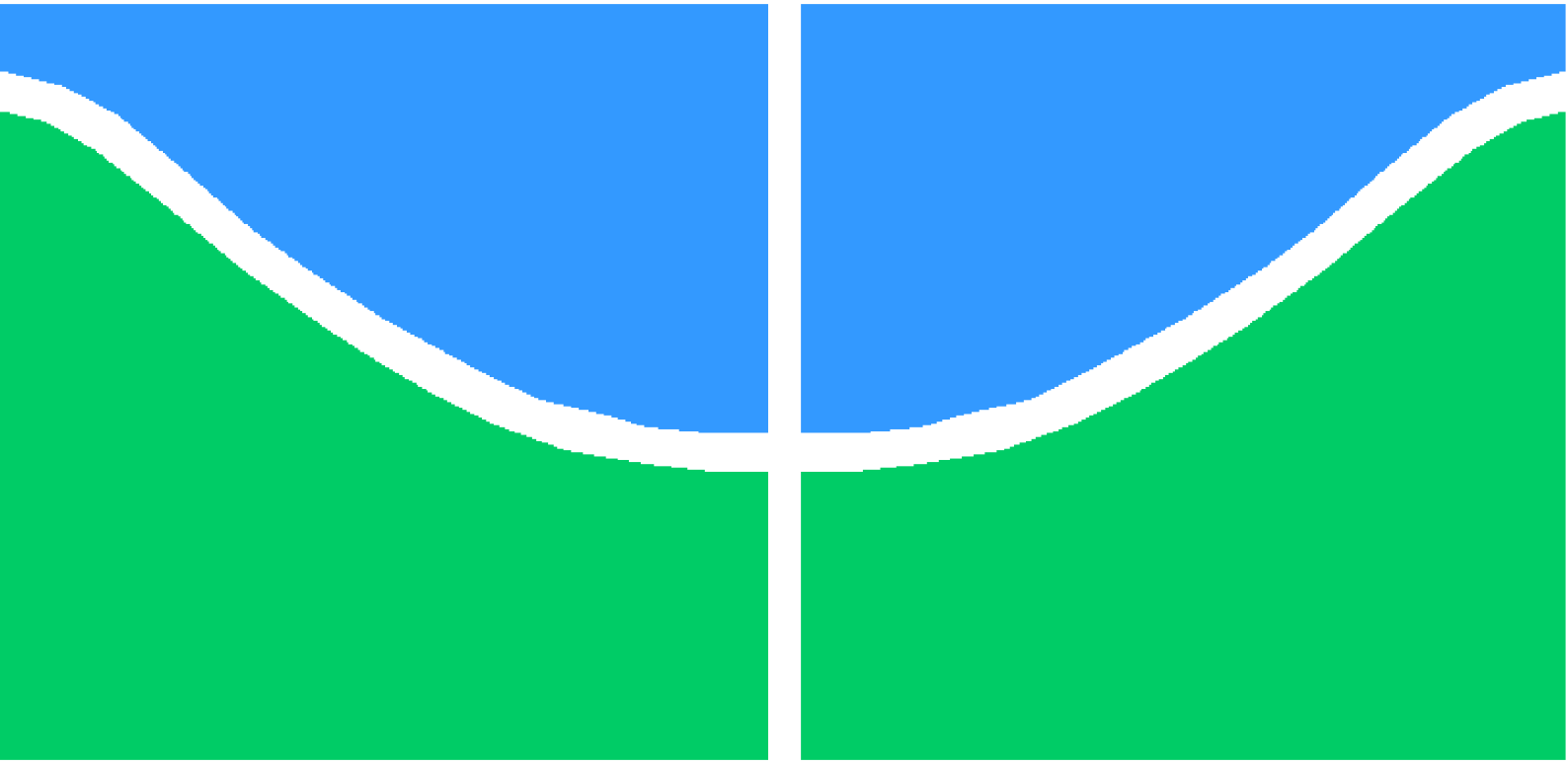

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Projeto Integrador Engenharia 2

UMISS - Unidade Móvel de Identificação de Saúde e Socorro

Projeto UMISS

Orientadores: Alex Reis, Luiz Laranjeira, Rhander Viana e Sebastièn Rondineau

Brasília, DF

2017

Afonso Delgado, Cesar Marques, Dylan Guedes, Felipe Assis, Gustavo Cavalcante, Johnson Andrade, Lucas Castro, Lunara Martins, Mariana Andrade, Nivaldo Lopo, Rafael Amado, Tiago Assunção, Wilton Rodrigues

UMISS - Unidade Móvel de Identificação de Saúde e Socorro

Relatório técnico referente à disciplina de Projeto Integrador 2, reunindo os cursos de Engenharias presentes no Campus Gama, da Universidade de Brasília.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Alex Reis, Luiz Laranjeira, Rhander Viana e Sebastièn Rondineau

Brasília, DF

2017

UMISS - Unidade Móvel de Identificação de Saúde e Socorro

Relatório técnico referente à disciplina de Projeto Integrador 2, reunindo os cursos de Engenharias presentes no Campus Gama, da Universidade de Brasília.

Brasília, DF
2017

Resumo

Pacientes com capacidade motora reduzida, em certo grau, necessitam de observação contínua a fim de evitar acidentes ou outros problemas. Além disso, em alguns casos, a presença de um cuidador é necessária para ajudar na movimentação da cadeira de rodas e na captura de sinais vitais. Tecnologias nesse campo não evoluem rápido o suficiente, não resolvem estes cenários ao mesmo tempo, e, mais ainda, são custosas. Neste trabalho nós apresentamos a UMISS, uma cadeira elétrica que extrai sinais vitais, notifica eventos críticos, e se move sem intervenção de terceiros. Com a UMISS nós esperamos criar uma solução de baixo custo, que permita ao paciente cuidar de si mesmo de maneira segura.

Palavras-chaves: cadeira de rodas. acessível. monitoramento. sensores.

Abstract

Handicapped people, in a certain degree, needs continuous monitoring in order to prevent accidents or other issues. Besides that, in some cases, the presence of a carer is needed to help with the wheelchair, and to track vital signals. Technologies in this field are not evolving fast enough, does not solve these scenarios at the same time, and, even more, are costly. In this work we present UMISS, an electric wheelchair that tracks vital signals, notifies critical events, and moves without third party intervention. With UMISS we expect to create a low cost solution, that allows the patient to securely take care of himself.

Key-words: wheelchair. accessible. monitoring. sensors

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

PSM	Processamento de Sinais e Monitoramento
CeA	Controle e Alimentação
PE	Projeto Estrutural
PC1	Ponto de controle 1
UMISS	Unidade Móvel de Identificação de Saúde e Socorro

Sumário

1 Introdução

Neste relatório apresentamos o andamento e a progressão do projeto UMISS que ocorreu entre os pontos de controle 1 e 2. Focaremos em questões mais práticas, e levantaremos os resultados obtidos e os esperados.

A organização se dará da seguinte forma: cada capítulo que segue será relativo a um subsistema do projeto. No Capítulo 2 apresentamos o andamento do subsistema de Processamento de Sinais e Monitoramento, que contempla o *middleware*, a aquisição de sinais, o servidor remoto (*backend*), o cliente *web* (*frontend*) e o aplicativo Android.

2 Processamento de Sinais e Monitoramento

2.1 Aquisição de Sinais

2.2 Middleware

O *middleware* do subsistema, uma Raspberry, tinha como resultados esperados uma aplicação que pudesse, ao rodar no embarcado, receber sinais e enviá-los de maneira correta ao servidor.

Os resultados esperados foram atingidos. Foi desenvolvido a aplicação Shoelace[1], que serve como abstração para a aquisição dos dados do conversor A/D e que envia os resultados para um servidor remoto.

Definimos que uma regra de negócio deveria ser que toda Raspberry tivesse um *token* e uma senha incluída, e esses dados são então utilizados nas requisições para os servidores. Isso foi feito através da geração de dados aleatórios (*token* e senha), que são obtidos sempre que a Raspberry é ligada, por estarem no *bash_rc*. Assim, todas as adições de sinais feitas por uma dada Raspberry já serão relacionados com o respectivo paciente, que poderá ter seus dados visualizados por parentes cadastrados.

A comunicação entre a Raspberry e o conversor é feita através do pacote em Python **Adafruit_ADS**, capaz de ler de até quatro canais ao mesmo tempo. Contudo, uma ressalva: os valores enviados pelo sensor de temperatura não são recebidos de uma maneira apresentável, por não estarem normalizados. Utilizamos então a equação de Steinhart-hart:

$$1/t = A + B * ln(R) + C[ln(R)]^3$$

Onde A, B e C são coeficientes, R é a resistência, e T a temperatura que desejamos apresentar. Utilizamos os seguintes valores para os coeficientes:

$$A = 0.001129148; B = 0.000234125; C = 0.0000000876741$$

Para o cálculo do *ln* utilizamos a função *log1p* do Python, e ressaltamos que tivemos diversos problemas de precisão, pois o Python arredonda os resultados das operações de maneira grosseira em diversas situações. Por fim, ajustamos outros parâmetros (como os utilizados na conversão da resistência) utilizando outros resultados como base, de maneira experimental.

[1] <https://github.com/cadeiracuidadora/shoelace>

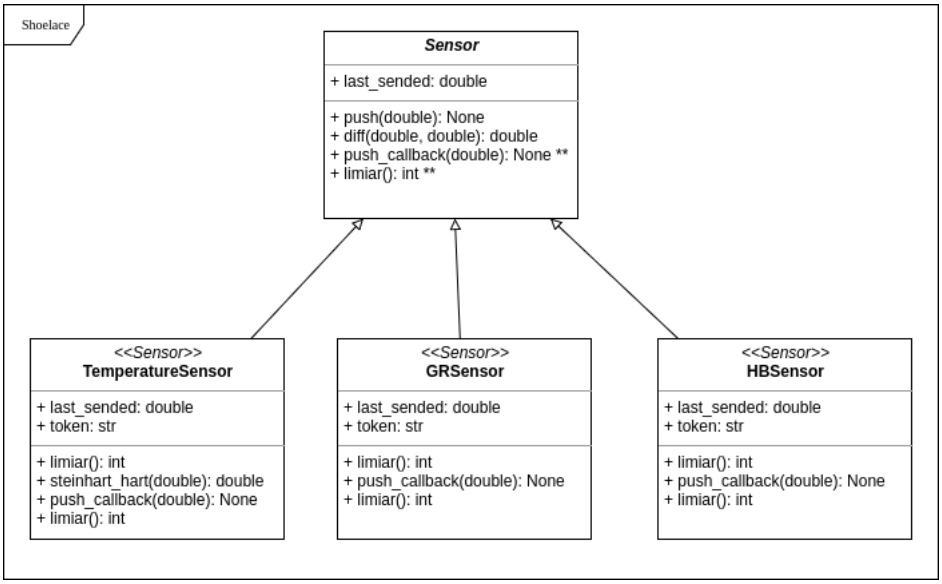

Figura 1: Diagrama de classes do Shoelace. Métodos marcados com ** são abstratos.

2.2.1 Arquitetura

A arquitetura do Shoelace foi feita pensando-se principalmente na **extensão**. Desenvolvemos então essa aplicação de modo que, caso novos sensores dos mais diversos tipos precisem ser utilizados, a adaptação no código da Raspberry é simples. Uma classe abstrata **Sensor** força a implementação do limiar referente ao sensor, e trás consigo funções que serão úteis, como o cálculo da diferença percentual entre o valor enviado para o servidor e o valor que está sendo analisado. A implementação de um novo sensor deve então sobrescrever somente dois métodos: (i) o método *limiar*, que diz qual a diferença mínima entre o último valor enviado e um novo valor para que seja enviado (útil na diminuição de sobrecarga do servidor, dificultando cenários de *backpressure*); e (ii) o método *push_callback*, que dá para o sensor a liberdade de decidir o que fazer quando o limiar definido for atingido. É nesse *callback* que fazemos a requisição no servidor remoto.

A Figura 1 apresenta a arquitetura geral do Shoelace. É ilustrado a implementação dos três sensores que utilizamos, mas caso novos sensores precisem ser usados, basta criar uma classe que herde de Sensor, e implementar os métodos necessários (*limiar* e *push_callback*).

2.3 API Django Rest Framework

Esta sessão tratará sobre o servidor do sistema, também conhecido como *backend* ou API Django. Na primeira subsessão, é tratada uma breve descrição sobre as funções e características deste. A segunda apresenta o *design* e a arquitetura de desenvolvimento. Por fim, trataremos as instâncias do projeto desenvolvido.

2.3.1 Papel da API

O backend do sistema, responsável por fazer o controle das requisições, atuando como intermediador à cadeira e aos seus respectivos monitores foi desenvolvida utilizando o *Django Rest Framework*. Este é uma referência no mundo de desenvolvimento *Python* para *API's Rest*.

O backend manipula todos os dados enviados pela cadeira, utilizando a *Rasp* executa o processamento destes dados e envia notificações para os *Smartphones* cadastrados no sistama. Além disso, serve dados para uma interface web de controle do monitor.

A manipulação dos dados é feita de forma segura, tratando a autenticação dos usuários que lidam com o sistema e redirecionando os dados de acordo com a sua autenticação. Para executar estes passos, o servidor conta com um poderoso sistema de autencação dos seus usuários, tanto para os pacientes que utilizam a cadeira, quanto para os monitores que manipulam as interfaces para cliente, como mobile e interface web. Os pacientes são cadastrados no sistema e enviam os sinais corporias para o servidor. O servidor, além de enviar uma notificação para o monitor respectivo, armazena este sinal em sua base de dados e retorna ao monitor quando solicitado. Os dados enviados ao monitor são apenas os sinais respectivos ao paciente que está sendo monitorado.

A ligação entre o paciente e o monitor pode ser feita no momento do cadastro ou em atualizações futuras. O paciente contém um identificador único em sua cadeira, que pode ser utilizada pelo monitor para supervisioná-lo. No momento que o monitor insere os dados no sistema, automaticamente serão ligados e o monitor terá acesso aos dados de sinais enviados pelo paciente.

2.3.2 *Design* e Arquitetura da Solução

O django utiliza de uma arquitetura própria para padronização e aplicação de boas práticas em desenvolvimento de API's. Esta arquitetura está dividida em camadas, que são responsáveis por funções específicas e cada uma tem seu papel definido para a manipulação de dados no sistema. A seguir está uma ilustração do funcionamento da arquitetura do servidor.

Como pode ser observado, existem três camadas principais, que são utilizadas para a manipulação dos dados. Elas são:

- Presentation Tier;
- Business Tier;
- Data Tier.

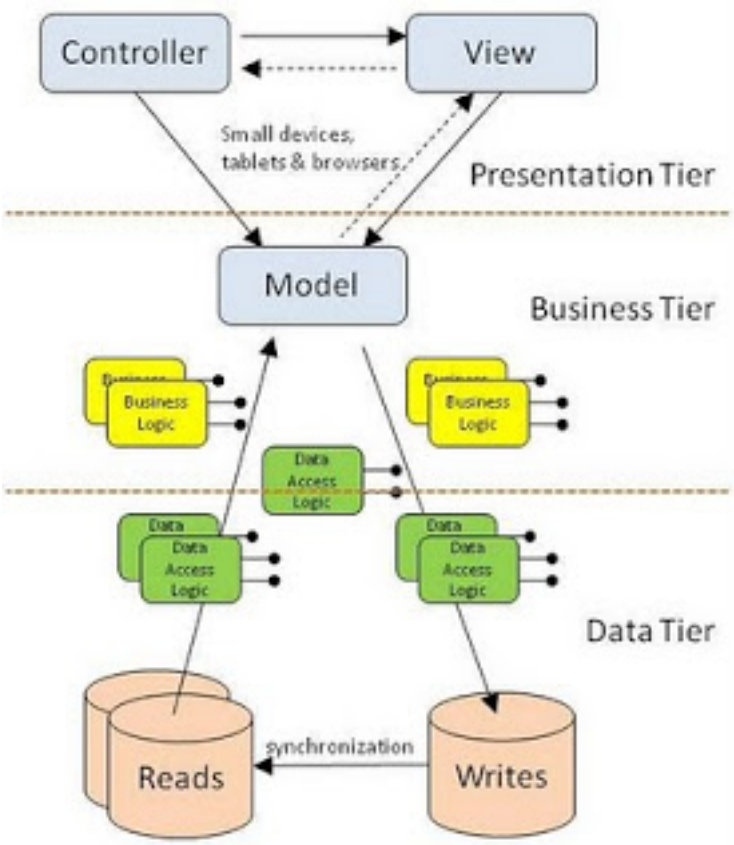

Figura 2: *Design* e Arquitetura do servidor Django

A primeira, *Presentation Tier* é resposável por ser a interface com as demais aplicações que irão utilizar os servidor. Essa camada é responsável por gerir as rotas do sistema, direcionando as urls solicitadas a funções específicas. Após mapeado nas funções, estas irão verificar as permissões de cada requisição, bem como cada usuário que está solicitando uma determinada ação.

As funções mapeadas na Tier passada utiliza de modelos preedefinidas na Tier de Business. Aqui são definidos os domínios da aplicação, bem como quais são as classes e negócio da aplicação. Nesta fase são definidos os tipos de usuários do sistema, as classes de sinais corporais do paciente e toda parte a de negócios da aplicação.

A última camada, chamada de *Data Tier* é responsável por lidar com a persistência dos dados de todos os usuários e sinais que são manipulados no sistema. Esta camada modela o bando de dados assim como foi definido na aplicação de negócios e aplica inserções neste a medida que novas requisições são feitas.

2.4 Módulo JavaScript EmberJS

O *frontend* do subsistema, módulo responsável pela visualização dos dados, tinha como resultados esperados uma aplicação que pudesse prover a visualização dos sinais referentes a um determinado paciente, baseado nas informações recebidas do servidor de *backend*. Devido à necessidade de uma certa dinamicidade na apresentação desses dados, foi escolhido o *Framework* JavaScript EmberJS[2]. Desta forma, foi então desenvolvida a aplicação UMISS-frontend[3], que apresenta de forma gráfica e textual o histórico dos sinais monitorados pelo projeto UMISS.

Baseado no *token* fornecido em cada cadeira, o usuário monitor pode fazer o cadastro na aplicação e assim fazer o monitoramento do paciente vinculado à cadeira em questão. O servidor de *backend* é responsável pela filtragem dos dados recebidos, garantindo então que apenas os dados do paciente vinculado ao token fornecido possam ser visualizados pelo usuário monitor.

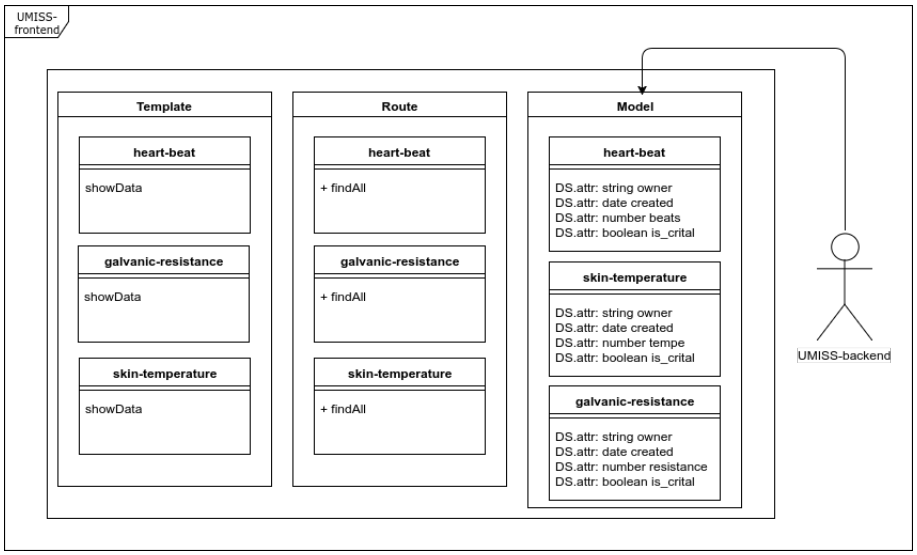

Figura 3: Diagrama da arquitetura parcial do UMISS-frontend.

2.5 Módulo Android

[2] <https://emberjs.com/>
[3] <https://github.com/cadeiracuidadora/UMISS-frontend>

Referências