



**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 Sébastien 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

# Listas de ilustrações

Figura 1 – Sensor para medição de temperatura corporal.	11
Figura 2 – Eletrodos fabricados para medição de umidade e resistência galvânica da pele.	11
Figura 3 – Circuito para captura de umidade da pele por GSR.	11
Figura 4 – Sensor de presença do usuário na cadeira de rodas.	12
Figura 5 – Circuito condicionador e de aquisição de sinais de eletrocardiograma.	13
Figura 6 – Conversor A/D ADS1115.	14
Figura 7 – Diagrama de classes do Shoelace. Métodos marcados com ** são abstratos.	15
Figura 8 – <i>Design</i> e Arquitetura do servidor Django (TECHNOLOGY, 2013)	17
Figura 9 – Diagrama da arquitetura parcial do UMISS-frontend.	19
Figura 10 – Diagrama de classes do módulo Android.	21
Figura 11 – <i>Design</i> e Arquitetura do servidor Django	27

## 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

<b>1</b>	<b>INTRODUÇÃO</b>	<b>9</b>
<b>2</b>	<b>PROCESSAMENTO DE SINAIS E MONITORAMENTO</b>	<b>10</b>
2.1	Aquisição e Condicionamento de Sinais	10
2.2	Middleware	14
2.2.1	Arquitetura	15
2.3	<b>API Django Rest Framework</b>	<b>16</b>
2.3.1	Papel da API	16
2.3.2	<i>Design</i> e Arquitetura da Solução	16
2.3.3	Projeto Desenvolvido	18
2.4	<b>Módulo JavaScript EmberJS</b>	<b>19</b>
2.5	<b>Módulo Android</b>	<b>19</b>
2.5.1	Funcionalidades	20
2.5.2	Arquitetura	20
<b>3</b>	<b>PLANO DE INTEGRAÇÃO</b>	<b>22</b>
3.1	Integração do subsistema Processamento de Sinais e Monitoramento	22
3.2	Cronograma	23
	<b>REFERÊNCIAS</b>	<b>24</b>
	<b>ANEXOS</b>	<b>25</b>
	<b>ANEXO A – DIAGRAMA DE CLASSES BACKEND</b>	<b>26</b>

# 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.

O objetivo desta etapa é desenvolver os subsistemas separadamente, levando em consideração a segurança e redundância de cada um, bem como a sua qualidade final. Estes são desenvolvidos em ambientes diferentes, porém, com o foco na integração de cada um, que será a próxima fase deste desenvolvimento. Desta maneira, este relatório trata sobre os resultados obtidos ao longo das semanas de implementação do ponto de controle 2.

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. Por fim, no Capítulo 3 traremos o nosso planejamento para a integração final dos diferentes subsistemas, e as considerações finais sobre a segunda parte do projeto.

## 2 Processamento de Sinais e Monitoramento

### 2.1 Aquisição e Condicionamento de Sinais

O projeto eletrônico do sistema de processamentos de sinais e monitoramento consiste em quatro módulos de sistemas de captura e condicionamento para os seguintes sinais: temperatura corporal e do ambiente, resistência galvânica da pele (GSR), acontecimento de quedas do usuário da cadeira de rodas e eletrocardiograma.

Para o circuito de monitoramento de sinais de temperatura, foi utilizado um sensor de medição linear de temperatura LM35, fabricado pela *Texas Instruments*. O sensor foi selecionado por sua variação linear de tensão de saída com a variação de temperatura, além do seu baixo consumo e alta precisão de medidas de temperatura com rápida resposta e sua faixa de captura de -55 à 150 graus Celsius, tornando-o viável para monitoramento de temperaturas corporais e ambientais. Tanto o circuito quanto o sistema embarcado foram projetados e dimensionados para tratar de sua variação linear de 10 mV em sua saída a cada 1 grau de variação na temperatura<sup>1</sup>.

Foram também realizados testes e algoritmos para medição da temperatura corporal com termistores NTC, porém, os mesmos mostraram-se menos acurados devido ao seu comportamento não linear dado pela equação de *Equação de Steinhart-Hart*<sup>2</sup> e problemas de arredondamento de valores capturados pelo sistema embarcado. Dessa forma, foi decidido o uso do sensor LM35.

O sensor fabricado para medição da temperatura corporal, apresentado na [1](#), é aplicado à superfície da pele, no antebraço, possibilitando leitura facilitada e de forma menos invasiva. Uma vez em contato com o corpo do usuário, o sensor apresenta alteração nos valores lineares de temperatura até o momento de estabilidade com a temperatura corporal.

O sistema de monitoramento de sinais de resistência galvânica da pele (GSR), tem como principal função o monitoramento de variações na umidade da pele do usuário, podendo indicar níveis perigosos de estresse ou até mesmo colapsos de hipoglicemias ou por fraqueza por falta de alimentação. O sistema consiste em dois eletrodos fabricados, como visto na [2](#), para contato com o antebraço do usuário, com uma distância fixa de 5 cm entre eles, dessa forma, mantendo os valores das medições confiáveis para cada usuário com diferentes tipos de pele.

Além disso, o circuito, visto na Figura [3](#) embarcado conta com um capacitor de

---

<sup>1</sup> <<http://www.ti.com/lit/ds/symlink/lm35.pdf>>

<sup>2</sup> <[lens.unifi.it/ew/dwl.php?dwl...mtyp=application/pdf](http://lens.unifi.it/ew/dwl.php?dwl...mtyp=application/pdf)>

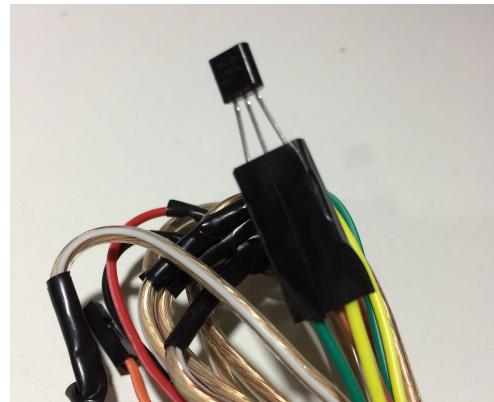


Figura 1 – Sensor para medição de temperatura corporal.

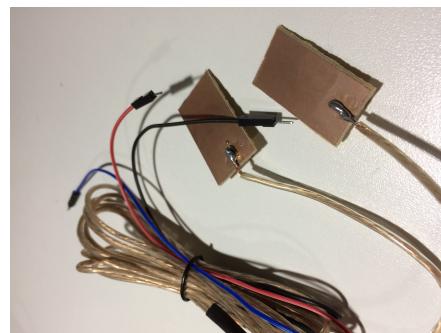


Figura 2 – Eletrodos fabricados para medição de umidade e resistência galvânica da pele.

acoplamento para o contato com a pele humana e um resistor de alta impedância para a medição da variação da resistência da pele por divisor de tensão. O sistema foi simplificado para que pudesse ser implementado junto aos cabos dos eletrodos, de forma a utilizar o mínimo de espaço possível na cadeira de rodas.



Figura 3 – Circuito para captura de umidade da pele por GSR.

O algoritmo para o monitoramento da resistência galvânica e umidade conta com comparadores para diferentes status do usuário, podendo alertar quando o usuário não está realizando a medição, quando apresenta características normais para sua pele ou quando

o usuário encontra-se em situação de risco. O sistema embarcado para o monitoramento e integração com o *middleware* foi programado em linguagem Python.

O sistema de alerta de quedas é responsável por enviar alertas para o servidor caso o usuário encontre-se fora da cadeira, indicando que o mesmo caiu da cadeira e necessita de auxílio urgente. Para esse sistema, apresentado na Figura 4 foi desenvolvido um circuito com um sistema de medição de distância e presença utilizando um módulo com sensor piezoelétrico. O sistema é responsável pelo monitoramento de variações de tensão no sensor, possibilitando o processamento dessa variação, alertando a presença ou não do usuário na cadeira de rodas. O sensor piezoelétrico é inserido no assento da cadeira para que o mesmo esteja pressionado durante todo o momento que o usuário esteja sentado na cadeira, e dessa forma, o algoritmo é capaz de alertar caso a presença do usuário na cadeira não seja identificada.



Figura 4 – Sensor de presença do usuário na cadeira de rodas.

O sistema de monitoramento dos sinais de eletrocardiografia, apresentado na ?? foi desenvolvido para captura dos sinais de pulso cardíaco a partir de eletrodos conectados ao antebraço do usuário, para que as medidas e a captura do sinal seja feita da forma menos invasiva possível. Para isso, o circuito deve ser capaz de capturar o sinal de ECG com precisão e assim, foi decidido pelo uso de um amplificador de instrumentação INA 128p, fabricado pela *Texas Instruments*<sup>3</sup>, principalmente por suas características de alta rejeição de CMRR, além de ganho de fácil regulagem. Além do amplificador de instrumentação, foram projetados filtros passa-baixa para rejeição de frequências acima de 100Hz que possam interferir no sinal de eletrocardiograma, que para monitoramento, variam entre baixas frequências de 0.5Hz e 40Hz<sup>4</sup>.

O projeto de captura dos sinais de eletrocardiograma utilizando o amplificador INA128 é dado a partir do ganho necessário para a visualização e processamento do sinal, com isso, o ganho do amplificador é dado pela equação:

<sup>3</sup> <<http://www.ti.com/lit/ds/symlink/ina129.pdf>>

<sup>4</sup> Yazicioglu RF, van Hoof C, Puers R. Biopotential Readout Circuits for Portable Acquisition Systems, 2009, Springer Science, ISBN: 978-1-4020-9092-9)

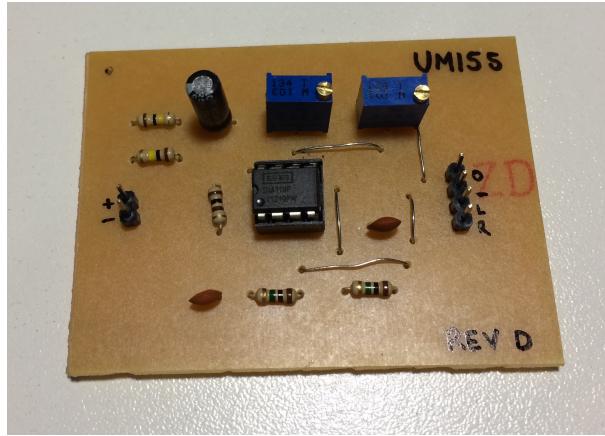


Figura 5 – Circuito condicionador e de aquisição de sinais de eletrocardiograma.

$$Av = 1 + \frac{50000}{R_G} \quad (2.1)$$

Sabe-se que a amplitude de sinais de ECG é baixa, na faixa de  $0.1\text{ mV}$  a  $5\text{ mV}$ . Com isso, foi implementado um resistor  $R_G = 100\omega$ , resultando em um ganho  $Av = 501$ . Dessa forma, o sinal terá uma amplitude de cerca de  $2.5\text{ V}$ .

Após testes, os primeiros sinais obtidos foram satisfatórios para monitoramento de frequência cardíaca. Dessa forma, após o condicionamento, os sinais são enviados para um microcontrolador Arduino, que realiza a amostragem e envia os dados de forma Serial para o processador principal presente na Raspberry Pi, onde os dados são capturados pela porta USB, e dispostos em um *array*, alterado a cada período de tempo. Para teste de visualização destes dados, os mesmos foram plotados no sistema da Raspberry Pi utilizando a biblioteca *matplotlib*<sup>5</sup>.

Após a captura e condicionamento de todos os sinais analógicos, os mesmos são direcionados para um módulo de conversão Analógico-Digital (A/D). O conversor selecionado foi o ADS1115<sup>6</sup>, devido a sua alta resolução de 16 bits, capaz de converter valores analógicos em valores digitais de 0 a 65535, e por sua taxa de amostragem de 800Hz, suficientes para a captura dos sinais a serem monitorados para o projeto. O conversor é conectado com a Raspberry Pi via protocolo I2C, possibilitando o uso de menos portas para o envio de até 4 sinais simultâneos.

Uma vez com os sinais devidamente convertidos para digital, o sistema embarcado e de *middleware* é responsável por monitorar os sinais, realizando verificações de seus valores a partir de calibrações realizadas para cada um dos módulos.

<sup>5</sup> <<https://matplotlib.org/>>

<sup>6</sup> <<https://cdn-shop.adafruit.com/datasheets/ads1115.pdf>>

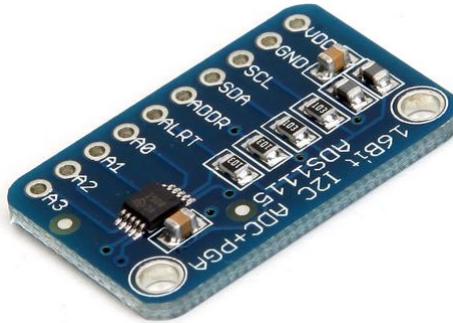


Figura 6 – Conversor A/D ADS1115.

## 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*<sup>7</sup>, 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

<sup>7</sup> <<https://github.com/cadeiracuidadora/shoelace>>

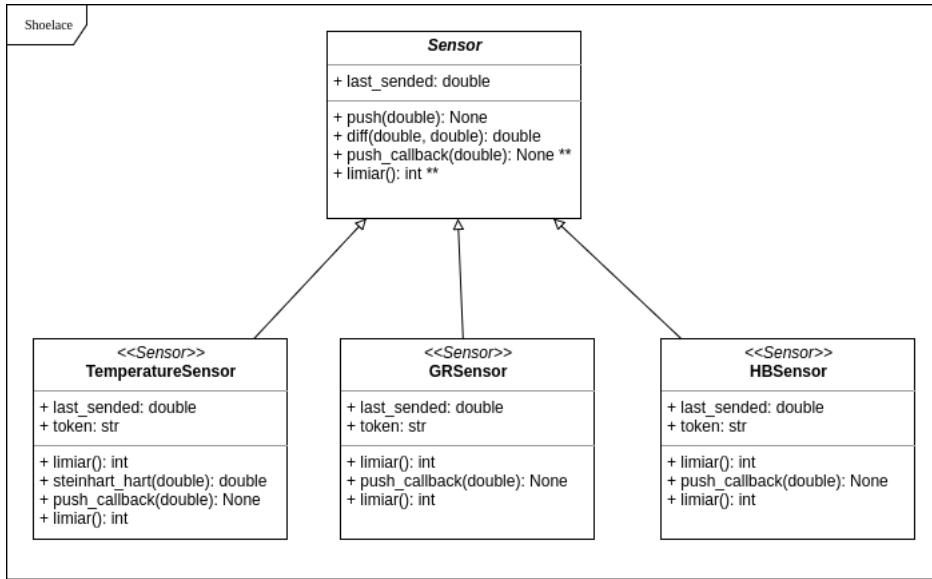


Figura 7 – Diagrama de classes do Shoelace. Métodos marcados com \*\* são abstratos.

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.

### 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 7 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 sistema. 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 autenticação 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 corporais 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 utilizado pelo monitor para supervisioná-lo. No momento que o monitor insere os dados no sistema, automaticamente será ligado ao paciente e 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.

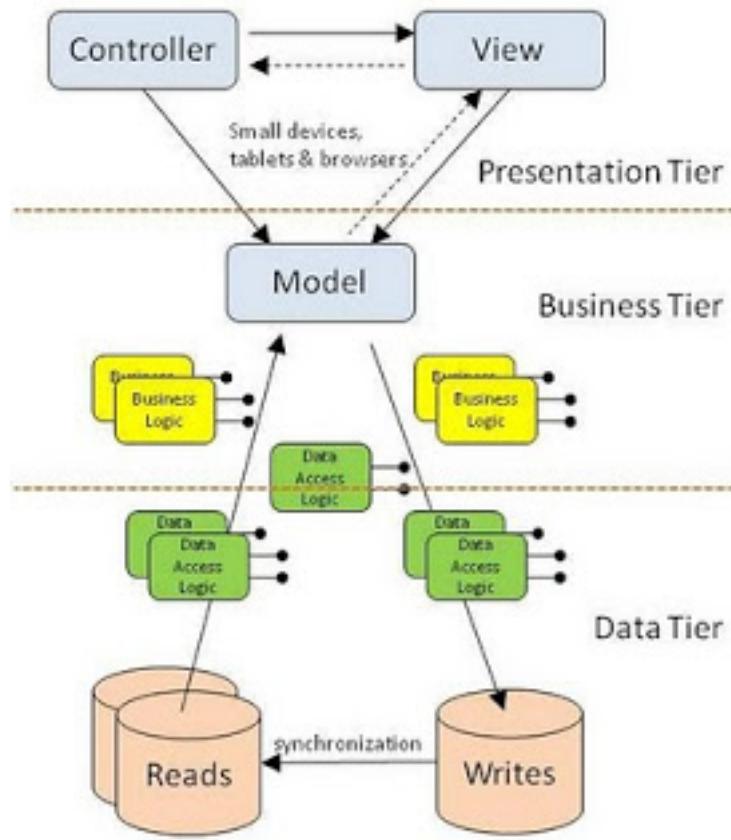


Figura 8 – *Design e Arquitetura do servidor Django* ([TECHNOLOGY, 2013](#))

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

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

A primeira, *Presentation Tier* é responsável por ser a interface com as demais aplicações que irão utilizar o 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.

Além disso, nesta camada, existe uma transformação importante para o consumo de dados por parte dos clientes. Esta é conhecida como serialização dos dados, tanto para quem envia, quanto para quem solicita. Sua responsabilidade é aplicar um modelo de dados formalizado na comunidade, conhecido como Json. Dessa forma, todo tipo de dado

recebido, assim como os enviados são transformados a partir dos objetos python. Esta fase é aplicada logo após a aplicação das rotas para as respectivas ações.

As funções mapeadas na Tier passada utiliza de modelos predefinidos 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 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.

### 2.3.3 Projeto Desenvolvido

O projeto UMISS-BackEnd, desenvolvido exatamente na arquitetura apresentada na sessão [2.3.2](#), possui entidades que deram formato à Business Tier do projeto, onde são definidas as propriedades do sistema, bem como ele atuará.

No projeto, foi desenvolvido o diagrama de classes, que auxilia na atividade de compreender o domínio do projeto. Este está no Anexo [A](#).

É possível visualizar que existem várias entidades de usuários. Isto foi desenvolvido pois existem dois tipos de usuários diferentes no sistema. Eles são o Paciente e o Monitor. Estes possuem atributos e comportamentos diferentes, gerando assim a necessidade de implementação de dois domínios diferentes.

Da mesma maneira, existem três tipos de sinais corporais que são enviados pelo paciente. Esses sinais possuem características diferentes, bem como atributos. Dessa maneira, existe uma entidade generalista chamada *BodySignal* com características comuns de todas. E foram criados as demais especializações com suas características específicas. Elas são:

- Sinais de temperatura corporal;
- Sinais de batimento cardíaco;
- Sinais de resistência galvânica.

Por fim, existem ligações dos usuários com os sinais corporais, sendo que os pacientes são donos destes. Assim como existem pacientes que possuem monitores ligados, gerando um relacionamento cíclico entre as entidades de usuário.

## 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<sup>8</sup>. Desta forma, foi então desenvolvida a aplicação UMISS-frontend<sup>9</sup>, 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.

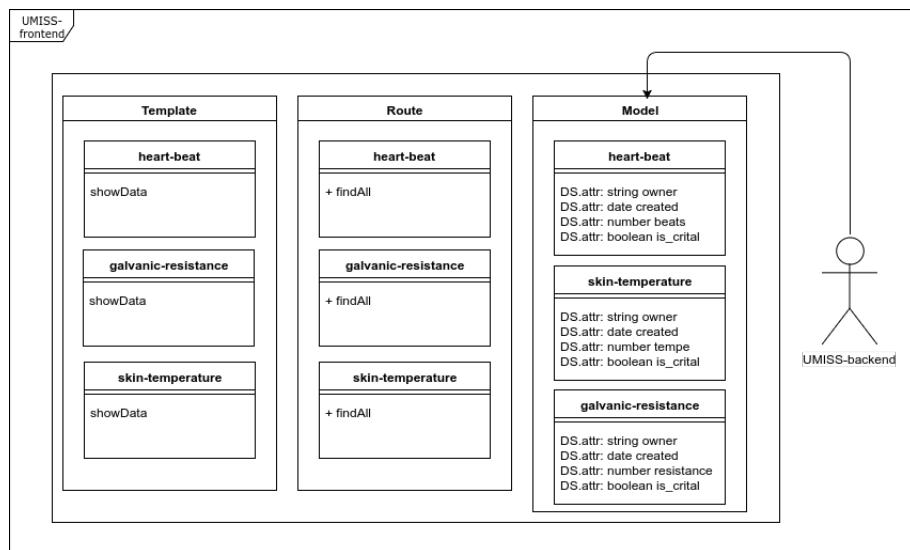


Figura 9 – Diagrama da arquitetura parcial do UMISS-frontend.

## 2.5 Módulo Android

O módulo Android do sistema é responsável pela visualização dos dados referentes a um determinado paciente e também pela notificação do usuário monitor via *push notification*.

Este módulo atua como cliente, ou seja, os dados referentes a um paciente estão armazenados no *backend* e a aplicação Android somente consome esses dados. O único

<sup>8</sup> <<https://emberjs.com/>>

<sup>9</sup> <<https://github.com/cadeiracuidadora/UMISS-frontend>>

valor que é armazenado na aplicação Android é um *token* de autenticação enviado do servidor quando um usuário monitor efetua o *login*.

### 2.5.1 Funcionalidades

- *Login*:

É possível fazer login com um usuário do tipo monitor na aplicação, ao fazê-lo, é recebido um *token* de autenticação do servidor que é armazenado no Android. Essa etapa se difere um pouco do *frontend* pois no Android ocorre um passo adicional de enviar um identificador para o *backend* ser capaz de enviar as notificações para o Android.

- Cadastro:

Caso o usuário do tipo monitor ainda não tenha feito um cadastro no *frontend*, é possível fazer o cadastro direto da aplicação Android, basta informar os dados de *login*, senha e o identificador da cadeira.

- Notificações:

Caso o *backend* receba um dado crítico do *shoelace*, é enviado ao aplicativo Android uma notificação do tipo *push notification* para que o usuário seja alertado que há algo errado acontecendo com o paciente.

- Gráficos:

Os dados referentes à temperatura, batimentos cardíacos e resistência galvânica são exibidos em forma de gráfico, toda vez que o usuário abrir a aplicação uma nova requisição será feita ao servidor e os gráficos serão criados.

### 2.5.2 Arquitetura

A arquitetura do módulo Android foi elaborada da seguinte maneira, foram criados dois pacotes, um chamado *com.umiss* que contém todas as *fragments* e *activities* da aplicação e um pacote chamado *network* que contém as classes responsáveis pelas requisições e notificações.

A *MainActivity* é a classe principal do sistema, ao iniciar, é executado o método *isLogged* que verifica se existe um *token* de autenticação válido, caso exista, será instanciado as *fragments* *HeartBeatsFragment*, *GalvanicFragment* e *TemperatureFragment* que são responsáveis pela criação dos gráficos de batimentos cardíacos, temperatura e resistência galvânica respectivamente. Caso contrário, será instanciado a *activity* de *Login*, a partir da *activity* de *Login* é possível abrir a *activity* de Cadastro.

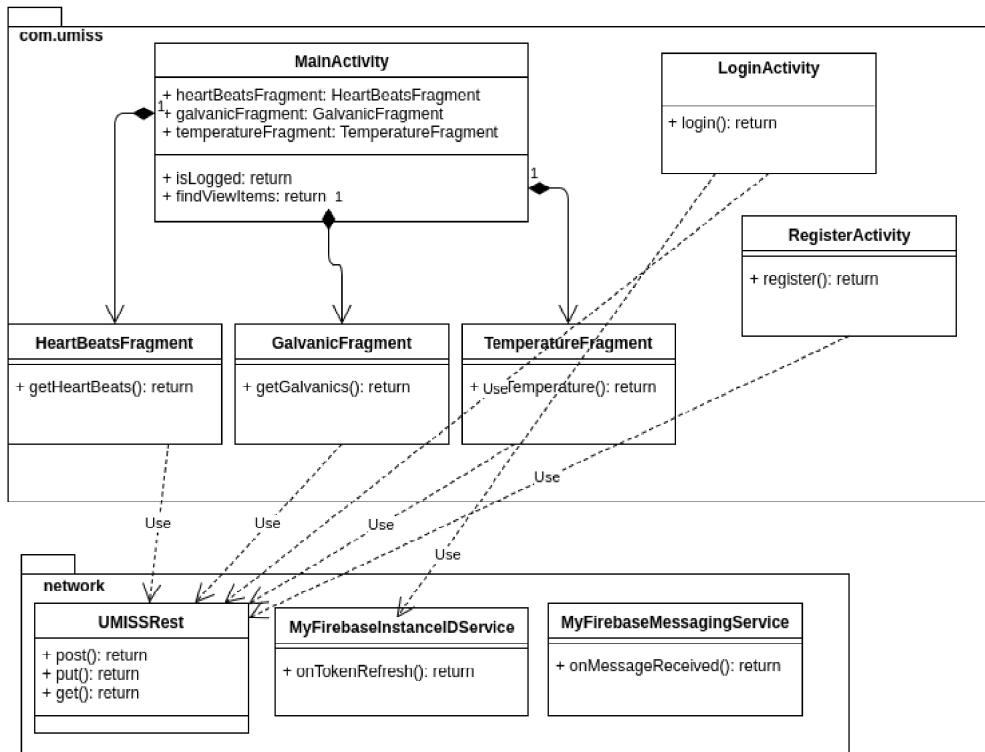


Figura 10 – Diagrama de classes do módulo Android.

O pacote `network` contém a classe `UMISSRest` que é responsável pelos métodos http *POST*, *GET* e *PUT*. Todas as classes que realizam alguma requisição com o *backend* utilizam os métodos estáticos da classe `UMISSRest`. As notificações são feitas pelo serviço Firebase Cloud Messaging através das classes `MyFirebaseMessagingService` que é responsável por criar uma notificação e pela classe `MyFirebaseInstanceIdService` que cria um identificador que é enviado ao *backend*.

# 3 Plano de Integração

Neste capítulo apresentamos nosso planejamento para a integração dos diferentes subsistemas, que deve ocorrer na terceira etapa do projeto. Adaptamos os pontos levantados no guia do PMBOK, adequando o plano ao contexto da disciplina, e focando na integração de subsistemas de um projeto de engenharia, e não da área de gestão de pessoas.

A integração dos subsistemas ocorrerá no tempo das aulas, contudo, caso necessário, em outros horários o grupo se reunirá de forma a conseguir a integração completa. Ainda, ressaltamos que o subsistema de Processamento de Sinais e Monitoramento não é fortemente acoplado aos outros subsistemas, de modo que a integração entre ele e o restante do projeto será fácil e não carecerá de um grande esforço. Os outros dois subsistemas, Projeto Estrutural e Controle e Alimentação, por outro lado, são bem acoplados, e um maior cuidado será tomado a respeito desses dois subsistemas.

## 3.1 Integração do subsistema Processamento de Sinais e Monitoramento

Como mencionado, o subsistema de Processamento de Sinais e Monitoramento não deverá apresentar grandes problemas na integração, principalmente por não ser acoplado aos outros subsistemas. Os únicos componentes físicos desse subsistema são os componentes eletrônicos relacionados a aquisição de sinais (sensores, amplificadores, filtros e conversores) e o sistema embarcado (Raspberry Pi). Os outros componentes (servidor remoto, cliente *frontend* e *mobile*) estão na nuvem, e não causam impacto na integração com os outros componentes físicos.

É esperado que os componentes físicos desse subsistema sejam alocados em um compartimento de fácil manutenção, para que seja facilmente manuseado durante os diversos testes. Além disso, esse compartimento deve disponibilizar saídas para os fios, que serão então conectados a outros subsistemas, ou disponibilizados para serem utilizados pelo paciente. Assim, a integração deve ocorrer da seguinte forma:

1. Alocação dos componentes eletrônicos de modo seguro, mas que ocupe o menor espaço possível, pois os outros subsistemas carecem de bastante espaço;
2. Acoplamento do compartimento na parte inferior da cadeira (nos braços), parafusando-o (incluindo o *case* da Raspberry Pi);

3. Extensão e disponibilização dos cabos e dos sensores, para que sejam facilmente utilizados pelos pacientes;
4. Extensão da bateria conectada a Raspberry Pi, e conexão entre ela e a bateria do subsistema de Controle e Alimentação.

## 3.2 Cronograma

Tabela 1 – Cronograma para a integração dos subsistemas.

Atividade	Responsável	Deadline
Criar compartimento com sensores e embarcado	Afonso, Dylan, Gustavo, Tiago e Wilton	08/06
Acoplamento do compoartimento na parte inferior da cadeira	Dylan e Afonso	08/06
Extensão e disponibilização dos cabos	Gustavo, Tiago e Wilton	08/06
Extensão da bateria da Raspberry Pi	Dylan e Afonso	08/06
Teste do subsistema de Processamento e Monitoramento	Afonso, Dylan, Gustavo, Tiago e Wilton	08/06

## Referências

TECHNOLOGY, C. *Arquitetura do servidor Django*. [S.l.], 2013. Disponível em: <<http://criticaltechnology.blogspot.com.br/2011/09/mvc-in-three-tier-architecture.html>>. Citado 2 vezes nas páginas 5 e 17.

## Anexos

## ANEXO A – Diagrama de Classes Backend

Diagrama de classes do backend na figura 11, desenvolvido no projeto UMISS.

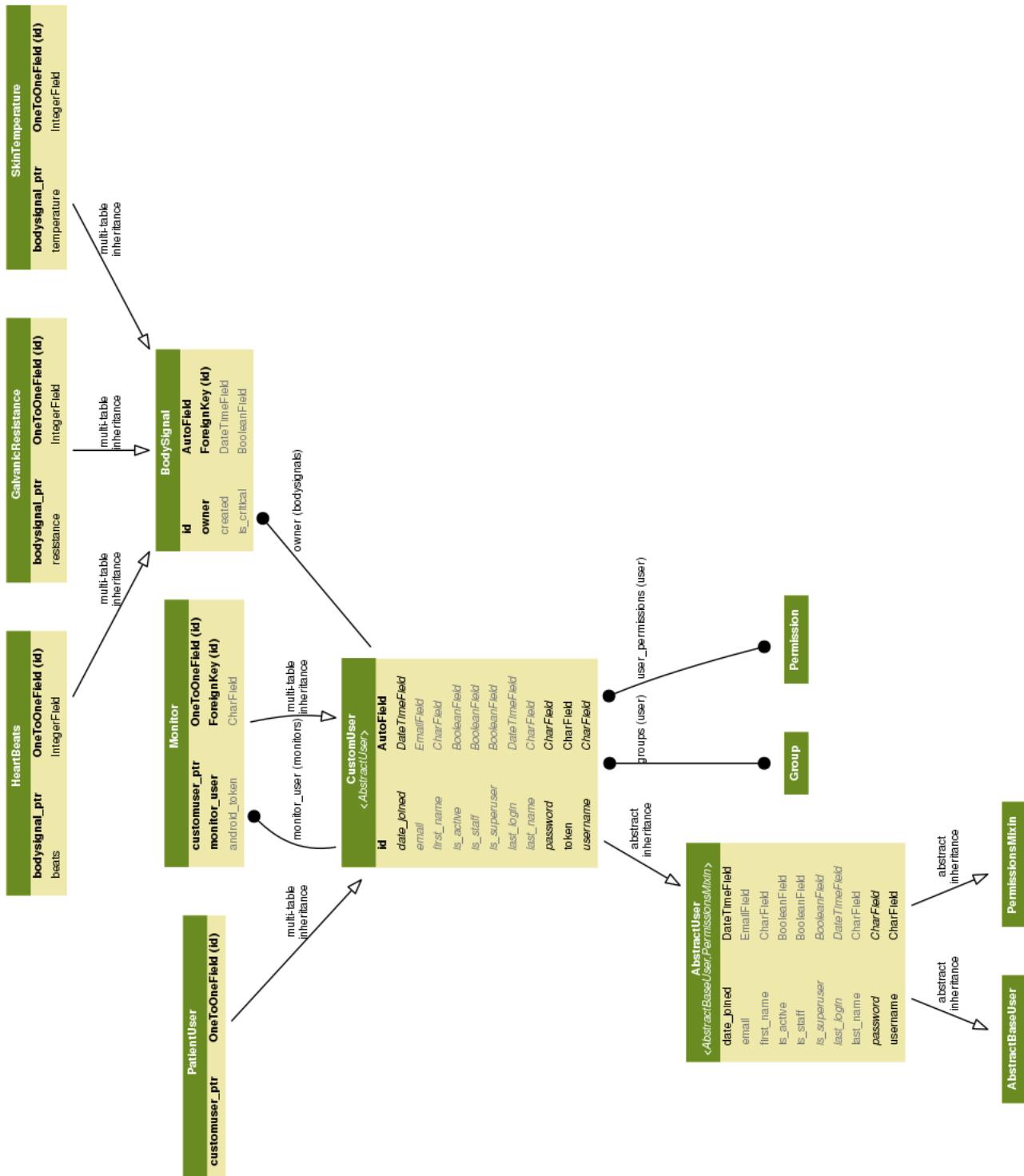


Figura 11 – Design e Arquitetura do servidor Django