Introduction

Integration

Tele-medicine platform

**Representational State Transfer** (**REST**)

**Representational State Transfer** (**REST**) is an architectural style that abstracts the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.REST has emerged as a predominant web API design model.

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine. Fielding is one of the principal authors of HTTP 1.0 and 1.1

The REST architectural style was developed by W3C Technical Architecture Group (TAG) in parallel with HTTP 1.1, based on the existing design of HTTP 1.0. The World Wide Web represents the largest implementation of a system conforming to the REST architectural style.

REST-style architectures conventionally consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can essentially be any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition.

**Web service**

The server side web service API allows clients like electronic health record providers to query medical information and to administer users and medical devices.

The type of web services implemented is REST web services based on the JSON notation. REST web services are supported on many platforms and libraries to call web services and process their response for most important programming languages exist such as, JavaScript, PHP and .NET.

A quick and easy check if the web service is installed correctly is to type in the following sample URLs in the browser:

* <https://zsrv02.zydacron.com/ZydacronWebService/CareuserService.svc/Rest/Hello>

Will return a “Hello Message”

The above example works with the Zydacron telecare demonstration server. After typing in these URLs a dialog will open prompting for a username and a password. For the demonstration server, the username is “webservice” and the password is “zydacron”.

**Apache HttpClient library**

The Hyper-Text Transfer Protocol (HTTP) is perhaps the most significant protocol used on the Internet today. Web services, network-enabled appliances and the growth of network computing continue to expand the role of the HTTP protocol beyond user-driven web browsers, while increasing the number of applications that require HTTP support.

Although the java.net package provides basic functionality for accessing resources via HTTP, it doesn't provide the full flexibility or functionality needed by many applications. The Jakarta Commons HttpClient component seeks to fill this void by providing an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations.

Designed for extension while providing robust support for the base HTTP protocol, the HttpClient component may be of interest to anyone building HTTP-aware client applications such as web browsers, web service clients, or systems that leverage or extend the HTTP protocol for distributed communication.

There are many projects that use HttpClient to provide the core HTTP functionality. Some of these are open source while others are closed source that you would never see or hear about. The Apache Source License provides maximum flexibility for source and binary reuse.

HttpClient was started in 2001 as a subproject of the Jakarta Commons, based on code developed by the Jakarta Slide project. It was promoted out of the Commons in 2004, graduating to a separate Jakarta project. In 2005, the HttpComponents project at Jakarta was created, with the task of developing a successor to HttpClient 3.x and to maintain the existing codebase until the new one is ready to take over. The Commons project, cradle of HttpClient, left Jakarta in 2007 to become an independent Top Level Project. Later in the same year, the [HttpComponents](http://httpcomponents.apache.org/" \o "External Link) project also left Jakarta to become an independent Top Level Project, taking the responsibility for maintaining HttpClient with it.

**HttpClient Features**

* Standards based, pure Java, implementation of HTTP versions 1.0 and 1.1
* Full implementation of all HTTP methods (GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE) in an extensible OO framework.
* Supports encryption with HTTPS (HTTP over SSL) protocol.
* Transparent connections through HTTP proxies.
* Tunneled HTTPS connections through HTTP proxies, via the CONNECT method.
* Basic, Digest, NTLMv1, NTLMv2, NTLM2 Session, SNPNEGO, Kerberos authentication schemes.
* Plug-in mechanism for custom authentication schemes.
* Pluggable secure socket factories, making it easier to use third party solutions
* Connection management support for use in multi-threaded applications. Supports setting the maximum total connections as well as the maximum connections per host. Detects and closes stale connections.
* Automatic Cookie handling for reading Set-Cookie: headers from the server and sending them back out in a Cookie: header when appropriate.
* Plug-in mechanism for custom cookie policies.
* Request output streams to avoid buffering any content body by streaming directly to the socket to the server.
* Response input streams to efficiently read the response body by streaming directly from the socket to the server.
* Persistent connections using KeepAlive in HTTP/1.0 and persistance in HTTP/1.1
* Direct access to the response code and headers sent by the server.
* The ability to set connection timeouts.
* Support for HTTP/1.1 response caching.
* Source code is freely available under the Apache License.

**The integration code and the results**

Firstly to connect to the Telemedicine server via the web services that provided by Zydacron Company we should import the jar files that contain all HttpClient libraries produced by Apache Company, and we can download these files from <http://hc.apache.org/downloads.cgi>, then we have to write REST client side using java programming language to invoke the services that we have. The Rest client side has five request methods Get, Put, Post, Delete, and Patch, REST uses these operations and other existing features of the HTTP protocol. We use Get method to retrieve the data because we want to get data from the telemedicine platform server.

The implementation of the REST client side explained below in more detail:

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**impl**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.impl)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.impl.client)**.CloseableHttpClient** |
| code | CloseableHttpClient client = HttpClients.*createDefault*(); |
| Description | Implement the client’s object to allow us to connect to the telemedicine platform server |

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.**[**config**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client.config)**.RequestConfig** |
| code | RequestConfig conf = RequestConfig.*custom*().setSocketTimeout(1000).setConnectTimeout(1000).build(); |
| Description | Implement the configuration object to configure the connection to the telemedicine platform server |

In the above code we set connection time out to 1000ms and the socket time out to 1000ms.

|  |  |
| --- | --- |
| Library | [**java**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava)**.**[**lang**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava.lang)**.String** |
| code | String server\_address="https://zsrv02.zydacron.com/ZydacronWebService/"; |
| Description | Implement the server\_address as string and assign it with the telemedicine platform server real address |

|  |  |
| --- | --- |
| Library | [**java**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava)**.**[**lang**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava.lang)**.String** |
| code | String [] careuser\_service={"CareUserService.svc/Rest/GetCareUsersByFilter","CareUserService.svc/Rest/GetCareUser?username=","CareUserService.svc/Rest/GetDefaultContact?username=","CareUserService.svc/Rest/GetAvailableDefaultContacts?username="};    String [] medical\_service={"MedicalService.svc/Rest/GetMedicalDevicesForStoredMeasurements?username=","MedicalService.svc/Rest/GetAvailableVitalSignsForUser?username=","MedicalService.svc/Rest/GetMedicalThresholds?username=","MedicalService.svc/Rest/GetMedicalAlarms"}; |
| Description | Implement the web services address as array of string to allow us to connect to the correct rout |

In the above code we declare two arrays of web services one of them to get the careuser information and more details while the second array for the medical web service using this service we can get all careuser medical tests and measurements.

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.**[**methods**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client.methods)**.HttpGet** |
| code | HttpGet httpget1= **new** HttpGet(server\_address + careuser\_service[0]); |
| Description | Declare HTTP request as HttpGet method to retrieve the data from the assign web service address. |

In the previous code the HttpGet had one parameter as String , this parameter contain the address of the web service and the query that retrieve the data that we want. HttpGet similar to select statement in SQL language.

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.CredentialsProvider** |
| code | CredentialsProvider authio = **new** BasicCredentialsProvider();  CredentialsProvider authio1 = **new** BasicCredentialsProvider(); |
| Description | Implement authio, authio1 objects as CredentialsProvider for the authentication. |

In the previous code we had to implement an object as CredentialsProvider to pass the username and the password to the connection and allow it to access the telemedicine server, because we have a username and password in the web service which ensures the safety of the data.

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**auth**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.auth)**.UsernamePasswordCredentials** |
| code | UsernamePasswordCredentials userpass = **new** UsernamePasswordCredentials("webservice", "zydacron");  UsernamePasswordCredentials userpass1 = **new** UsernamePasswordCredentials("Operator1", "op"); |
| Description | Implement userpass, userpass1 objects as UsernamePasswordCredentials and pass the username and password as argument to the function |

In the previous code we had implemented two objects as UsernamePasswordCredentials one for the Careusers web service to allow the client access their information from the server while the second object for the medical web service which allow the client to access the medical information of the Careuser.

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.CredentialsProvider** |
| code | authio.setCredentials( **new** AuthScope(AuthScope.*ANY\_HOST*, AuthScope.*ANY\_PORT*), userpass );  authio1.setCredentials( **new** AuthScope(AuthScope.*ANY\_HOST*, AuthScope.*ANY\_PORT*), userpass1 ); |
| Description | Pass the main parameters to the CredentialsProvider objects (autho,autho1), these parameters are the host name , the host port and the username and password using setCredentials method. |

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.**[**protocol**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client.protocol)**.HttpClientContext** |
| code | HttpClientContext context= HttpClientContext.*create*();  context.setCredentialsProvider(authio);  context.setRequestConfig(conf);  HttpClientContext context1= HttpClientContext.*create*();  context1.setCredentialsProvider(authio1);  context1.setRequestConfig(conf); |
| Description | We had implemented context object as HttpClientContext to set the configuration and the authentication to the client connection |

|  |  |
| --- | --- |
| Library | [**org**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg)**.**[**apache**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache)**.**[**http**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http)**.**[**client**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client)**.**[**methods**](eclipse-javadoc:%E2%98%82=medclient/D:%5C/telemedicen%20platform%5C/lib%5C/httpclient-4.3.jar%3Corg.apache.http.client.methods)**.CloseableHttpResponse** |
| code | CloseableHttpResponse response = client.execute(httpget1 , context); |
| Description | We had implemented response object as CloseableHttpResponse to get response from the server and execute the HttpGet request method, and this function has two arguments one is the HttpGet while the second is the context. |

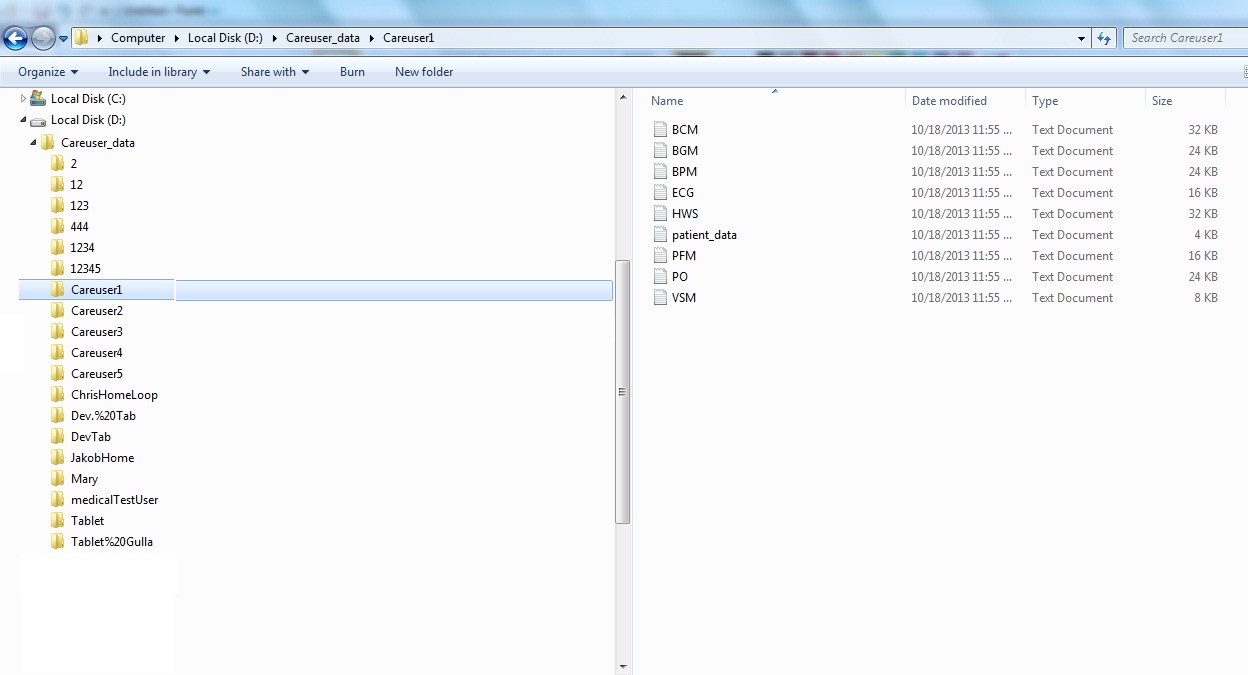
|  |  |
| --- | --- |
| Library | [**java**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava)**.**[**io**](eclipse-javadoc:%E2%98%82=medclient/C:%5C/Program%20Files%5C/Java%5C/jre7%5C/lib%5C/rt.jar%3Cjava.io)**.BufferedReader** |
| code | BufferedReader rd = **new** BufferedReader (**new** InputStreamReader(response.getEntity().getContent())); |
| Description | We had implemented rd object as BufferedReader to read the data from the response object. |

In the previous code we had retrieved the data from the response command and we had stored the data in BufferedReader object, so we have to extract the data and we want to write it in output file, this file located in the D:/ driver with the careuser name.

In the following code we extracted the Careuser username to use it in another web services to retrieve more details about the careuser .

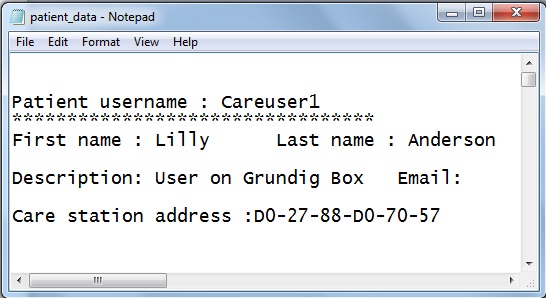
|  |
| --- |
| String line = "";  // extract the retrieved data from the BufferReader  **while** ((line = rd.readLine()) != **null**) {  // convert the retrieved data from string to array to extract all care users  String [] ar= line.split(",");  //Extract the care users username using the data in the array (ar)  **for** (**int** i=0; i<ar.length; i++){  //find the string which starts with Username and do some filtration to get the //careuser username clear  **if**(ar[i].startsWith("\"Username\":\"")==**true**){  String user=ar[i].replace("\"Username\":\"","");  user=user.replace("\"}", "");  user=user.replace("]", "").replace(" ", "%20");  //Create file and directory in d: drive to write Careuser data inside it  File file = **new** File("D:/Careuser\_data/" + user +"/patient\_data.txt");  File dirctory= **new** File("D:/Careuser\_data/" + user);  **if** (!dirctory.exists()) {  dirctory.mkdirs();  }  // if file doesn't exists, then create it  **if** (!file.exists()) {  file.createNewFile();  }    FileWriter fw = **new** FileWriter(file.getAbsoluteFile());  BufferedWriter bw = **new** BufferedWriter(fw); |

The output of the previous code shown in the following figure:

The folders which were shown in the above image under careuser\_data directory(2, 12, 123,….., Careuser1, Careuser2,…..etc) each one of them is for one Careuser and the name of these folders is the username of careuser .

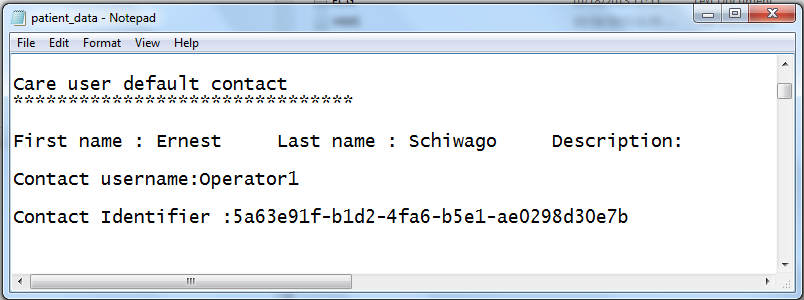
After extracting all careusers’ username we will use them to retrieve their details using this web service address (CareUserService.svc/Rest/GetCareUser?username=) as the following code shows:

|  |
| --- |
| //Retrieve all the care users details for specific careuser user name    HttpGet httpget2= **new** HttpGet(server\_address + careuser\_service[1]+user);  CloseableHttpResponse response1 = client.execute(httpget2, context);  //read the data from the response1 object and print it out  BufferedReader rd1 = **new** BufferedReader (**new** InputStreamReader(response1.getEntity().getContent()));  **try** {  String line1="";  **while** ((line1 = rd1.readLine()) != **null**) {  line1=line1.replace("{", "").replace("}", "");  String [] patientinfo= line1.split(",");  //filter the data and delete un wanted info and make it clean  String patient\_fname=patientinfo[4].replace("\"Firstname\":", "").replace("\"", "");  String patient\_lname=patientinfo[6].replace("\"Lastname\":", "").replace("\"", "");  String patient\_description=patientinfo[1].replace("\"Description\":", "").replace("\"", "");  String patient\_email=patientinfo[2].replace("\"Email\":", "").replace("\"", "");  String patient\_carestation=patientinfo[7].replace("\"LinkedCarestation\":", "").replace("\"", "");  // write the data in the created file  bw.append("Patient username : " + user);  bw.newLine();  bw.append( "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" );  bw.newLine();  bw.append("First name : " + patient\_fname + "\t" + "Last name : " + patient\_lname + "\t Description: " + patient\_description);  bw.append("Email:" + patient\_email + "\t Care station address :" + patient\_carestation + "\n");  } |

The output of the above code is stored in the text file called patient\_data in the first section , and the output is shown in the following figure for careuser (username = Careuser1):

Now we want to retrieve all default contacts that careuser has, so to retrieve these contacts we should use this web service (CareUserService.svc/Rest/GetDefaultContact?username=) as the following code shows:

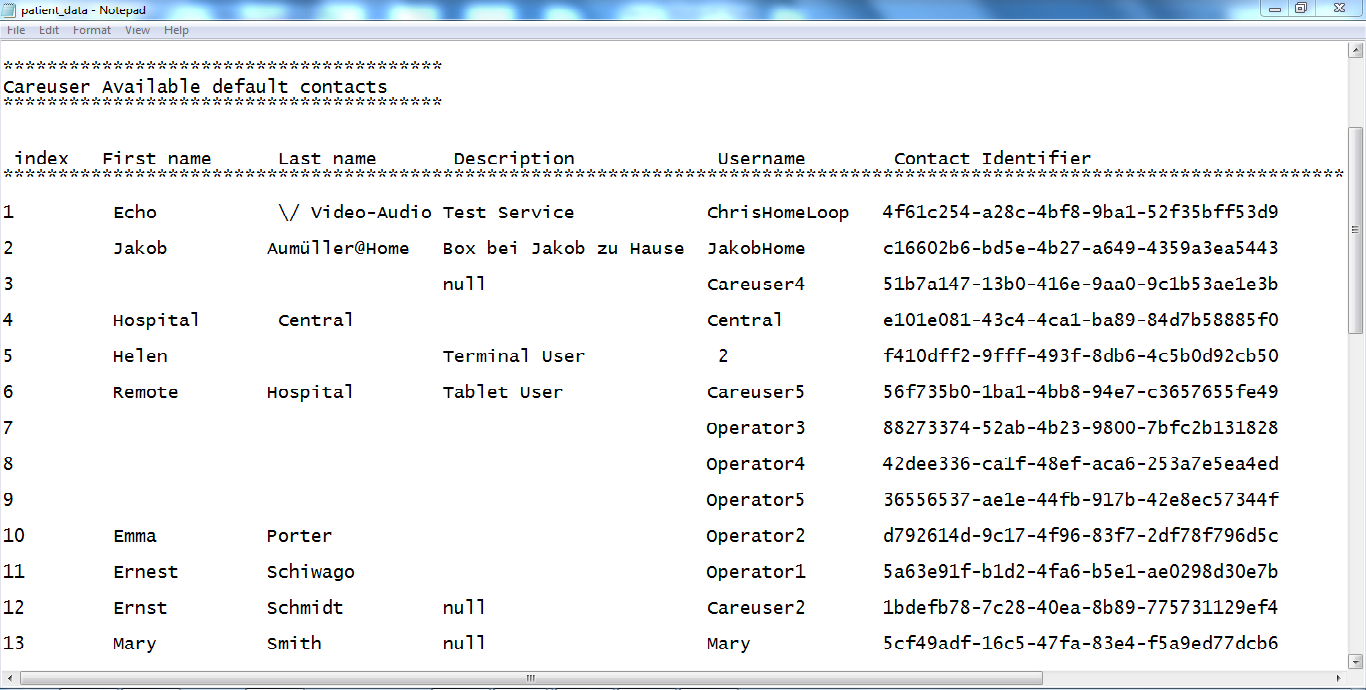
|  |
| --- |
| // retrieve care user default contacts  HttpGet httpget3= **new** HttpGet(server\_address + careuser\_service[2]+user);  CloseableHttpResponse response2 = client.execute(httpget3, context);  BufferedReader rd2 = **new** BufferedReader (**new** InputStreamReader(response2.getEntity().getContent()));    String line2="";  String [] patient\_default\_contact;  String contact\_firstname;  String contact\_lastname ;  String contact\_description;  String contact\_username;  String contact\_identifier;    bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  bw.append(" Care user default contact ");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();    **while** ((line2 = rd2.readLine()) != **null**) {  line2=line2.replace("{", "").replace("}", "");  patient\_default\_contact= line2.split(",");  **if**(patient\_default\_contact.length > 1) {  contact\_firstname= patient\_default\_contact[2].replace("\"Firstname\":", "").replace("\"", "");  contact\_lastname =patient\_default\_contact[4].replace("\"Lastname\":", "").replace("\"", "");  contact\_description= patient\_default\_contact[1].replace("\"Description\":", "").replace("\"", "");  contact\_username= patient\_default\_contact[8].replace("\"Username\":", "").replace("\"", ""); contact\_identifier=patient\_default\_contact[3].replace("\"Identifier\":", "").replace("\"", "");  // Show the careuser default contacts  bw.newLine();  bw.append("First name : " + contact\_firstname + "\t" + "Last name : " + contact\_lastname + "\t Description: " + contact\_description);  bw.newLine();  bw.append("Contact username:" + contact\_username + "\t Contact Identifier :" + contact\_identifier + "\n");  bw.newLine();  }    **else**  {  bw.newLine();  bw.append("No Availabel default contacts");  bw.newLine();  }  } |

The output of the above code is stored in the text file called patient\_data in careuser default contact section , and the output is shown in the following figure for careuser (username = Careuser1):

There are other important contacts and some responsible parts for the careuser that we have to retrieve , so to retrieve these Available default contacts we should use this web service address (CareUserService.svc/Rest/GetAvailableDefaultContacts?username=) as the following code shows:

|  |
| --- |
| // Retrieve all Available Default Contacts for care user  HttpGet httpget4= **new** HttpGet(server\_address + careuser\_service[3]+user);  CloseableHttpResponse response3 = client.execute(httpget4, context);  BufferedReader rd3 = **new** BufferedReader (**new** InputStreamReader(response3.getEntity().getContent()));    String line3="";  String [] patient\_Available\_default\_contact;  // to save all contact information and save each one of them in own index alone  String [] patient\_available\_default\_contact\_details;  // to save the details of the contact information String default\_contact\_firstname;  String default\_contact\_lastname ;  String default\_contact\_description;  String default\_contact\_username;  String default\_contact\_identifier;  **int** user\_contact\_index=1;    bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  bw.append("Careuser Available default contacts");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  **while** ((line3 = rd3.readLine()) != **null**) {  // Separate the contacts each one of them alone patient\_Available\_default\_contact= line3.split("\\}\\,\\{");    **if**(patient\_Available\_default\_contact.length > 1){  bw.append("\n index \t First name\t Last name\t Description \t \t Username \t Contact Identifier" );  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); bw.newLine();  **for** (**int** contact\_index=0; contact\_index<patient\_Available\_default\_contact.length;contact\_index++){  // replace the special characters from the default contacts and make it clear then split it to extract the details patient\_available\_default\_contact\_details=patient\_Available\_default\_contact[contact\_index].replace("[","").replace("]","").replace("{","").replace("}","").split(",");  //Extract and print out the patient default contacts  default\_contact\_firstname= patient\_available\_default\_contact\_details[2].replace("\"Firstname\":", "").replace("\"", "");  default\_contact\_lastname = patient\_available\_default\_contact\_details[4].replace("\"Lastname\":", "").replace("\"", "");  default\_contact\_description= patient\_available\_default\_contact\_details[1].replace("\"Description\":", "").replace("\"", "");  default\_contact\_username= patient\_available\_default\_contact\_details[8].replace("\"Username\":", "").replace("\"", "");  default\_contact\_identifier= patient\_available\_default\_contact\_details[3].replace("\"Identifier\":", "").replace("\"", "");  //write the Available default contact for the careuser  bw.newLine();  bw.append( user\_contact\_index +"\t " + default\_contact\_firstname + "\t \t" + default\_contact\_lastname + "\t\t" + default\_contact\_description +"\t" + default\_contact\_username +"\t" + default\_contact\_identifier + "\n");  bw.newLine();  user\_contact\_index++;  }  }  **else**  {  bw.newLine();  bw.append("No Availabel default contacts"); bw.newLine();  }  **}** |

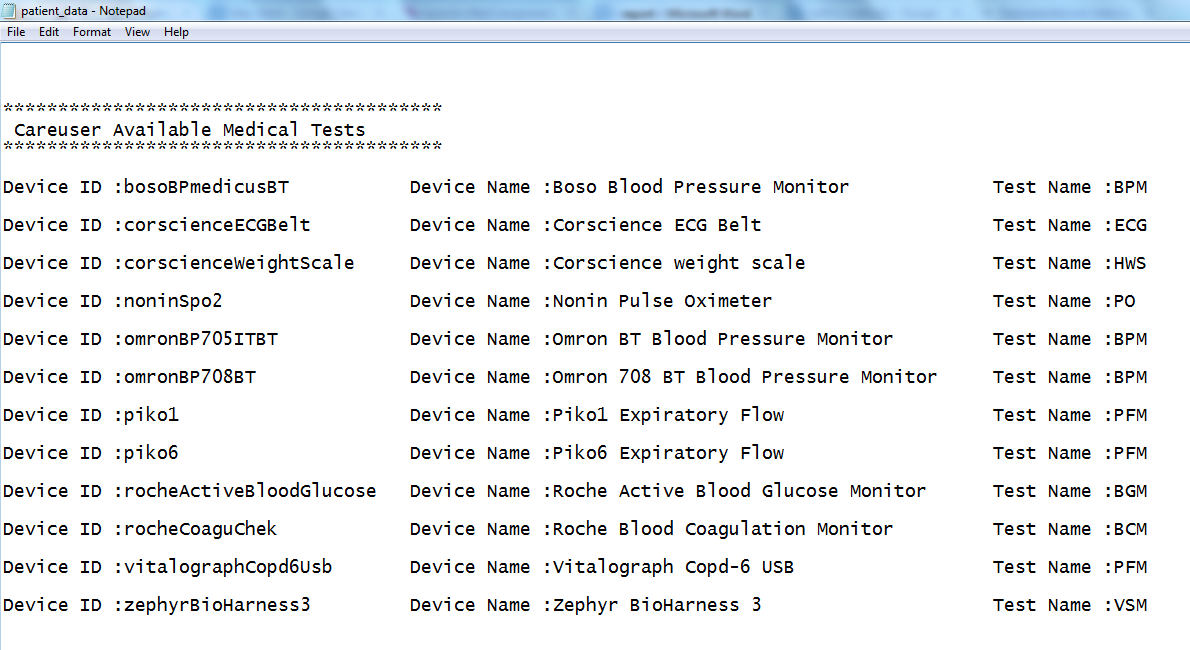
The output of the above code is stored in the text file called patient\_data in careuser available default contacts, and the output is shown in the following figure for careuser (username = Careuser1):



After retrieving the careuser information and the most important details, we have to retrieve their medical information (Health Record) and we should use this web service address (MedicalService.svc/Rest/GetMedicalDevicesForStoredMeasurements?username=)that allows us to connect to the telemedicine server, and we have to use context1 with different authentication parameters as the following code shows:

|  |
| --- |
| //retrieve care user medical tests  HttpGet httpget5= **new** HttpGet(server\_address + medical\_service[0] + user);  CloseableHttpResponse response4 = client.execute(httpget5, context1);  BufferedReader rd4 = **new** BufferedReader (**new** InputStreamReader(response4.getEntity().getContent()));    String line4="";  String [] patient\_medical\_measurments;  String [] patient\_measurment;  String[] patient\_test\_type;  String deviceid;  String devicename;  String testname;  String [] medical\_test\_image;  **int** [] medical\_test\_image\_integer;  **while** ((line4 = rd4.readLine()) != **null**) {  patient\_medical\_measurments= line4.split("]},");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); bw.newLine();  bw.append(" Careuser Available Medical Tests");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  **if**(patient\_medical\_measurments.length > 1) {  **for** (**int** Patient\_M\_Mearments=0; Patient\_M\_Mearments<patient\_medical\_measurments.length; Patient\_M\_Mearments++){    patient\_measurment=  patient\_medical\_measurments[Patient\_M\_Mearments].split("\"imageBytes\":");  patient\_measurment[0]=patient\_measurment[0].replace("{", "");  patient\_test\_type=patient\_measurment[0].split(",");  deviceid=patient\_test\_type[1].replace("\"DeviceID\":\"", "").replace("\"", ""); devicename = patient\_test\_type[2].replace("\"DeviceName\":\"", "").replace("\"", ""); testname = patient\_test\_type[9].replace("\"TypeName\":\"", "").replace("\"", "");    bw.newLine();  bw.append("Device ID :" + deviceid + "\t" + "Device Name :" + devicename + "\t" +"Test Name :" + testname + "\n");  bw.newLine();    // extract the test image pixels and convert them to image  patient\_measurment[1]=patient\_measurment[1].replace("[", "").replace("]", "").replace("{", "").replace("}",""); medical\_test\_image=patient\_measurment[1].split(",");  // convert the image pixels array from string to array of integer  medical\_test\_image\_integer =**new** **int**[medical\_test\_image.length];  **for**(**int** pixel\_index=0; pixel\_index < medical\_test\_image.length; pixel\_index++){  medical\_test\_image\_integer[pixel\_index]=Integer.*parseInt*(medical\_test\_image[pixel\_index]);  }  //Create file in d: drive to write patient image test pixels inside it  File image\_pixels = **new** File("D:/Careuser\_data/" + user +"/"+ testname + ".txt");  // if file doesn't exists, then create it  **if** (!image\_pixels.exists()) {  image\_pixels.createNewFile();  }    FileWriter image\_pixels\_fw = **new** FileWriter(image\_pixels.getAbsoluteFile());  BufferedWriter image\_pixels\_bw = **new** BufferedWriter(image\_pixels\_fw);    //Write the image pixels as array inside file    **for**(**int** image\_pixel\_index=0; image\_pixel\_index < medical\_test\_image\_integer.length; image\_pixel\_index++){  image\_pixels\_bw.append (medical\_test\_image\_integer[image\_pixel\_index] + " ");    }  //Close the buffer writer  image\_pixels\_fw.close();  }  }    **else**  {  System.*out*.println("No available measurment tests");  }  } |

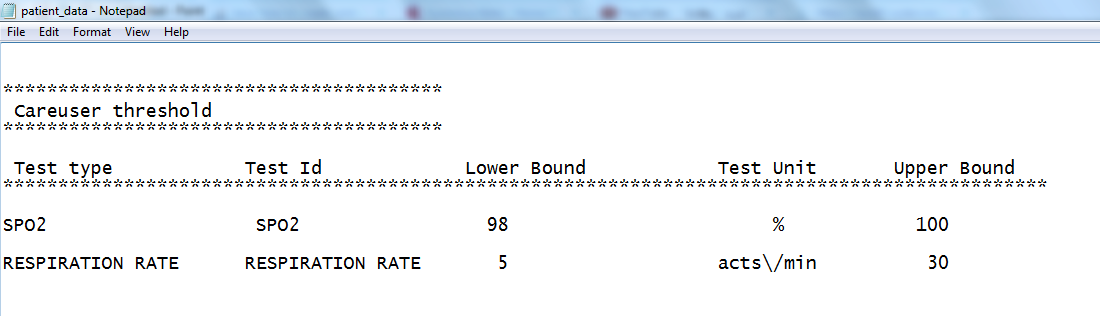
The output of the above code is stored in the text file called patient\_data in careuser available medical test section , and the output is shown in the following figure for careuser (username = Careuser1) , the output contains the careuser medical test names , medical device name, and the medical device ID which is used by the doctor to perform the medical tests for the careuser and we have the test represented as image like(ECG,BGM, BCM…. etc):



Each careuser has medical tests and each one of them has minimum value and maximum value and measure unit , so to retrieve these threshold for each careuser we should use this web service address (MedicalService.svc/Rest/GetMedicalThresholds?username=) that allows us to get the threshold of specific careuser from the telemedicine server as the following code shows:

|  |
| --- |
| //Returns all thresholds set for the careuser  HttpGet httpget6= **new** HttpGet(server\_address + medical\_service[2] + user); CloseableHttpResponse response5 = client.execute(httpget6, context1); BufferedReader rd5 = **new** BufferedReader (**new** InputStreamReader(response5.getEntity().getContent()));    String line5="";  String [] patient\_threshold;  String [] patient\_threshold\_details;  **int** lbound;  String m\_unit;  String test\_name;  String testid; **int** ubound;  **while** ((line5 = rd5.readLine()) != **null**) {    patient\_threshold= line5.split("\\}\\,\\{");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  bw.append(" Careuser threshold ");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  **if**(patient\_threshold.length > 1) {  //write the careuser threshold in their ouput file  bw.newLine();  bw.append(" Test type\t Test Id \t Lower Bound \t Test Unit \t Upper Bound" );  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  **for** (**int** Patient\_M\_tests=0; Patient\_M\_tests<patient\_threshold.length; Patient\_M\_tests++) {  patient\_threshold[Patient\_M\_tests]=patient\_threshold[Patient\_M\_tests].replace("{", "").replace("[", "").replace("}", "").replace("]", ""); patient\_threshold\_details=patient\_threshold[Patient\_M\_tests].split(",");  lbound= Integer.*parseInt*(patient\_threshold\_details[0].replace("\"LowerBound\":", ""));  m\_unit= patient\_threshold\_details[1].replace("\"MedicalDataType\":\"NormUnit\":", "").replace("\"", "");  test\_name= patient\_threshold\_details[2].replace("\"TypeName\":", "").replace("\"","");  testid= patient\_threshold\_details[3].replace("\"MedicalDataTypeId\":", "").replace("\"","");  ubound= Integer.*parseInt*(patient\_threshold\_details[4].replace("\"UpperBound\":", ""));    bw.newLine();  bw.append(test\_name+"\t" + testid +"\t"+ lbound +"\t"+ m\_unit +"\t" + ubound);  bw.newLine();  }  }  **else**  {  bw.newLine(); bw.append("No available threshold for this careuser");  bw.newLine();  }  } |

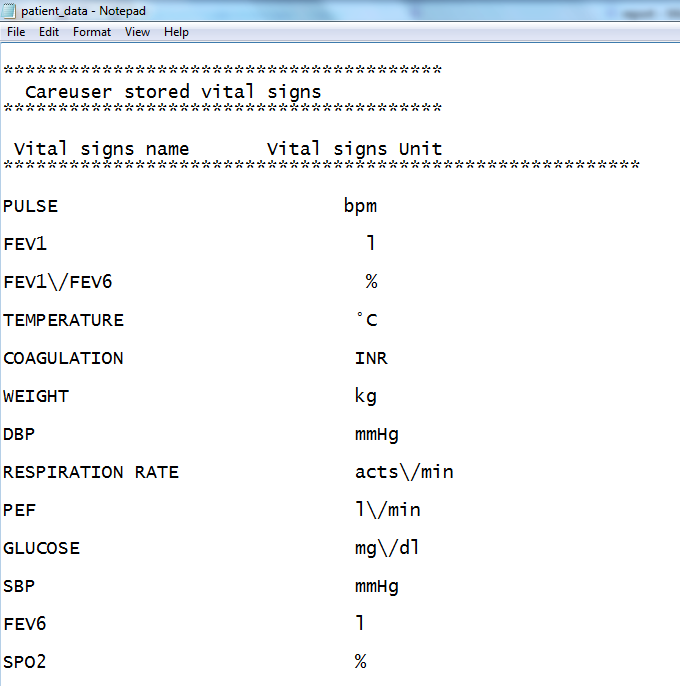
The output of the above code is stored in the text file called patient\_data in careuser threshold section, and the output is shown in the following figure for careuser (username = Careuser1) , the output shows the Test type , Test Id, the test threshold (Lower Bound, Upper Bound), and the Test Unit:



In the following code we will retrieve all Available vital sign for specific careuser through passing the careuser username to the web service, and we should use this web service address (MedicalService.svc/Rest/GetAvailableVitalSignsForUser?username=) :

|  |
| --- |
| //Returns stored vital signs for the user  HttpGet httpget8= **new** HttpGet(server\_address + medical\_service[1] + user);  CloseableHttpResponse response7 = client.execute(httpget8, context1);  BufferedReader rd7 = **new** BufferedReader (**new** InputStreamReader(response7.getEntity().getContent()));    String line7="";  String [] patient\_vatil\_signs; String [] patient\_vatil\_signs\_details;  String vatil\_s\_unit; String vatil\_s\_name;  //Extract the careuser Available vitalsigns from the Buffer reader and clean it to be clear  **while** ((line7 = rd7.readLine()) != **null**) {    patient\_vatil\_signs= line7.split("\\}\\,\\{");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  bw.append(" Careuser stored vital signs ");  bw.newLine();  bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine();  **if**(patient\_vatil\_signs.length > 1) {  bw.newLine();  bw.append(" Vital signs name\t Vital signs Unit " ); bw.newLine();    bw.append("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  bw.newLine(); **for** (**int** Patient\_v\_signs\_index=0; Patient\_v\_signs\_index<patient\_vatil\_signs.length; Patient\_v\_signs\_index++){  patient\_vatil\_signs[Patient\_v\_signs\_index]=patient\_vatil\_signs[ Patient\_v\_signs\_index].replace("{", "").replace("[", "").replace("}", "").replace("]", "");  patient\_vatil\_signs\_details=patient\_vatil\_signs[Patient\_v\_signs\_index].split(",");  vatil\_s\_unit= patient\_vatil\_signs\_details[0].replace("\"NormUnit\":", "").replace("\"", "");  vatil\_s\_name= patient\_vatil\_signs\_details[1].replace("\"TypeName\":", "").replace("\"","");  bw.newLine();  bw.append(vatil\_s\_name +"\t" + vatil\_s\_unit ); bw.newLine();  } }  **else** {  bw.newLine();  bw.append("No available Vital signs for this careuser"); bw.newLine();    }  }  } |

The output of the above code is stored in the text file called patient\_data in careuser stored vital signs section, and the output is shown in the following figure for careuser (username = Careuser1) , the output shows the Vital signs name , and the Vital signs Unit:



In the following code we will retrieve all Available vital sign for specific careuser through passing the careuser username to the web service, and we should use this web service address (MedicalService.svc/Rest/GetAvailableVitalSignsForUser?username=) :

|  |
| --- |
| //Returns all medical alarms (values that exceeded the upper or lower bound of a threshold).  HttpGet httpget7= **new** HttpGet(server\_address + medical\_service[3]);  CloseableHttpResponse response6 = client.execute(httpget7, context1);  BufferedReader rd6 = **new** BufferedReader (**new** InputStreamReader(response6.getEntity().getContent()));    String line6="";  String [] medical\_alarms;  **while** ((line6 = rd6.readLine()) != **null**) {    //Create file in D:/Careuser\_data/medical\_alarms.txt to write the medical alarms inside it  File M\_alarm = **new** File("D:/Careuser\_data/medical\_alarms.txt");  // if file doesn't exists, then create it    **if** (!M\_alarm.exists()) {  M\_alarm.createNewFile();  }    FileWriter M\_alarm\_fw = **new** FileWriter(M\_alarm.getAbsoluteFile());  BufferedWriter M\_alarm\_bw = **new** BufferedWriter(M\_alarm\_fw);  medical\_alarms= line6.split("\\}\\,\\{");    **if**(medical\_alarms.length > 1) {  **for**(**int** m\_alarms\_index=0;m\_alarms\_index<medical\_alarms.length;m\_alarms\_index++){  M\_alarm\_bw.newLine();  M\_alarm\_bw.append(medical\_alarms[m\_alarms\_index]);  M\_alarm\_bw.newLine();  }  }    **else**  {    M\_alarm\_bw.append("No available Medical Alarms for any careuser");  M\_alarm\_bw.newLine();  }  //Close the M\_alarm buffer writer  M\_alarm\_bw.close();  }  } |

The output of the above code will return all medical alarms values that exceeded the upper or lower bound of a threshold.

References