from "node-uuid";
        import {Hub} from "./Hub";

        export class Message {
                private _previous_message: string;
                private _promise: BBPromise<any>;
                private _source_id: string;
                private _data: MessageData;
                private _event: string;
                private _id: string;

                constructor(source_id: string, event: string, data:
MessageData, previous_message?: string) {
                this.id = v4();
                this.data = data; // {error: error, success: success}
                this.source_id = source_id;
                this.previous_message = previous_message;
                this.event = event;
                }
```

```
                    set source_id(source_id) {
                    this._source_id = source_id;
                    }

                    get source_id() {
                    return this._source_id;
                    }

                    set id(id) {
                    this._id = id;
                    }

                    get id() {
                    return this._id;
                    }

                    set previous_message(previous_message) {
                    this._previous_message = previous_message;
                    }

                    get previous_message() {
                    return this._previous_message;
                    }

                    set data(data) {
                    if (!data || !data.hasOwnProperty('success') ||
!data.hasOwnProperty('error'))
                    throw new Error(`Mensagem no formato incorreto.
                     Esperado: {success: any, error: any}.
                     Recebido ${JSON.stringify(data)}`);

                    this._data = data;
```

```
}

get data() {
return this._data;
}

set event(event) {
this._event = event;
}

get event() {
return this._event;
}

set promise(promise) {
this._promise = promise;
}

get promise() {
if (this._promise) return this._promise;

let hub = Hub.getInstance();

let promiseHandler = null;
this.promise = new BBPromise((resolve) => {
promiseHandler = (message) => {
if (this.id === message.previous_message) {
hub.un(this.event, promiseHandler);
return resolve(message);
}
}
});
```

```
                        hub.on(this.event, promiseHandler);
                        this.promise.timeout(30000).catch((e) => {// If we have an
catch/timeout remove this from HUB;
                        console.log("error", e);
                        hub.un(this.event, promiseHandler);
                        });

                        return this._promise;
                        }
        }

        import {Hub} from "./Hub";
        import {v4} from "node-uuid";

        export class Source {
          _id: string;
          _hub: Hub;

          constructor() {
            this.id = v4();
            this.hub = Hub.getInstance();
          }

          set id(uuid) {
            this._id = uuid;
          }

          get id() {
            return this._id;
          }

          set hub(hub) {
```

```
    this._hub = hub;
  }

  get hub() {
    return this._hub;
  }

  /**
   *
   * @param {any[]} requiredAttributes
   * @param data
   * @returns {boolean}
   *
   * Verifica se existe algum dos atributos dos dados passados.
   */
  private orValidator(requiredAttributes: any[], data: any) {
    for (let i = 0; i < requiredAttributes.length; i++) {
      if (typeof data[requiredAttributes[i]] !== 'undefined' && data[requiredAttributes[i]] !== null) return true;
    }
    return false;
  }

  /**
   *
   * @param {any[]} requiredAttributes
   * @param data
   * @returns {any}
   *
   * Faz o tratamento para verificar se os dados obrigatorios da funcao estao sendo mandados.
   */
```

```
protected attributeValidator(requiredAttributes: any[], data: any): { success: boolean, error?: string } {
  for (let i = 0; i < requiredAttributes.length; i++) {
   if (Array.isArray(requiredAttributes[i])) {
    let verified = this.attributeValidator(requiredAttributes[i], data[requiredAttributes[i - 1]]);
    if (!verified.success) {
     verified.error = `${requiredAttributes[i - 1]}:{ ${verified.error} }`;
     return verified;
    }
   } else {
    if (requiredAttributes[i] === "$or") {
     let orVerified = this.orValidator(requiredAttributes[i + 1], data[requiredAttributes[i - 1]]);
     if (!orVerified) {
      let errorMessage = ``;
      for (let e = 0; e < requiredAttributes[i + 1].length; e++) {
       if (e !== requiredAttributes[i + 1].length - 1) {
        errorMessage = `${errorMessage} ${requiredAttributes[i + 1][e]} or `
       } else {
        errorMessage = `${errorMessage} ${requiredAttributes[i + 1][e]} `
       }
      }
      i++;
      return {
       success: false,
       error: `${errorMessage}`
      }
     }
     i++;
    } else {
```

```
        if (typeof data[requiredAttributes[i]] === 'undefined' ||
          data[requiredAttributes[i]] === null) return {
          success: false,
          error: `${requiredAttributes[i]}`
        };
       }
     }
   }
   return {success: true};
  }

  /**
   * Call this method everytime, to avoid leak on the HUB. if you like
   * to override this,
   * call super.destroy after your code.
   */
  destroy() {
    this.hub.send(this, "hub.core.source.destroyed", {
      error: null,
      success: this.id,
    });
    this.id = null;
  }
}

export interface Populate {
  path: string,
  select?: string,
  populate?: Populate[],
  model?: string
}
```

```
export class BaseUtil {
  protected _query: object = {removed: false};
  private _populate: Populate[];
  private _id: string;

  constructor() {};

  protected set id(id: string) {
    this.clear();
    this._id = id;
  }

  protected get id(): string {
    return this._id;
  }

  protected set query(query: object | string) {
    if (typeof query === "string") {
      this.id = query;
    } else {
      this._query = {...this._query, ...query};
    }
  }

  protected get query(): object | string {
    return this._query;
  }

  protected set populate(populate: Populate[]) {
    if (populate) this._populate = this.handlerPopulate(populate);
  }
```

```
  protected get populate(): Populate[] {
    return this._populate;
  }

  /**
   * Limpa os dados que são desnecessarios na busca pelo id.
   */
  protected clear() {
    delete this.query;
  }

  /**
   *
   * @param {Populate[]} populate
   * @returns {Populate[]}
   *
   * Ajusta o populate quando é chamando.
   */
  protected handlerPopulate(populate: Populate[]): Populate[] {
    if (!Array.isArray(populate)) populate = [populate];
    for (let i = 0; i < populate.length; i++) {
      if (populate[i].hasOwnProperty('select')) populate[i].select = 'id ' + populate[i].select;
      if (populate[i].hasOwnProperty('populate')) populate[i].populate = this.handlerPopulate(populate[i].populate);
    }
    return populate;
  }
}

import {BaseUtil,Populate} from './BaseUtil';
```

```
interface Pagination {
  nextPage: number,
  limit: number,
}

interface OrderBy {
  asc: string[],
  desc: string[],
}

interface QueryData {
  query?: object | string,
  select?: string,
  populate?: Populate[],
  pagination?: Pagination,
  orderBy?: OrderBy,
  findOne?: boolean,
}

export class FindObject extends BaseUtil{
  private _limit: number;
  private _select: string;
  private _skip: number;
  private _sort: object;
  private _collation: object;
  private _findOne: boolean;

  constructor(data: QueryData) {
    super();
    if (data.pagination) this.setPagination(data.pagination);
    if (data.orderBy) this.setOrderBy(data.orderBy);
    this.findOne = data.findOne ? data.findOne : false;
```

```
  this.query = data.query ? data.query : {};
  this.select = data.select;
  this.populate = data.populate;
};

public set query(query: object | string) {
  if (typeof query === "string") {
    this.id = query;
    this.findOne = false;
  } else {
    this._query = {...this._query, ...query};
  }
}

public get query(): object | string {
  return this._query;
}

private get sort(): object {
  return this._sort;
}

private set sort(sort: object) {
  this._sort = sort;
}

private get collation(): object {
  return this._collation;
}

private set collation(collation: object) {
  this._collation = collation;
```

```
}

private set limit(limit: number) {
  this._limit = limit;
}

private get limit(): number {
  return this._limit;
}

private set skip(skip: number) {
  this._skip = skip;
}

private get skip(): number {
  return this._skip;
}

private set select(select: string) {
  this._select = select;
}

private get select(): string {
  return this._select;
}

private set findOne(findOne: boolean){
  this._findOne = findOne;
}

private get findOne():boolean{
  return this._findOne;
```

```
  }

  /**
   *
   * @param {Pagination} pagination
   *
   * Configura a paginação.
   */
  private setPagination(pagination: Pagination) {
    this.skip = (pagination.nextPage - 1) * pagination.limit;
    this.limit = pagination.limit;
  }

  /**
   *
   * @param {OrderBy} orderBy
   *
   * Configura a order.
   */
  private setOrderBy(orderBy: OrderBy){
    let sortObj = {};
    for(let i = 0; i < orderBy.asc.length; i++){
      sortObj[orderBy.asc[i]] = 1;
    }
    for(let i = 0; i < orderBy.desc.length; i++){
      sortObj[orderBy.desc[i]] = -1;
    }
    this.sort = sortObj;
    this.collation = {locale: 'pt', strength: 2};
  }

}
```

```
import {BaseUtil} from './BaseUtil';

interface QueryData {
  query: object | string,
}

export class QueryObject extends BaseUtil {

  constructor(data: QueryData) {
    super();
    this.query = data.query;
  };

}

import {BaseUtil, Populate} from './BaseUtil';

interface UpdateData {
  query: object | string,
  update: any,
  new?: boolean,
  runValidators?: boolean,
  populate?: Populate[],
  select?: string[],
  options?: any,
}

export class UpdateObject extends BaseUtil {
  private _options: object;
  private _update: Object;
```

```
constructor(data: UpdateData) {
  super();
  this.query = data.query;
  this.update = data.update;
  this.populate = data.populate;
  this.newResponse = data.new ? data.new : true;
  this.runValidators = data.runValidators ? data.runValidators : true;
  this.options = data.options;
  this.select = data.select;
};

protected set query(query: object | string) {
  if (typeof query === "string") {
    this.id = query;
  } else {
    this._query = {...this._query, ...query};
    this.options = {multi: true};
  }
}

private set update(update: any) {
  this._update = this.handlerSetUpdate(update);
}

private get update() {
  return this._update;
}

private set options(options: any) {
  if(options)
    if(!this._options) this._options = options;
    else this._options = {...this._options, ...options};
```

```
}

private get options(): any {
  return this._options;
}

private set select(select: string[]) {
  if (Array.isArray(select) && select.length) {
    let fields = {
      select: {}
    };
    for (let i = 0; i < select.length; i++) {
      fields.select[select[i]] = 1;
    }
    this.options = fields;
  }
}

private set newResponse(newResponse){
  if(newResponse) this.options = {new: true};
}

private set runValidators(runValidators){
  if(runValidators) this.options = {runValidators: true};
}

/**
 *
 * @param {object} update
 * @returns {object}
 *
 * Make a handler to create a pattern of update param.
```

```
     */
    private handlerSetUpdate(update: object): object {
      this.beforeSetUpdate(update);
      let ret = {$set: update};
      if (update.hasOwnProperty("$inc") || update.hasOwnProperty("$addToSet")
|| update.hasOwnProperty("$pull") || update.hasOwnProperty("$push") ||
update.hasOwnProperty("$set") || update.hasOwnProperty("$unset"))
        return update;
      return ret;
    }

    /**
     *
     * @param update
     *
     * Remove the attribute id and _id, if the model pass in UpdateObject.
     */
    private beforeSetUpdate(update) {
      if (Array.isArray(update)) {
        for (let i = 0; i < update.length; i++) {
          delete update[i].id;
          delete update[i]._id;
        }
      } else {
        delete update.id;
        delete update._id;
      }
    }

  }
```