



Universidade do Minho
Escola de Engenharia

MESTRADO INTEGRADO EM ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA

LABORATÓRIOS DE TELECOMUNICAÇÕES E INFORMÁTICA II

SISTEMA DE MONITORIZAÇÃO DE ATIVIDADE FÍSICA FASE C

Grupo 2:

David José Ressurreição Alves - A79625

José Pedro Afonso Rocha - A70020

Luís Pedro Lobo de Araújo - A73232

Guimarães, 28 de Junho de 2019

Índice

1	Introdução	3
2	Planeamento	4
2.1	Planeamento temporal	4
2.2	Ferramentas utilizadas	5
3	Síntese do projeto	6
4	Arquitetura do sistema	7
4.1	Esquema geral	7
4.2	Sistema Gestor de Serviço	7
4.3	Sistema Central	8
4.4	Browser	8
4.5	Base de dados central	9
4.6	Protocolo de comunicação entre Gestor de Serviço e Sistema Central	9
4.6.1	Recursos da API	10
5	Requisitos	12
5.1	Requisitos funcionais	12
5.2	Requisitos não funcionais	12
6	Implementação	13
6.1	Front-end	13
6.2	Back-end	29
6.2.1	Classe: <i>dataHandler.js</i>	30
6.2.2	Classe: <i>server.js</i>	33
6.2.3	Base de Dados Relacional: <i>bd_central.sql</i>	34
7	Conclusão	37
8	Referências	38

Índice de Imagens

1	Diagrama com a planeamento temporal da fase C.	4
2	Arquitetura do sistema para a Fase C.	7
3	Mockup da UI.	8
4	Arquitetura da Base de Dados relacional.	9
5	Visualização da página inicial do Sistema Central.	13
6	Visualização da página de <i>login</i> do Sistema Central.	13
7	Visualização da página de registo do Sistema Central.	14
8	Visualização da página <i>home</i> do utilizador “Admin”.	15
9	Visualização da página para adicionar paciente.	17
10	Visualização da página de editar pacientes.	18
11	Visualização da página de listagem de áreas.	20
12	Visualização da página de adicionar áreas.	21
13	Visualização da página de editar áreas.	22
14	Visualização da página de listar os utilizadores.	24
15	Visualização da página de adicionar utilizadores.	25
16	Visualização da página de editar utilizadores.	26
17	Visualização da página de listar os utilizadores.	29

1. Introdução

Na unidade curricular de Laboratórios de Telecomunicações e Informática II, foi-nos proposto realizar um projeto que consiste na criação de um sistema de monitorização de atividade física para doentes internados numa instituição de saúde ou de apoio social.

É de extrema importância a integração da tecnologia com os cuidados de saúde pois permite uma maior atenção aos pacientes de uma instituição de saúde e a obtenção de dados em tempo real.

Assim, maximiza-se e melhora-se as prestações de cuidados que os médicos e enfermeiros realizam obtendo, por exemplo, se não for possível estar em acompanhamento pessoal e contínuo com o doente, todos os dados relativos desse mesmo doente em qualquer lugar, sendo alertados para qualquer problema com rapidez e com a devida urgência, se necessário.


Este sistema global terá que ter um conjunto de sistemas críticos, que serão: dispositivos concentradores de dados obtidos, dispositivos sensores atuadores, servidor web e de base de dados e uma aplicação web que tenha uma interface para interação com o utilizador.

Nesta terceira fase (Fase C), o foco será o desenvolvimento de um Sistema Central que irá gerir toda a informação de uma Base de Dados relacional que foi criada na fase anterior e na qual irão ser acrescentados mais campos para disponibilização de informação mais completa e precisa para um utilizador deste sistema. Este sistema vai conter uma interface gráfica que irá fornecer várias funcionalidades ao utilizador e que terá um mecanismo de autenticação para a atribuição de tipos de acesso diferenciados.

2. Planeamento

Nesta secção encontra-se disponível a planificação temporal, do nosso grupo para esta fase C do projeto, bem como o conjunto de ferramentas que serão utilizadas neste projeto.

2.1. Planeamento temporal



Nome	Data de início	Data de fim
☐ • FASE C	06-05-2019	02-06-2019
• Compreensão de conceitos-chave	06-05-2019	06-05-2019
• Introdução ao relatório da Fase C	06-05-2019	08-05-2019
• Desenvolvimento da Especificação da Fase C	08-05-2019	11-05-2019
• Conclusão da Especificação da Fase C	11-05-2019	12-05-2019
• Entrega da Especificação da Fase C	13-05-2019	13-05-2019
☐ • Desenvolvimento de código (Base de Dados relacional)	13-05-2019	16-05-2019
• Preparação da BD relacional	13-05-2019	16-05-2019
• Normalização da BD relacional	13-05-2019	16-05-2019
☐ • Desenvolvimento de código (Sistema Central)	17-05-2019	26-05-2019
• Elaboração da base do sistema	17-05-2019	21-05-2019
• Desenvolvimento inicial da UI do sistema	22-05-2019	26-05-2019
• Desenvolvimento do WebService	22-05-2019	26-05-2019
• Configuração do protocolo HTTP	22-05-2019	26-05-2019
• Configuração da porta HTTP	22-05-2019	26-05-2019
☐ • Conclusão de código (Base de Dados Relacional)	27-05-2019	29-05-2019
• Integração da BD relacional no Sistema Central	27-05-2019	29-05-2019
• Execução de testes na BD	27-05-2019	29-05-2019
☐ • Conclusão de código (Sistema Central)	27-05-2019	01-06-2019
• Desenvolvimento final da UI do Sistema	27-05-2019	30-05-2019
• Desenvolvimento final do WebService	27-05-2019	30-05-2019
• Testes de comunicações com Gestor e Browser	30-05-2019	01-06-2019
• Interpretação dos dados recebidos e armazenados na BD ...	30-05-2019	01-06-2019
• Testes Finais	01-06-2019	02-06-2019
• Conclusão do Relatório da Fase C	01-06-2019	02-06-2019
• Entrega da Fase C	01-06-2019	02-06-2019

Figura 1: Diagrama com a planeamento temporal da fase C.

2.2. Ferramentas utilizadas

As ferramentas utilizadas serão as seguintes:

- Programa **GanttProject** para planeamento temporal das tarefas do grupo;
- Programa **Arduino IDE**, para editar, compilar e enviar código para a placa Arduino;
- Programa **MySQL**, para criação e gestão de bases de dados;
- Programa **GnuPlot**, para elaboração de gráficos em tempo real;
- Programa **Visual Studio Code**, para editar e compilar código;
- Programa **Visual Paradigm**, para elaboração de diagramas.
- Programa **Adobe Photoshop**, para elaboração de logótipos;
- Plataforma **Slack**, para comunicação entre os membros do grupo;
- Plataforma **GitHub**, para partilha e organização do código desenvolvido pelo grupo.
- Plataforma **Google Drive**, para partilha de ficheiros entre os membros do grupo.
- Plataforma **OverLeaf**, para elaboração de relatórios em LaTeX.
- Plataforma **Vectary**, para elaboração modelos em 3D.
- Plataforma **Fluid UI**, para elaboração da *mockup*.
- Plataforma **Postman**, para testes à API usada.

3. Síntese do projeto

Na sua globalidade, este sistema irá conter um conjunto de dispositivos e sistemas críticos para o desenvolvimento deste projeto, sendo estes: sensores, servidores, base de dados, dispositivos de comunicação e microcontroladores.

Nesta fase iremos focar o desenvolvimento no Sistema Central, que irá integrar a comunicação e interpretação dos dados recebidos pelos gestor de serviço e irá permitir a visualização de todos os dados tratados para um utilizador que esteja a usar o serviço através de interface gráfica que se irá desenvolver e que poderá ser acedida através de um *browser* [1].

A comunicação entre Gestor de Serviço e Sistema Central é feita via serviço *web*, sendo que o fluxo de dados entre eles deve ser bidireccional, onde do sistema central para os gestores de serviço deve conter dados acerca dos pacientes e dos próprios serviços e dos gestores para o sistema central deve conter dados acerca do comportamento físico e identificativos de um ou mais pacientes.

4. Arquitetura do sistema

4.1. Esquema geral

Para esta fase C a arquitetura do nosso sistema pode ser visualizada na seguinte figura:

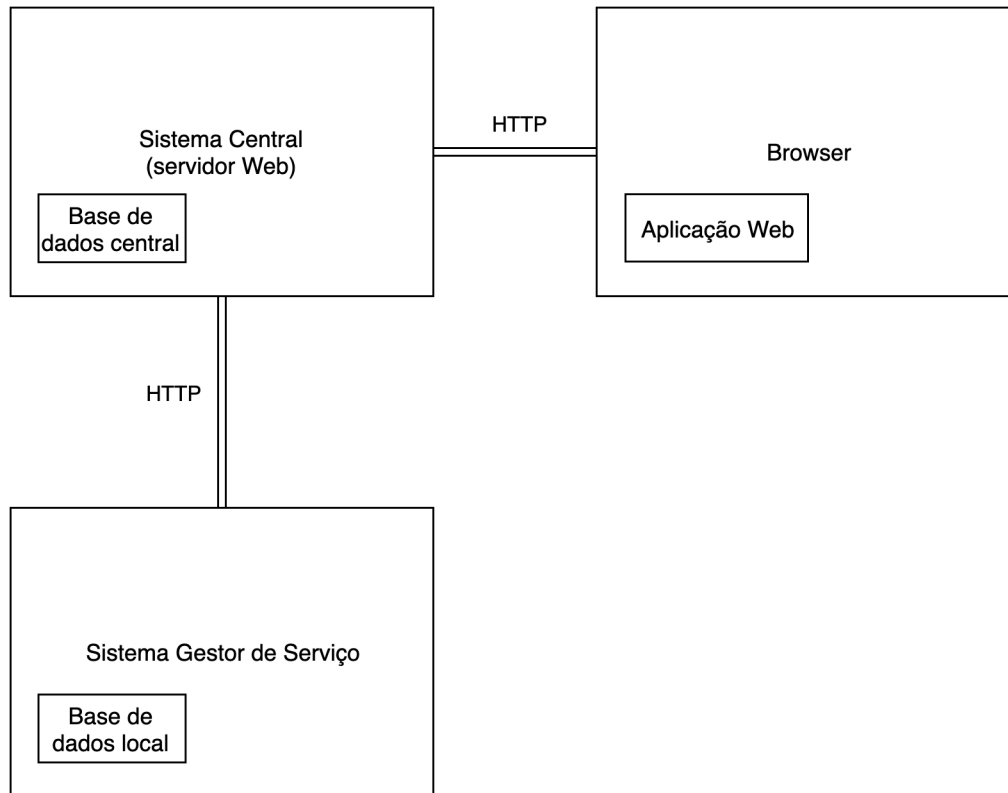


Figura 2: Arquitetura do sistema para a Fase C.

4.2. Sistema Gestor de Serviço

O sistema gestor de serviço serve para monitorizar os dados recolhidos pelos concentradores dos vários sujeitos, guardando os dados recebidos numa base de dados local MySQL.

A comunicação do Sistema Gestor de Serviço com o Sistema Central, faz-se utilizando métodos HTTP, na qual é enviada a informação interpretada pelo sistema, como por exemplo, o estado do sujeito (PARADO,QUEDA,AGITADO,ANDAMENTO).

4.3. Sistema Central

O grande foco desta fase C é a criação de um sistema central que seja capaz de gerir e armazenar toda a informação que recebe do gestor de serviço. Este sistema será acedido por vários utilizadores através de um *Web Browser* e a comunicação entre eles será feita através de métodos *HTTP*.

Na interface de utilizador (UI) do Sistema Central será possível fazer *login* com várias contas de utilizador e de seguida visualizar os vários estados dos sujeitos que se pretendem analisar. Pretende-se ainda implementar várias estatísticas tais como, a percentagem de tempo que um sujeito passou em cada estado, num intervalo de tempo definido pelo utilizador.

Poderão ainda ser acrescentadas funcionalidades extra que o grupo ache pertinente no contexto do projeto.

4.4. Browser

Através de um *browser* podemos aceder às páginas HTML do servidor web desenvolvido para o Sistema Central, podendo assim observar de uma forma intuitiva e remota os estados dos vários sujeitos analisados. O grupo preparou umas *mockups* que servem de preparação inicial para a interface gráfica que deverá ser implementada e que são fieis à visualização final que o utilizador vai experienciar:

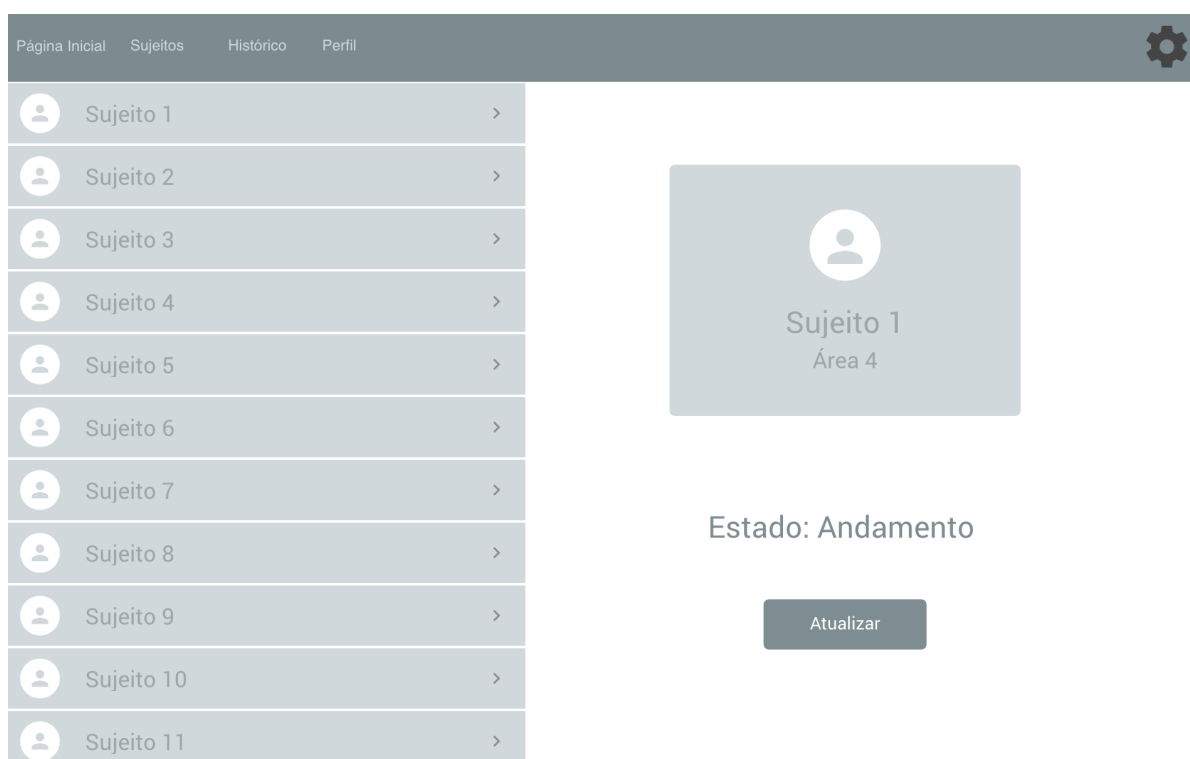


Figura 3: Mockup da UI.

4.5. Base de dados central

A base de dados relacional presente no Sistema Central tem a seguinte constituição final:

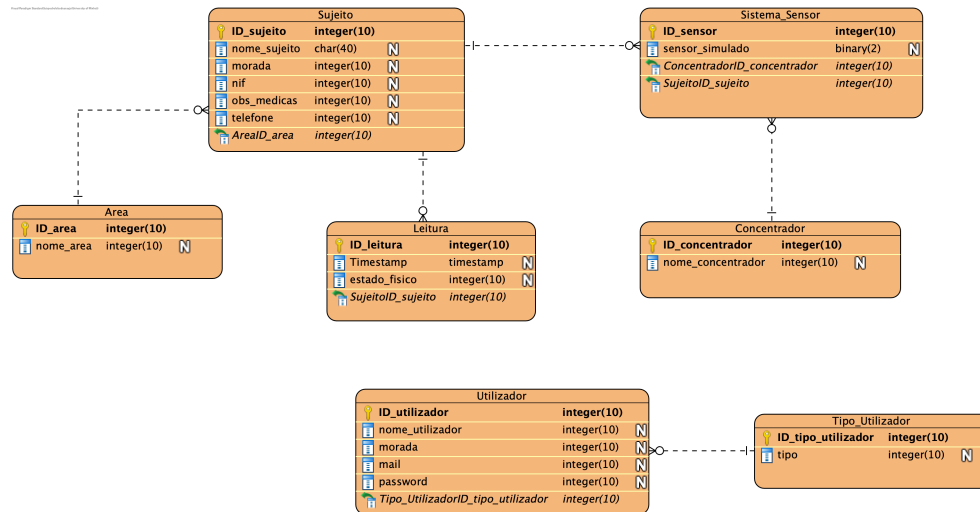


Figura 4: Arquitetura da Base de Dados relacional.

O objectivo desta constituição é que um paciente esteja sempre relacionado com o ou os sistemas sensores que esteja a usar, com a área onde ele se encontra e com a leitura do seu estado/comportamento físico. O concentrador agrega essas relações todas pois para um concentrador podemos ter nenhum ou vários sistemas sensores associados e para esses sistemas sensores podemos ter um sujeito associado que por sua vez terá uma área e leitura do seu comportamento associada.

Para a parte do utilizador, que terá um acesso limitado aos dados que poderá visualizar, será relacionado um tipo que consoante o seu registo e autenticação será capaz de ver apenas o que lhe é permitido para esse tipo.

4.6. Protocolo de comunicação entre Gestor de Serviço e Sistema Central

A comunicação entre gestores de serviço e o serviço central é absolutamente crucial para o sistema que irá ser implementado, uma vez que é a base de funcionamento do sistema central, pois sem comunicação entre eles, os utilizadores não dispõem de informação sobre os sujeitos a serem monitorizados.

Para a implementação desta comunicação o grupo decidiu utilizar uma arquitetura *Rest API*, uma vez que esta permite a interoperabilidade entre vários sistemas computacionais, e a utilização de métodos HTTP facilita a comunicação em terminais baseados na Web.

4.6.1. Recursos da API

Na tabela seguinte é possível visualizar os recursos disponíveis pela API:

Tabela 1: Recursos da API

URI	Método HTTP	Descrição	Parâmetros
/login	POST	Efetua o login	-utilizador -password
/logout	GET	Efetua logout	
/sujeito	GET	Obtém todos os sujeitos	
/sujeito/:id	GET	Obtém sujeito através do id	
/sujeito	POST	Insere novo sujeito	-nome -morada -nif -obs_medicas -telefone -id_area
/sujeito/:id	PUT	Atualiza sujeito	-nome -morada -nif -obs_medicas -telefone -id_area
/sujeito/:id	DELETE	Remove sujeito	
/area	GET	Obtém todas as áreas	
/area/:id	GET	Obtém área através do id	
/area	POST	Insere nova área	-nome
/area/:id	PUT	Atualiza área	-nome
/area/:id	DELETE	Remove área	
/sistema_sensor	GET	Obtém todos os sistemas sensores	
/sistema_sensor/:id	GET	Obtém sistema sensor através do id	
/sistema_sensor	POST	Insere novo sistema sensor	-sensor_simulado -id_concentrador -id_sujeito
/sistema_sensor/:id	PUT	Atualiza sistema sensor	-sensor_simulado -id_concentrador -id_sujeito
/sistema_sensor/:id	DELETE	Remove sistema sensor	
/leitura	GET	Obtém todas as leitura	
/leitura/:id	GET	Obtém leitura através do id	
/leitura	POST	Insere nova leitura	-timestamp -estado_fisico -id_sujeito
/leitura/:id_leitura	PUT	Atualiza leitura	-timestamp -estado_fisico -id_sujeito
/leitura/:id_leitura	DELETE	Remove leitura	

Tabela 2: Recursos da API (cont.)

URI	Método HTTP	Descrição	Parâmetros
/concentrador	GET	Obtém todos os concentradores	
/concentrador/:id	GET	Obtém concentrador através do id	
/concentrador	POST	Insere novo concentrador	-nome
/concentrador/:id	PUT	Atualiza concentrador	-nome
/concentrador/:id	DELETE	Remove concentrador	
/utilizador	GET	Obtém todos os utilizadores	
/utilizador/:id	GET	Obtém utilizador através do id	
/utilizador	POST	Insere novo utilizador	-nome -mail -password -morada -id_tipo
/utilizador/:id	PUT	Atualiza utilizador	-nome -mail -password -morada -id_tipo
/utilizador/:id	DELETE	Remove utilizador	
/tipo_utilizador	GET	Obtém todos os tipos de utilizadores	
/tipo_utilizador/:id	GET	Obtém tipo de utilizador através do id	
/tipo_utilizador	POST	Insere novo tipo de utilizador	-tipo
/tipo_utilizador/:id	PUT	Atualiza tipo de utilizador	-tipo
/tipo_utilizador/:id	DELETE	Remove tipo de utilizador	

5. Requisitos

5.1. Requisitos funcionais

Para o correto funcionamento do sistema num todo, é necessário que os seguintes requisitos sejam cumpridos:

- Captação comunicação dos dados do gestor de serviço para o sistema central;
- Correta apresentação dos comportamentos físico dos sujeitos;
- Base de dados relacional bem estruturada e que armazene corretamente todos os dados obtidos pelo sistema central;
- Uma interface de acordo com o objetivo deste sistema e que satisfaça os requisitos do cliente;

5.2. Requisitos não funcionais

Estes requisitos da fase C que serão abaixo enumerados são não funcionais o que indica que não estão diretamente ligados com as funcionalidades do sistema e estão mais relacionados como o tempo de resposta e a fiabilidade do sistema. Os requisitos não funcionais definidos são:

- Tempo reduzido de apresentação da condição do sujeito baixo;
- Fiabilidade da comunicação entre os dispositivos concentradores e os dispositivos sensores;
- Fiabilidade da comunicação entre o gestor de serviço e sistema central;
- Fiabilidade da comunicação entre o sistema central e o *browser*;
- Fiabilidade dos dados recolhidos pelos sistemas sensores;
- Interface de utilizador (UI) responsiva e de fácil utilização para os utilizadores.

6. Implementação

6.1. Front-end

Inicialmente começamos por construir a interface gráfica do nosso Sistema Central, as quais passámos a apresentar nas seguintes figuras.

Na figura 5, é possível observar a página inicial do Sistema Central, na qual é possível observar uma breve apresentação do serviço, bem como a possibilidade de se fazer *login*.



Figura 5: Visualização da página inicial do Sistema Central.

Na figura 6, pode-se observar a página de *login* na qual um utilizador com uma conta previamente criada poderá aceder aos serviços disponibilizados no Sistema Central.

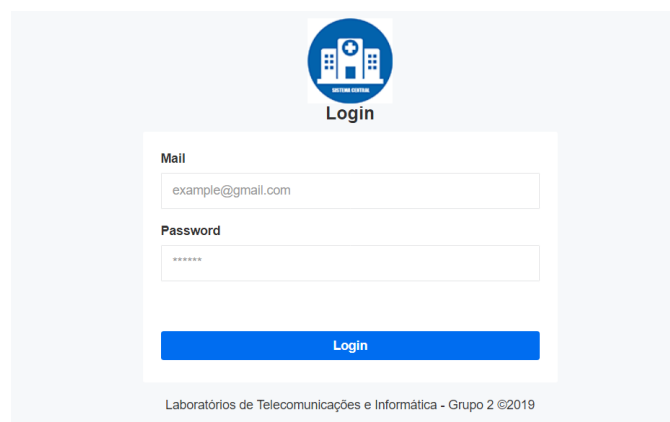


Figura 6: Visualização da página de *login* do Sistema Central.

A implementação correspondente a esta página foi a seguinte:

```
// var BASE_URL = 'http://localhost:8080/';
$(document).ready(function(){
  jQuery.support.cors = true;
  $('#login_btn').click(function (){
    var user = $('#username_login').val();
    var psw = $('#password_login').val();
    var token = '';
    if(user == '' || psw == ''){
      alert("Preencha todos os dados");
    }
    else{
      $.ajax({
        url: localStorage.getItem('base_url')+"login",
        type: 'POST',
        dataType: 'json',
        contentType: "application/json; charset=utf-8",
        crossDomain: true,
        data: JSON.stringify({"mail":user,"password":psw}),
        success:function(data){
          console.log(data);
          if(data.data.auth == true ){
            localStorage.setItem('token',data.data.token);
            console.log("localStorage: "+localStorage.getItem('token'));
            // alert(data.data);
            // salto para pagina de admin se for admin
            if(data.data.id_tipoUtilizador == 1){
              window.location.href = "admin.html";
            }
          }
          else{
            alert("Enganou-se no username e/ou password, tente novamente");
          }
        },
        error: function (xhr, ajaxOptions, thrownError) {
          console.log(JSON.stringify(xhr));
          console.log(JSON.stringify(thrownError));
          alert("Enganou-se no username e/ou password, tente novamente");
        }
      });
    }
  });
});
```

Na figura 7, pode-se observar a página onde um utilizador pode ser registado pelo "Admin", sendo esta a única via de registo de utilizadores.

Figura 7: Visualização da página de registo do Sistema Central.

A implementação correspondente ao registo do utilizador foi o seguinte:

```
// var BASE_URL = 'http://localhost:8080/';
$(document).ready(function(){
  jQuery.support.cors = true;
  $('#register_btn').click(function (){
    var user = $('#username').val();
    var morada = $('#morada').val();
    var pass = $('#password').val();
    var mail = $('#mail').val();
    var tipo = $('#tipo_user').val();
    if(user == '' || morada == '' || pass == '' || mail == '' || tipo == ''){
      alert("Preencha todos os dados.");
      alert(""+user+" "+morada+" "+pass+" "+mail+" "+tipo);
    }
    else{
      $.ajax({
        url: localStorage.getItem('base_url')+"register",
        type: 'POST',
        dataType: 'json',
        contentType: "application/json; charset=utf-8",
        crossDomain: true,
        data:
          JSON.stringify({"username":user,"morada":morada,"mail":mail,"password":pass,"id_tipo":tipo}),
        success:function(data){
          console.log(data);
          localStorage.setItem('token',data.data.token);
          console.log("localStorage: "+localStorage.getItem('token'));
          alert("Registado com sucesso.");
          // salto para proxima pagina
          if(tipo == 1){
            window.location.href = "admin.html";
          }
        },
        error: function (xhr, ajaxOptions, thrownError) {
          console.log(JSON.stringify(xhr));
          console.log(JSON.stringify(thrownError));
          alert("No foi possivel registar.");
        }
      });
    }
  });
});
```

Após o registo ou *login* com sucesso, cada utilizador pode observar a sua página *home* respetiva, como mostra a figura 8, neste caso para o utilizador "Admin".

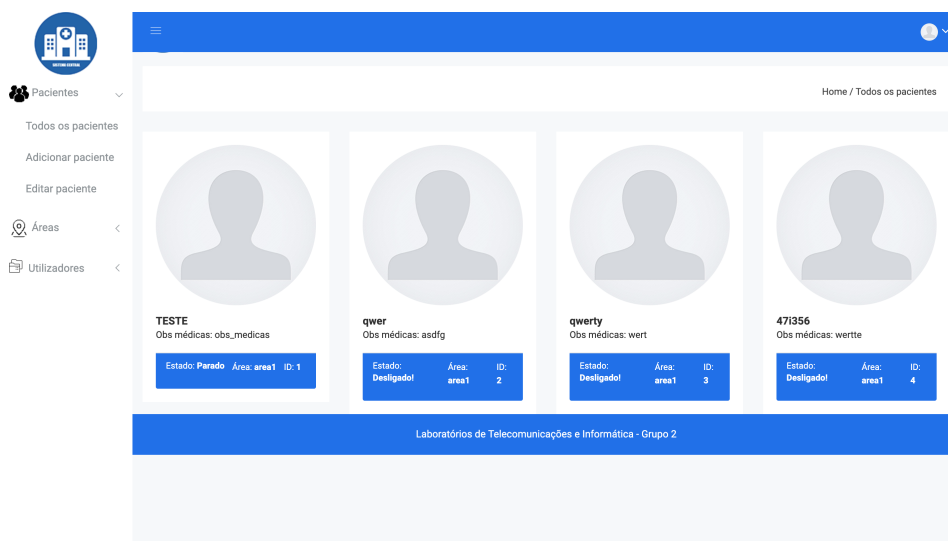


Figura 8: Visualização da página *home* do utilizador "Admin".

Para implementar esta página foi construído o seguinte código:

```

var info;
var j=0;
$(document).ready(function () {
    jQuery.support.cors = true;
    $.ajax({
        url: localStorage.getItem("base_url") + "info/pacientes",
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            info=data.data;
            console.log(info);
            for(i=0;i<info.length;i++){
                $.ajax({
                    url: localStorage.getItem("base_url") + "last/leituras/"+info[i].id_paciente,
                    type: 'GET',
                    contentType: "application/json; charset=utf-8",
                    dataType: "json",
                    beforeSend: function (xhr) {
                        xhr.setRequestHeader("Authorization", "Bearer " +
                            localStorage.getItem('token'));
                    },
                    data: {},
                    success: function (data2) {
                        console.log("Data2:"+JSON.stringify(data2.data));
                        setEstado(data2);
                    },
                    error: function (xhr, ajaxOptions, thrownError) {
                        alert(xhr.status);
                        alert(thrownError);
                    }
                })
            }
        },
        error: function (xhr, ajaxOptions, thrownError) {
            alert(xhr.status);
            alert(thrownError);
        }
    });
});
function setEstado(estado){
    for(i=0;i<info.length;i++){
        if(info[i].id_paciente == estado.data[0].id_paciente){
            info[i].estado_fisico = estado.data[0].estado_fisico;
            appendPacienteInfo(i);
        }
    }
}
function appendPacienteInfo(i) {
    $("##pacientes").append(
        "<div class=\"col-lg-3 col-md-6 col-sm-6 col-xs-12\">"+
        "<div class=\"hpanel hblue contact-panel contact-panel-cs responsive-mg-b-30\">"+
        "<div class=\"panel-body custom-panel-jw\">"+
        "<img alt=\"logo\" class=\"img-circle m-b\" src=\"default-user.png\">"+
        "<h3>"+info[i].nome_paciente+"</h3>"+
        "<p>Obs mdicas: "+info[i].obs_medicas+"</p>"+
        "</div>"+
        "<div class=\"panel-footer contact-footer\">"+
        "<div class=\"professor-stds-int\">"+
        "<div class=\"professor-stds\">"+
        "<div class=\"contact-stat\"><span>"+
        "<p>Estado: "+
        "</span> <strong>"+info[i].estado_fisico+"</strong></div>"+
        "</div>"+
        "<span style=\"display:inline-block; width: 0.3cm;\"></span>"+
        "<div class=\"professor-stds\">"+
        "<div class=\"contact-stat\"><span>rea: </span> <strong>"+
        "+info[i].nome_area+"</strong></div>"+
        "</div>"+
        "<span style=\"display:inline-block; width: 0.3cm;\"></span>"+
        "<div class=\"professor-stds\">"+
        "<div class=\"contact-stat\"><span>ID: </span>

```

```

    <strong>"+info[i].id_paciente+"</strong></div>"+
  "</div>"+
  "</div>"+
  "</div>"+
  "</div>"
}
)

```

O "Admin" poderá ter controlo total sobre o Sistema Central, e na figura 9 é possível observar a página onde o "Admin", pode adicionar pacientes.

Figura 9: Visualização da página para adicionar paciente.

De modo a cumprir este procedimento, foi implementado o seguinte código:

```

$(document).ready(function(){
  jQuery.support.cors = true;
  // obter todas as areas e inserir no template
  $('#submit').click(function (){
    var nome = $('#nome').val();
    var morada = $('#morada').val();
    var nif = $('#nif').val();
    var contacto = $('#contacto').val();
    var obs_medicas = $('#obs_medicas').val();
    var area = $('#area').val();
    if(nome=='' || morada=='' || nif=='' || contacto=='' || obs_medicas=='' || area==''){
      alert("Preencha todos os dados.");
    }
    else{
      $.ajax({
        url: localStorage.getItem('base_url')+"pacientes",
        type: 'POST',
        dataType: 'json',
        contentType: "application/json; charset=utf-8",
        crossDomain: true,
        beforeSend: function (xhr) {
          xhr.setRequestHeader("Authorization", "Bearer " +
            localStorage.getItem('token'));
        },
        data: JSON.stringify({"nome_paciente": nome,"morada": morada,"nif":
          nif,"obs_medicas": obs_medicas,"telefone": contacto,"id_area": area}),
        success:function(data){
          console.log(data);
        },
        error: function (xhr, ajaxOptions, thrownError) {
          console.log(JSON.stringify(xhr));
          console.log(JSON.stringify(thrownError));
        }
      });
    }
  });
}

```

```

    alert("No foi possvel adicionar paciente.");
  });
}
});
});

```

O administrador poderá editar dados relativos a pacientes existentes, como se pode ver na figura 10.

Figura 10: Visualização da página de editar pacientes.

Para ser possível esta funcionalidade, foi preciso a seguinte implementação:

```

$(document).ready(function () {
  jQuery.support.cors = true;
  // obter todas as areas e inserir no template
  $.ajax({
    url: localStorage.getItem("base_url") + "areas",
    type: 'GET',
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    beforeSend: function (xhr) {
      xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
    },
    data: {},
    success: function (data) {
      appendAreaInfo(data.data);
      console.log(JSON.stringify(data.data));
    },
    error: function (xhr, ajaxOptions, thrownError) {
      console.log(xhr.status);
      console.log(thrownError);
    }
  });
  function appendAreaInfo(areas) {
    for (i = 0; i < areas.length; i++) {
      $("#area").append(
        "<option value=\"" + areas[i].id_area + "\">" + areas[i].id_area + " - " +
          areas[i].nome_area + "</option>" + "</select>");
    }
  }
  // obter todas os pacientes e inserir no template
  $.ajax({
    url: localStorage.getItem("base_url") + "pacientes",
    type: 'GET',
  });

```

```

contentType: "application/json; charset=utf-8",
dataType: "json",
beforeSend: function (xhr) {
    xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
},
data: {},
success: function (data) {
    appendPaciente(data.data);
    console.log(JSON.stringify(data.data));
},
error: function (xhr, ajaxOptions, thrownError) {
    console.log(xhr.status);
    console.log(thrownError);
}
});
function appendPaciente(pac) {
    for (i = 0; i < pac.length; i++) {
        $("#select").append(
            "<option value=\"" + pac[i].id_paciente + "\"" + ">" + pac[i].id_paciente + " - " +
            pac[i].nome_paciente + "</option>" + "</select>");
    }
}
// carregar dados para paciente escolhido
$('#select').change(function () {
    id = $('#select').val();
    console.log("selected: " + id);
    $.ajax({
        url: localStorage.getItem("base_url") + "pacientes/" + id,
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            console.log(JSON.stringify(data.data));
            $('#nome').val(data.data[0].nome_paciente);
            $('#morada').val(data.data[0].morada);
            $('#nif').val(data.data[0].nif);
            $('#contacto').val(data.data[0].telefone);
            $('#obs_medicas').val(data.data[0].obs_medicas);
            $('#area').val(data.data[0].id_area);
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(xhr.status);
            console.log(thrownError);
        }
    });
});
$('#submit').click(function () {
    var id = $('#select').val();
    var nome = $('#nome').val();
    var morada = $('#morada').val();
    var nif = $('#nif').val();
    var contacto = $('#contacto').val();
    var obs_medicas = $('#obs_medicas').val();
    var area = $('#area').val();
    if (nome == '' || morada == '' || nif == '' || contacto == '' || obs_medicas == '' ||
        area == '' || id == '') {
        alert("Preencha todos os dados.");
    }
    else {
        $.ajax({
            url: localStorage.getItem('base_url') + "pacientes/" + id,
            type: 'PUT',
            dataType: 'json',
            contentType: "application/json; charset=utf-8",
            crossDomain: true,
            beforeSend: function (xhr) {
                xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
            },
            data: JSON.stringify({ "nome_paciente": nome, "morada": morada, "nif": nif,
                "obs_medicas": obs_medicas, "telefone": contacto, "id_area": area }),
            success: function (data) {
                console.log(data);
                window.location.href = "admin.html";
            },
            error: function (xhr, ajaxOptions, thrownError) {

```

```

        console.log(JSON.stringify(xhr));
        console.log(JSON.stringify(throwError));
        alert("No foi possvel editar o paciente.");
    });
}
});
});

```

O utilizador "Admin" também poderá ver todas as áreas, adicionar áreas e editar áreas, como se pode ver nas figuras 11, 12 e 13, respetivamente.

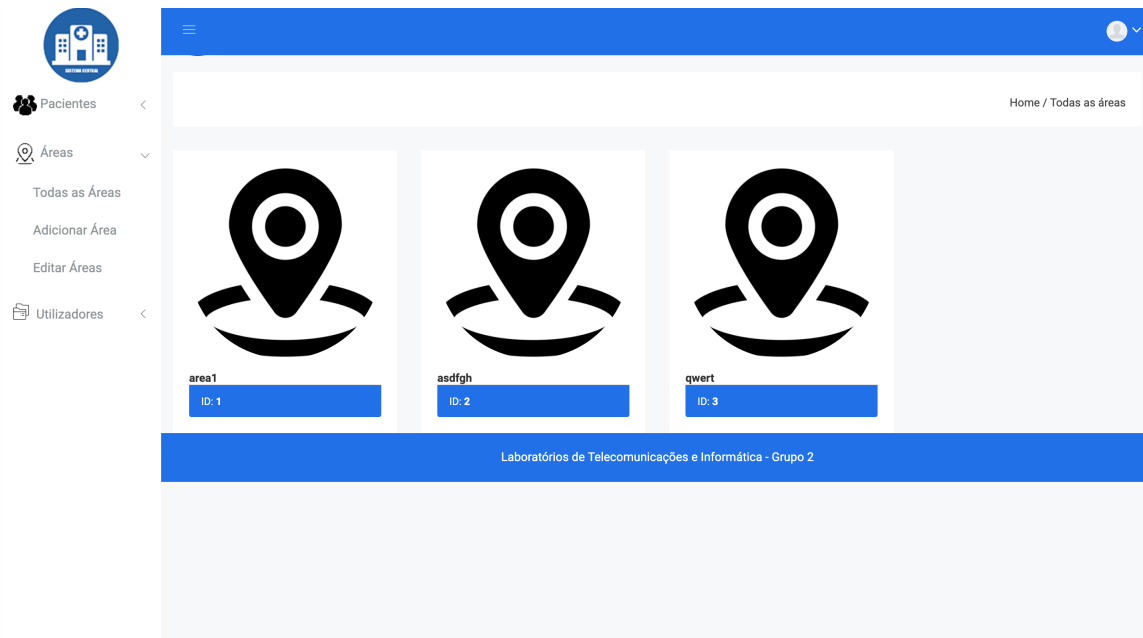


Figura 11: Visualização da página de listagem de áreas.

Esta listagem de todas as áreas disponíveis é implementada da seguinte forma:

```

$(document).ready(function () {
    jQuery.support.cors = true;
    $.ajax({
        url: localStorage.getItem("base_url") + "areas",
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            appendAreaInfo(data.data);
        },
        error: function (xhr, ajaxOptions, errorThrown) {
            alert(xhr.status);
            alert(throwError);
        }
    });
});

function appendAreaInfo(areas) {
    // console.log("APPEND INFO: "+JSON.stringify(info));
    for(i=0; i<areas.length;i++){
        $("#areas").append(
            "<div class=\"col-lg-3 col-md-6 col-sm-6 col-xs-12\">"+
                "<div class=\"hpanel hblue contact-panel contact-panel-cs"+
                    responsive-mg-b-30\">"+

```

```

        "<div class=\"panel-body custom-panel-jw\">"+<img alt=\"logo\"
        class=\"img-circle m-b\" src=\"location-pointer.png\">"+
        "<h3><a href=\"#\">"+areas[i].nome_area+"</a></h3>"+</div>"+<div
        class=\"panel-footer contact-footer\">"+
        "<div class=\"professor-stds-int\">"+<div class=\"professor-stds\">"+
        "<div class=\"contact-stat\"><span>ID: </span>
        <strong>"+areas[i].id_area+"</strong></div>"+</div>"+</div>"+</div>"+
        "</div>"+</div>");
    }
}

```

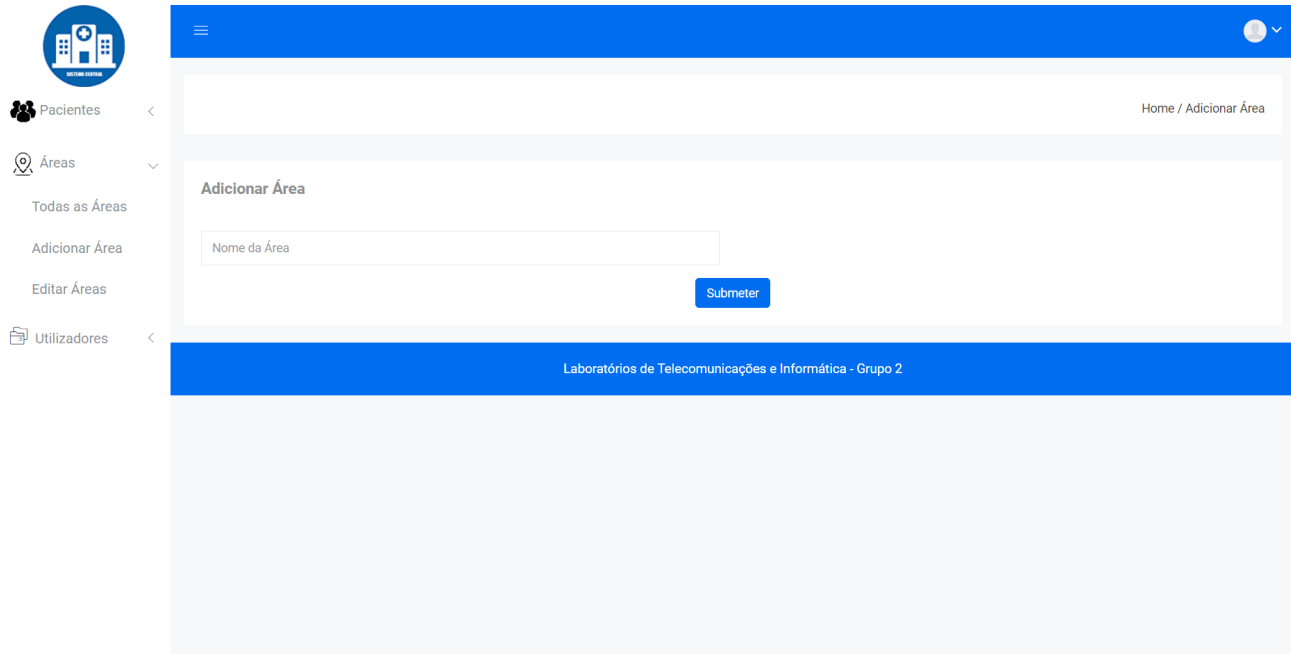


Figura 12: Visualização da página de adicionar áreas.

De forma a que o administrador possa adicionar novas áreas, foi implementado o seguinte código:

```

$(document).ready(function(){
    jQuery.support.cors = true;
    $('#submit-button').click(function (){
        var nome = $('#nome').val();
        if(nome==''){
            alert("Preencha todos os dados.");
        }
        else{
            $.ajax({
                url: localStorage.getItem('base_url')+"areas",
                type: 'POST',
                dataType: 'json',
                contentType: "application/json; charset=utf-8",
                crossDomain: true,
                beforeSend: function (xhr) {
                    xhr.setRequestHeader("Authorization", "Bearer " +
                        localStorage.getItem('token'));
                },
                data: JSON.stringify({"nome_area": nome}),
                success: function(data){
                    console.log(data);
                },
                error: function (xhr, ajaxOptions, thrownError) {
                    console.log(JSON.stringify(xhr));
                    console.log(JSON.stringify(thrownError));
                    alert("No foi possvel adicionar a rea.");
                }
            });
        }
    });
});

```

});

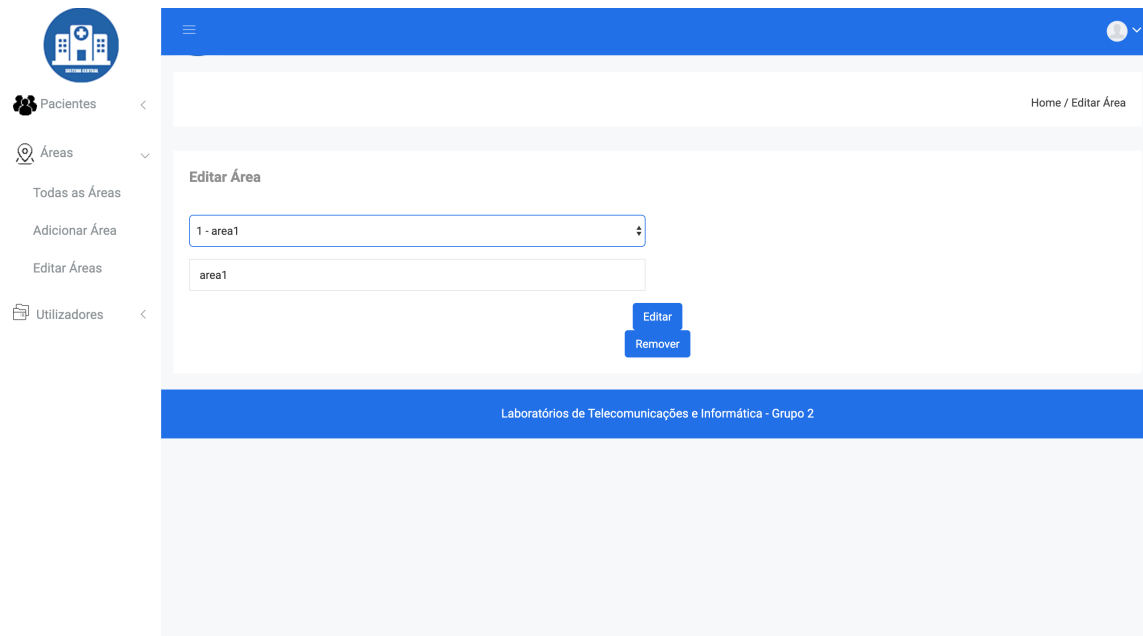


Figura 13: Visualização da página de editar áreas.

Para editar parâmetros de áreas existentes a implementação foi a seguinte:

```
$(document).ready(function(){
jQuery.support.cors = true;
var id;
var dataglobal;
$.ajax({
url: localStorage.getItem("base_url") + "areas",
type: 'GET',
contentType: "application/json; charset=utf-8",
dataType: "json",
beforeSend: function (xhr) {
xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
},
data: {},
success: function (data) {
appendAreaInfo(data.data);
console.log(JSON.stringify(data.data));
dataglobal=data.data;
},
error: function (xhr, ajaxOptions, thrownError) {
alert(xhr.status);
alert(thrownError);
}
});
function appendAreaInfo(areas) {
for(i=0; i<areas.length;i++){
$("#select").append(
"<option value='"+areas[i].id_area+"'>" + ">" + areas[i].id_area + "</option>" +
"</select>");
}
}
$('#select').change(function (){
id = $('#select').val();
console.log("selected");
if(id==''){
alert("Preencha todos os dados.");
}
$.ajax({
url: localStorage.getItem("base_url") + "areas",
type: 'GET',
contentType: "application/json; charset=utf-8",
```

```

        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            appendAreaInfo(data.data);
            console.log(JSON.stringify(data.data));
        },
        error: function (xhr, ajaxOptions, thrownError) {
            alert(xhr.status);
            alert(thrownError);
        }
    });
    console.log(dataglobal[id-1].nome_area);
    $("#nome_area").val(dataglobal[id-1].nome_area);
});
$('#submit').click(function () {
    console.log("selected");
    var nome = $('#nome_area').val();
    if(id=='' && nome==''){
        alert("Preencha todos os dados.");
    }
    else{
        $.ajax({
            url: localStorage.getItem('base_url')+"areas/"+id,
            type: 'PUT',
            dataType: 'json',
            contentType: "application/json; charset=utf-8",
            crossDomain: true,
            beforeSend: function (xhr) {
                xhr.setRequestHeader("Authorization", "Bearer " +
                    localStorage.getItem('token'));
            },
            data: JSON.stringify({"nome_area": nome}),
            success: function (data) {
                console.log(data);
                alert("rea editada com sucesso.");
            },
            error: function (xhr, ajaxOptions, thrownError) {
                console.log(JSON.stringify(xhr));
                console.log(JSON.stringify(thrownError));
                alert("No foi possvel editar a rea.");
            }
        });
    }
});
$('#apagar').click(function () {
    $.ajax({
        url: localStorage.getItem('base_url')+"areas/"+id,
        type: 'DELETE',
        dataType: 'json',
        contentType: "application/json; charset=utf-8",
        crossDomain: true,
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        success: function (data) {
            console.log(data);
            alert("Area removida.");
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(JSON.stringify(xhr));
            console.log(JSON.stringify(thrownError));
            alert("No foi possvel remover a rea.");
        }
    });
});
});
});

```

O "Admin" também pode ainda, listar os utilizadores, adicionar novos utilizadores, e editar utilizadores, como mostram as figuras 17, 15, 16, respetivamente.

Para visualizar todos os utilizadores disponíveis, foi preciso implementar o seguinte código:

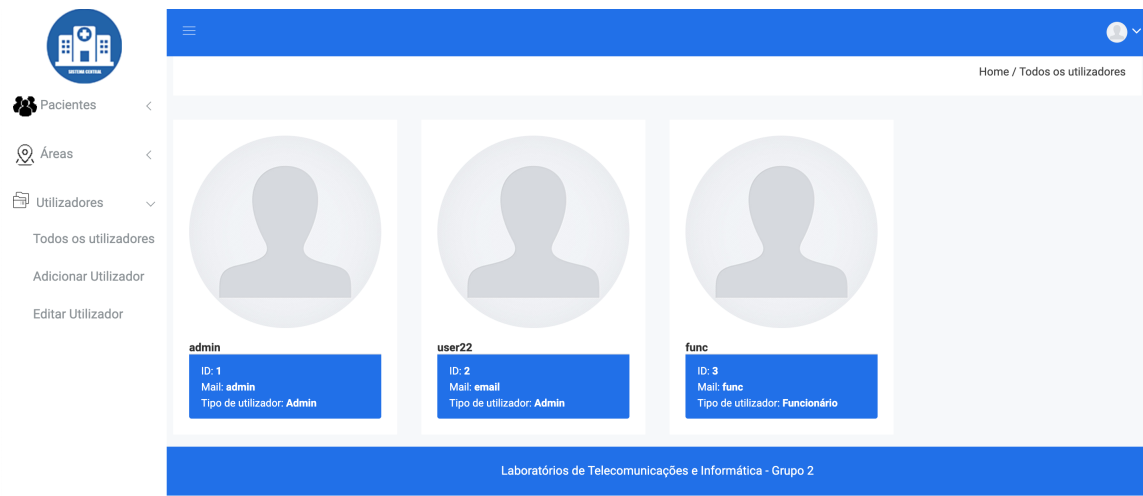


Figura 14: Visualização da página de listar os utilizadores.

```

$(document).ready(function () {
jQuery.support.cors = true;
$.ajax({
  url: localStorage.getItem("base_url") + "utilizadores",
  type: 'GET',
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  beforeSend: function (xhr) {
    xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
  },
  data: {},
  success: function (data) {
    appendUsersInfo(data.data);
  },
  error: function (xhr, ajaxOptions, thrownError) {
    alert(xhr.status);
    alert(thrownError);
  }
});
});

function appendUsersInfo(users) {
  for(i=0; i<users.length;i++){
    $("#users").append(
      "<div class=\"col-lg-3 col-md-6 col-sm-6 col-xs-12\">"+
        "<div class=\"hpanel hblue contact-panel contact-panel-cs responsive-mg-b-30\">"+
          "<div class=\"panel-body custom-panel-jw\">"+
            "<img alt=\"logo\" class=\"img-circle m-b\" src=\"default-user.png\">"+
            "<h3><a href=\"edit-admin.html\">"+
              users[i].username+
            "</a></h3>"+
          "</div>"+
          "<div class=\"panel-footer contact-footer\">"+
            "<div class=\"professor-stds-int\">"+
              "<div class=\"professor-stds\">"+
                "<div class=\"contact-stat\"><span>ID: </span> <strong>"+
                  users[i].id_utilizador+
                "</strong></div>"+
                "<div class=\"contact-stat\"><span>Mail: </span> <strong> "+
                  users[i].mail+
                "</strong></div>"+
              "</div>"+
            "</div>"+
          "</div>"+
        "</div>"+
      "</div>"+
    );
  }
}

```

The screenshot shows a web application interface for adding a new user. The page is titled 'Adicionar Utilizador' and is part of a system for 'Pacientes' (Patients). The left sidebar contains navigation links: 'Pacientes', 'Áreas', 'Utilizadores' (with a dropdown), 'Todos os utilizadores', 'Adicionar Utilizador', and 'Editar Utilizador'. The main form includes input fields for 'Username', 'Password', 'Morada', 'E-mail', and a dropdown for 'Tipo de utilizador'. A 'Submeter' button is located below the form. The footer of the page indicates it is from 'Laboratórios de Telecomunicações e Informática - Grupo 2'.

Figura 15: Visualização da página de adicionar utilizadores.

Para a implementação desta funcionalidade de adicionar utilizadores foi preciso o seguinte código:

```
$(document).ready(function(){
    jQuery.support.cors = true;
    // obter todos tipos de user e inserir no template
    $.ajax({
        url: localStorage.getItem("base_url") + "tiposUtilizador",
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            appendUserTypesInfo(data.data);
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(xhr.status);
            console.log(thrownError);
        }
    });
    function appendUserTypesInfo(userTypes) {
        for(i=0; i<userTypes.length;i++){
            $("#userTypes").append(
                "<option value="+
                userTypes[i].id_tipoUtilizador+">" + userTypes[i].id_tipoUtilizador + " - " +
                userTypes[i].tipo +
                "</option>"
            );
        }
    }
    $('#submit').click(function () {
        var username = $('#username').val();
        var password = $('#password').val();
        var morada = $('#morada').val();
        var mail = $('#mail').val();
        var userTypes = $('#userTypes').val();
        if(username=='' || morada=='' || mail=='' || userTypes=='' || password==""){
            alert("Preencha todos os dados.");
        }
        else{
            $.ajax({
                url: localStorage.getItem('base_url')+"register",
```

```

type: 'POST',
dataType: 'json',
contentType: "application/json; charset=utf-8",
crossDomain: true,
beforeSend: function (xhr) {
    xhr.setRequestHeader("Authorization", "Bearer " +
        localStorage.getItem('token'));
},
data: JSON.stringify({"username": username, "password": password, "morada":
    morada, "mail": mail, "id_tipoUtilizador": userTypes}),
success: function(data){
    console.log(data);
    window.location.href = "all-users.html";
},
error: function (xhr, ajaxOptions, thrownError) {
    console.log(JSON.stringify(xhr));
    console.log(JSON.stringify(thrownError));
    alert("No foi possível adicionar o utilizador.");
}
});
});
});
});
});

```

Figura 16: Visualização da página de editar utilizadores.

De modo a permitir que seja possível a modificação de parâmetros de um utilizador existente, foi necessário a seguinte implementação:

```
$(document).ready(function () {
    jQuery.support.cors = true;
    // obter todos tipos de user e inserir no template
    $.ajax({
        url: localStorage.getItem("base_url") + "tiposUtilizador",
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            appendUserTypesInfo(data.data);
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(xhr.status);
            console.log(thrownError);
        }
    });
    // obter todos users e inserir no template
    $.ajax({
        url: localStorage.getItem("base_url") + "utilizadores",
        type: 'GET',
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
        },
        data: {},
        success: function (data) {
            appendUsersInfo(data.data);
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(xhr.status);
            console.log(thrownError);
        }
    });
    function appendUsersInfo(users) {
        for (i = 0; i < users.length; i++) {
            $("#users").append(
                "<option value=" +
                users[i].id_utilizador + ">" + users[i].id_utilizador + " - " + users[i].username +
                "</option>"
            );
        }
    }
    function appendUserTypesInfo(userTypes) {
        for (i = 0; i < userTypes.length; i++) {
            $("#userTypes").append(
                "<option value=" +
                userTypes[i].id_tipoUtilizador + ">" + userTypes[i].id_tipoUtilizador + " - " +
                userTypes[i].tipo +
                "</option>"
            );
        }
    }
    // carregar dados para user escolhido
    $('#users').change(function(){
        id = $('#users').val();
        console.log("selected: "+id);
        $.ajax({
            url: localStorage.getItem("base_url") + "utilizadores/" + id,
            type: 'GET',
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            beforeSend: function (xhr) {
                xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
            },
            data: {},
            success: function (data) {
                console.log(JSON.stringify(data.data));
                $('#username').val(data.data[0].username);
            }
        });
    });
});
```

```

        $('#morada').val(data.data[0].morada);
        $('#mail').val(data.data[0].mail);
        $('#userTypes').val(data.data[0].id_tipoUtilizador);
    },
    error: function (xhr, ajaxOptions, thrownError) {
        console.log(xhr.status);
        console.log(thrownError);
    }
});
});
});
$('#submit').click(function () {
    var id = $('#users').val();
    var username = $('#username').val();
    var password = $('#password').val();
    var morada = $('#morada').val();
    var mail = $('#mail').val();
    var userTypes = $('#userTypes').val();
    if (id=='' || username == '' || morada == '' || mail == '' || userTypes == '' ||
        password == "") {
        alert("Preencha todos os dados.");
    }
    else {
        $.ajax({
            url: localStorage.getItem('base_url') + "utilizadores/"+id,
            type: 'PUT',
            dataType: 'json',
            contentType: "application/json; charset=utf-8",
            crossDomain: true,
            beforeSend: function (xhr) {
                xhr.setRequestHeader("Authorization", "Bearer " + localStorage.getItem('token'));
            },
            data: JSON.stringify({ "username": username, "password": password, "morada":
                morada, "mail": mail, "id_tipoUtilizador": userTypes }),
            success: function (data) {
                console.log(data);
                window.location.href = "all-users.html";
            },
            error: function (xhr, ajaxOptions, thrownError) {
                console.log(JSON.stringify(xhr));
                console.log(JSON.stringify(thrownError));
                alert("No foi possivel editar o utilizador.");
            }
        });
    }
});
});
});

```

Também podemos observar na seguinte figura a página *home* de um utilizador do tipo 2(Funcionário), que não sendo "Admin", apenas tem permissão para visualizar dados sobre os pacientes e áreas, mas não para alterá-los.

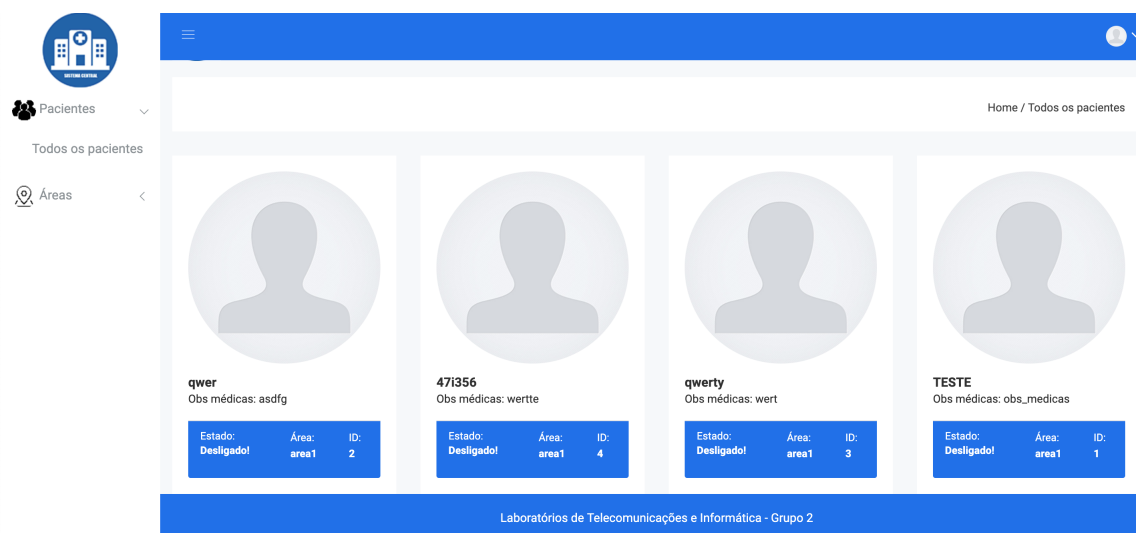


Figura 17: Visualização da página de listar os utilizadores.

Todo o *front-end* implementado foi com recurso ao *JavaScript* e o grupo teve sempre em consideração a interface construída pois deve ser *user-friendly*, ou seja, mantendo sempre a simplicidade de acesso a qualquer funcionalidade e de fácil compreensão.

6.2. Back-end

Depois de explicada a nossa implementação ao nível do *front-end*, iremos agora detalhar a parte que irá gerir todos os dados recebidos através de todos os componentes do sistema construído e que serão mostrados ao utilizador posteriormente.

Este *back-end* implementado pelo grupo é composto por um serviço *web* com recurso a *NodeJS* que contém duas classes que são determinantes para o funcionamento correto do Sistema Central e inclui uma nova base de dados relacional que irá guardar todos os dados inseridos pelo utilizador que usa a interface do sistema e os que chegam do Gestor de Serviço, através do serviço *web* implementado:

- **Serviço Web:**

- `"dataHandler.js";`
- `"server.js";`

- **Base de Dados Relacional:**

- `"bd_central.sql";`

6.2.1. Classe: *dataHandler.js*

Nesta classe são implementadas todas as *queries* necessárias para receber todos os dados necessários da base de dados de modo a que se possa enviar, posteriormente, para o utilizador através do *front-end* realizado.

De seguida, iremos mostrar exemplos de cada tipo de função implementada para as *queries* utilizadas:

- Obter Área pelo ID:

```
getAreaById: function (req, callback) {
  let query = "SELECT * FROM area WHERE id_area = ?";
  let table = [req.params.id_area];
  query = mysql.format(query, table);
  pool.query(query, function (error, results) {
    if (error) {
      err = { code: status.INTERNAL_SERVER_ERROR, message: error };
      return callback(err, null);
    }
    else {
      if (results.length > 0)
        return callback(null, results);
      else {
        let err = { code: status.NOT_FOUND, message: "Area doesn't exist" };
        return callback(err, null);
      }
    }
  });
},
```

- Inserir Sujeito:

```
insertPaciente: function (req, callback) {
  if (!req.body.nome_paciente || !req.body.morada || !req.body.nif ||
    !req.body.obs_medicas || !req.body.telefone || !req.body.id_area) {
    let err = { code: status.BAD_REQUEST, message: "Please provide paciente" };
    return callback(err, null);
  }
  else {
    let query = "call inserir_paciente(?,?,?,?,?)";
    let table = [req.body.nome_paciente, req.body.morada, req.body.nif,
      req.body.obs_medicas, req.body.telefone, req.body.id_area];
    query = mysql.format(query, table);
    pool.query(query, function (errorInsert, resultsInsert) {
      if (errorInsert) {
        err = { code: status.INTERNAL_SERVER_ERROR, message: errorInsert };
        return callback(err, null);
      }
      else {
        return callback(null, resultsInsert);
      }
    });
  }
},
```

- Atualização do estado do paciente:

```
updateLeitura: function (req, callback) {
  if (!req.body.estado_fisico || !req.body.id_paciente) {
    let err = { code: status.BAD_REQUEST, message: "Please provide Leitura" };
    return callback(err, null);
  }
  else {
    let query = "UPDATE leitura SET estado_fisico=?,id_paciente=? WHERE id_leitura = ?";
    let table = [req.body.estado_fisico, req.body.id_paciente, req.params.id_leitura];
```

```

    query = mysql.format(query, table);
    pool.query(query, function (error, results) {
      if (error) {
        err = { code: status.INTERNAL_SERVER_ERROR, message: error };
        return callback(err, null);
      }
      else {
        if (results.affectedRows > 0)
          return callback(null, results);
        else {
          err = { code: status.NOT_FOUND, message: "Leitura doesn't exist" };
          return callback(err, null);
        }
      }
    });
  },
},

```

- Apagar utilizador:

```

deleteUtilizador: function (req, callback) {
  let query = "DELETE from utilizador WHERE id_utilizador = ?";
  let table = [req.params.id_utilizador];
  query = mysql.format(query, table);
  pool.query(query, function (error, results) {
    if (error) {
      err = { code: status.INTERNAL_SERVER_ERROR, message: error };
      return callback(err, null);
    }
    else {
      if (results.affectedRows > 0)
        return callback(null, results);
      else {
        err = { code: status.NOT_FOUND, message: "Utilizador doesn't exist" };
        return callback(err, null);
      }
    }
  });
},

```

É possível visualizar nos exemplos apresentados a implementação de 4 tipos de operações nas funções construídas, onde nos baseamos em inserir, atualizar, remover ou obter dados vindos da base de dados implementada, sendo que foram realizadas várias operações para o funcionamento correto deste serviço.

De modo a conseguir implementar o *login* dos utilizadores e registo dos mesmos para acesso ao Sistema Central, foram construídas as seguintes funções:

- Registo:

```

register: function (req, callback) {
  if (!req.body.username || !req.body.morada || !req.body.mail || !req.body.password
    || !req.body.id_tipoUtilizador) {
    let err = { code: status.BAD_REQUEST, message: "Please provide User" };
    return callback(err, null);
  }
  else {
    //verificar se email j existe
    let query = "select * from utilizador where mail = ?";
    let table = [req.body.email];
    query = mysql.format(query, table);
    pool.query(query, function (errorMail, resultsMail) {
      if (errorMail) {
        err = { code: status.INTERNAL_SERVER_ERROR, message: errorMail };
        return callback(err, null);
      }
      else {
        if (resultsMail.length > 0) {
          //mail j existe, no possivel registar

```



```
err = { code: status.BAD_REQUEST, message: "Mail already exists" };
return callback(err, null);
} else {
    let hashedPassword = bcrypt.hashSync(req.body.password, 8);
    //mail no existe, pode fazer insert
    let query = "call inserir_utilizador(?,?,?,?)";
    let table = [req.body.username, req.body.morada, req.body.mail,
        hashedPassword, req.body.id_tipoUtilizador];
    query = mysql.format(query, table);
    pool.query(query, function (error, results) {
        if (error) {
            err = { code: status.INTERNAL_SERVER_ERROR, message: error };
            return callback(err, null);
        }
        else {
            //create token
            var token = jwt.sign({ id: results.insertId }, "config.secret", {
                expiresIn: 86400 // expires in 24 hours
            });
            results.token = token;
            results.auth = true;
            console.log(JSON.stringify(results));
            return callback(null, results);
        }
    });
}
});
}
});
},
}
```

- Login:

```
login: function (req, callback) {
  if (!req.body.password || !req.body.mail) {
    let err = { code: status.BAD_REQUEST, message: "Please provide utilizador" };
    return callback(err, null);
  } else {
    let query = "select * from utilizador where mail = ?";
    let table = [req.body.mail];
    query = mysql.format(query, table);
    pool.query(query, function (error, results) {
      if (error) {
        err = { code: status.INTERNAL_SERVER_ERROR, message: error };
        return callback(err, null);
      } else {
        if (results.length > 0) {
          //check password
          console.log(JSON.stringify(req.body) + " \n" + JSON.stringify(results));
          var passwordIsValid = bcrypt.compareSync(req.body.password,
            results[0].password);
          if (!passwordIsValid) {
            err = { code: status.UNAUTHORIZED, message: "Wrong password" };
            return callback(err, null);
          }
          var token = jwt.sign({ id: results.id_user }, "config.secret", {
            expiresIn: 86400 // expires in 24 hours
          });
          results[0].token = token;
          results[0].auth = true;
          console.log(JSON.stringify(results));
          return callback(null, results[0]);
        } else {
          let err = { code: status.NOT_FOUND, message: "Utilizador doesn't exist" };
          return callback(err, null);
        }
      }
    });
  }
}
```

6.2.2. Classe: *server.js*

Esta classe implementada pelo grupo é de extrema importância pois aqui estão declaradas todas as rotas necessárias para obter e introduzir todos os objetos e dados relevantes para o âmbito do projeto em questão.

Como já foi referido acima no relatório, a nossa *API* possui todos os parâmetros necessários para a gestão pretendida, sendo que os pedidos usados são o *POST*, *GET*, *PUT*, *DELETE*.

De seguida, é apresentada uma rota definida pelo grupo para cada pedido referido acima, dentro das várias disponíveis, de forma a exemplificar o uso que foi dado às mesmas:

- Rota Registrar Utilizador (POST):

```
app.post('/register', function (req, res) {
  dataHandler.register(req, function (error, results) {
    if (error) {
      res.status(error.code).send(error.message);
    }
    else {
      return res.status(status.CREATED).send({ error: false, data: results, message:
        'New User has been registered successfully' });
    }
  });
});
```

- Rota Obter Todos Os Utilizadores (GET):

```
// route: get all utilizadores
app.get('/utilizadores', verifyToken, function (req, res) {
  jwt.verify(req.token, "config.secret", (err, authData) => {
    if (err) {
      res.status(status.UNAUTHORIZED).send("Wrong token");
    }
    else {
      dataHandler.getUtilizadores(function (error, results) {
        if (error) {
          res.status(error.code).send(error.message);
        }
        else {
          res.status(status.OK).send({ error: false, data: results, message:
            'Utilizadores.' });
        }
      });
    }
  });
});
```

- Atualizar Utilizador (PUT)

```
app.put('/utilizadores/:id_utilizador', verifyToken, function (req, res) {
  jwt.verify(req.token, "config.secret", (err, authData) => {
    if (err) {
      res.status(status.UNAUTHORIZED).send("Wrong token");
    }
    else {
      dataHandler.updateUtilizador(req, function (error, results) {
        if (error) {
          res.status(error.code).send(error.message);
        }
        else {
          return res.status(status.CREATED).send({ error: false, data: results,
            message: 'New utilizador has been modified successfully.' });
        }
      });
    }
  });
});
```

```

    });
  });
});

```

- Remover Paciente (DELETE):

```

app.delete('/pacientes/:id_paciente', verifyToken, function (req, res) {
  jwt.verify(req.token, "config.secret", (err, authData) => {
    if (err) {
      res.status(status.UNAUTHORIZED).send("Wrong token");
    } else {
      dataHandler.deletePaciente(req, function (error, results) {
        if (error) {
          res.status(error.code).send(error.message);
        } else {
          return res.status(status.CREATED).send({ error: false, data: results,
            message: 'Paciente deleted successfully' });
        }
      });
    }
  });
});

```

O que cada rota faz é, consoante o pedido que têm por parte do utilizador, encaminham o mesmo para o servidor onde este irá executar alguma acção sobre a base de dados implementada dependendo do que foi requerido, podendo ser a obtenção de dados ou a alteração dos mesmos.

6.2.3. Base de Dados Relacional: *bd_central.sql*

A base de dados relacional implementada pelo grupo é constituída por 7 tabelas que irão guardar todos os dados relativos aos seguintes elementos:

- Concentrador;
- Área;
- Sujeito;
- Leitura;
- Sistema Sensor;
- Utilizador;
- Tipo Utilizador;

Estas tabelas guardam todos os dados necessários para a interpretação dos dados recebidos do Gestor de Serviço através do Serviço *Web* e de modo a que cada utilizador do serviço consiga visualizar

posteriormente cada comportamento de um sujeito através das amostras guardadas para cada um na interface desenvolvida.

A implementação necessária para esta base de dados foi a seguinte:

```

drop database if exists servicoCentral;
create database servicoCentral;
use servicoCentral;
create table concentrador(
    id_concentrador integer(10) not null auto_increment,
    nome varchar(100),
    primary key (id_concentrador)
);
create table area(
    id_area integer(10) not null auto_increment,
    nome_area varchar(100),
    primary key (id_area)
);
create table paciente(
    id_paciente integer(10) not null auto_increment,
    nome_paciente varchar(100),
    morada varchar(100),
    nif integer(100),
    obs_medicas varchar(100),
    telefone integer(100),
    id_area integer(10),
    primary key (id_paciente),
    foreign key (id_area) references area(id_area) on update cascade on delete set null
);
create table leitura(
    id_leitura integer(10) not null auto_increment,
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado_fisico varchar(100),
    id_paciente integer(10),
    primary key (id_leitura),
    foreign key (id_paciente) references paciente(id_paciente) on update cascade on delete set null
);
create table sistemaSensor(
    id_sistemaSensor integer(10) not null auto_increment,
    sensorSimulado boolean,
    id_paciente integer(10),
    id_concentrador integer(10),
    primary key (id_sistemaSensor),
    foreign key (id_paciente) references paciente(id_paciente) on update cascade on delete set null,
    foreign key (id_concentrador) references concentrador(id_concentrador) on update cascade on delete set null
);
create table tipoUtilizador(
    id_tipoUtilizador integer(10) not null auto_increment,
    tipo varchar(100),
    primary key (id_tipoUtilizador)
);
create table utilizador(
    id_utilizador integer(10) not null auto_increment,
    username varchar(100),
    morada varchar(100),
    mail varchar(100),
    password varchar(100),
    id_tipoUtilizador integer(10),
    primary key (id_utilizador),
    foreign key (id_tipoUtilizador) references tipoUtilizador(id_tipoUtilizador) on update cascade on delete set null
);
delimiter //
create procedure inserir_concentrador (in nom varchar(100))
begin
    insert into concentrador(nome)
    values(nom);
end//
delimiter ;
delimiter //
create procedure inserir_area (in nom varchar(100))
begin
    insert into area(nome_area)
    values(nom);
end//

```

```

end//
delimiter ;
delimiter //
create procedure inserir_paciente (in nomePaciente varchar(50), m varchar(100), n
integer(100), o varchar(100),
t integer(100), id_a integer(10))
begin
insert into paciente(nome_paciente,morada,nif,obs_medicas,telefone,id_area)
values(nomePaciente,m,n,o,t,id_a);
call inserir_leitura('Desligado!',(select id_paciente from paciente order by id_paciente
desc limit 1));
end//
delimiter ;
delimiter //
create procedure inserir_leitura (in ef varchar(100), id_s integer(10))
begin
insert into leitura(estado_fisico , id_paciente)
values(ef,id_s);
end//
delimiter ;
delimiter //
create procedure inserir_sistemaSensor(in ssimulado boolean, id_s integer(10), id_c
integer(10))
begin
insert into sistemaSensor(sensorSimulado , id_paciente, id_concentrador)
values(ssimulado,id_s,id_c);
end//
delimiter ;
delimiter //
create procedure inserir_tipoUtilizador(in t varchar(100))
begin
insert into tipoUtilizador(tipo)
values(t);
end//
delimiter ;
delimiter //
create procedure inserir_utilizador(in u varchar(100), mo varchar(100), ma varchar(100),
pass varchar(100), id_t integer(10))
begin
insert into utilizador(username, morada, mail, password, id_tipoUtilizador)
values(u,mo,ma,pass,id_t);
end//
delimiter ;
ALTER TABLE concentrador AUTO_INCREMENT=0;
ALTER TABLE area AUTO_INCREMENT=0;
ALTER TABLE paciente AUTO_INCREMENT=0;
ALTER TABLE leitura AUTO_INCREMENT=0;
ALTER TABLE sistemaSensor AUTO_INCREMENT=0;
ALTER TABLE tipoUtilizador AUTO_INCREMENT=0;
ALTER TABLE utilizador AUTO_INCREMENT=0;
call inserir_concentrador('concentrador 1');
call inserir_area('areal');
call inserir_paciente('nome_paciente','morada',123456789,'obs_medicas',987654321,1);
call inserir_sistemaSensor(true , 1, 1);
call inserir_tipoUtilizador('admin');
call inserir_tipoUtilizador('funcionario');
call inserir_utilizador('admin','rua do admin','admin@admin.com','hashDaPassDoAdmin',1);

```

Como é possível visualizar, além das tabelas criadas foram também implementados procedimentos e triggers que permitem o bom e lógico funcionamento da base de dados e a interação externa com a mesma.

7. Conclusão

Finalizada a realização desta fase, o grupo fica satisfeito com o desempenho realizado, onde graças ao cumprimento rigoroso do planeamento definido permitiu que fosse possível o sucesso da implementação final deste sistema.

Foi possível construir uma interface gráfica para interação com utilizadores deste sistema, um serviço *web* capaz de lidar com todos os pedidos feitos ao Sistema Central e uma base de dados capaz de guardar e tratar qualquer tipo de dados necessários para a finalidade desta implementação.

Os dados obtidos dos sistemas sensores colocados em pacientes foi realizado através da ligação entre o Gestor de Serviço já implementado e o Sistema Central, através do serviço *web* utilizado que por sua vez irá colocá-los na base de dados existente.

Para concluir, o grupo destaca a harmonia que existe entre todas as implementações feitas em todas as fases onde agora, ao fim desta fase C, essas mesmas estão todas ligadas entre si e cada uma tem a sua função de modo a que o produto final seja possível de ser usado em situação real e de maneira fiável para os utilizadores que usufruírem deste serviço.

8. Referências

- [1] **Docentes de Laboratórios de Telecomunicações e Informática II**, "*Sistema de Monitorização de Atividade Física - ANEXO 4 – Fase C*", Universidade do Minho, 2019