5-1-2013

Real-Time Package Monitoring System using RFID And GPS Technologies

Vivek Ashwin Raman

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
Real-Time Package Monitoring System using RFID And GPS Technologies

By

Vivek Ashwin Raman

Project submitted in partial fulfillment of the requirements for the degree of Master of Science in Networking and System Administration

Department of Information Sciences and Technology

Rochester Institute of Technology

B. Thomas Golisano College

of Computing and Information Sciences

5/1/2013
Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in Networking and System Administration

Project Approval Form

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Vivek Ashwin Raman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>Real-Time Package Monitoring System using RFID And GPS Technologies</td>
</tr>
</tbody>
</table>

Project Committee

<table>
<thead>
<tr>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tae Oh</td>
<td></td>
<td>5-1-2013</td>
</tr>
<tr>
<td>Changfeng Ge</td>
<td></td>
<td>5-2-2013</td>
</tr>
</tbody>
</table>

Committee Member
Abstract

The process of developing a coordinated system that encloses and protects products during their lifecycle of production, storage, transportation and sale is called packaging. During this lifecycle, packages are always under the risk of being damaged or spoiled due to unexpected changes in environmental factors such as changing temperature, humidity and vibration etc. The goal of this project is to develop a highly efficient and low cost real-time package monitoring system that could facilitate to timely locate and detect the abnormal environmental issues such as high temperature and high vibration of packages while in storage and in transportation. The research uses the technology of Radio-Frequency Identification (RFID) and Global Positioning System (GPS) to timely obtain the vital parameters of a package to help maintain the quality and keep track of packages adhering to the legal requirement from the government. The system could therefore significantly minimize the loss that the packaging industry faces due to damage/spoilage during the product lifecycle and also assist in developing enhanced package designs to prevent such damage in the future. Extensive experiments and testing were also performed to evaluate the feasibility and performance of the system.

Keywords

Package Monitoring, Radio frequency Identification, Global Positioning system, Client server architecture, Android application, amazon web services
TABLE OF CONTENTS

Abstract ........................................................................................................ 2
List of Figures ............................................................................................. 5
List of Tables ............................................................................................. 6
Introduction ................................................................................................ 7
Related Work ............................................................................................. 8
  Radio Frequency Identification ................................................................. 8
  Global Positioning System .......................................................................... 11
Scope and Objective .................................................................................. 11
Document Overview .................................................................................... 14
Implementation and Methodology ............................................................... 14
  Client ........................................................................................................ 16
  Server ....................................................................................................... 22
    Application Server .................................................................................. 23
    Database Server ..................................................................................... 25
    Web Server ............................................................................................. 28
  Android Application .................................................................................. 30
Results ......................................................................................................... 32
  Client ........................................................................................................ 32
  Server ....................................................................................................... 34
    Application Server .................................................................................. 34
    Database Server ..................................................................................... 38
    Web Server ............................................................................................. 40
List of Figures

Figure 1 - General architecture used to implement this project .................................. 12
Figure 2 - Detailed architecture of the system .......................................................... 16
Figure 3 - Screen shot of the code snippet on the client application ....................... 17
Figure 4 - Devices used for the implementation on the client side .......................... 18
Figure 5 - Client side set up with all the devices connected and the client application waiting to send data to the server ...................................................... 20
Figure 6 - Client side set up with all the devices connected and the client application working to send data to the server ................................................... 21
Figure 7 - Client side application working to send data to the server ........................ 22
Figure 8 - Server side application working to receive data from the client .................. 24
Figure 9 - Data reception on server side .................................................................... 25
Figure 10 - Database table structure on the database server ..................................... 26
Figure 11 - XML structure that is exchanged between the android application and the restful web service ............................................................................ 29
Figure 12 - Code snippet to send and receive http request and response from the android application .................................................................................. 31
Figure 13 - Functional client side windows application that starts to receive data from sensors and send it across to the application server ........................................... 33
Figure 14 - Network setup of the application server to which all client application send data .................................................................................................. 34
Figure 15 - Application server code snippet capable of accepting multiple client TCP connections ............................................................................................... 35
Figure 16 - Application server data reception every few seconds .................................. 36
Figure 17 - Application server handling GPS unavailable exception case .................. 37
Figure 18 - Data being inserted by the application server to the
truck information table in database server .............................................................. 38
Figure 19 - Data being inserted by the application server to the
device information table in database server .......................................................... 39
Figure 20 - Network and the processor information of the Amazon EC2
instance that is used as the database server and the restful web service .................. 40
Figure 21 - Android application login screen where the package
identification number is entered to get real time package information ................. 41
Figure 22 - Location data plotted using Google maps and the package
information such as temperature, vibration and time ......................................... 42

List of Tables

Table 1 - Columns of the truck information table in the database ............................... 27
Table 2 - Columns of the device information table in the database ......................... 27
1. Introduction

Packaging is the process of enclosing products to protect it from external damage during the process of storage, transportation and sale. A wide variety of products such as hazardous materials, industrial materials, food and medicines etc. are being packaged and transported all over the world through various means. The packaging and transportation industry holds revenue of $633 billion per year as of 2012 and is likely to increase by $100 billion each succeeding year. United States holds close to a 22% of share in this industry, which is the largest among all over the world, and is being followed by China [12]. This industry has been facing a number of challenges that has significantly impacted profits. The factors that contribute to these losses are the impact of changes in landscape, changes in lifestyle and demographic patterns, changes in environmental conditions, and changes in domestic as well as global trends. Due to this, packages are always in the risk of being damaged or spoiled as environmental factors (such as temperature, humidity, vibration and air pressure) change during their storage and transportation. These external factors do play a significant role in contributing to the enormous loss that the packaging and transportation industry face every year [11]. This as a result has contributed to wide spread research in different fields that could assist in managing and developing suitable techniques to combat these challenges.

Although factors such as changes in environmental conditions and variations in landscape cannot be controlled, certain techniques could be devised to overcome these challenges by repetitive analysis and through technological improvements [10].

The need for using information technology has played a primary role in many industries like automotive, telecommunication, and medicine by assisting in storage and processing of their critical data. The packaging and transportation industry could minimize a lot of the above-
mentioned challenges by using the information technology to timely obtain package related
information. The need for obtaining real-time data such as temperature, vibration, and humidity
of packages during storage and while transportation are so valuable that it would largely assist in
determining the condition of packages closely time to time to prevent damage, spoilage and even
theft thereby minimizing the huge losses that this industry incur from these factors every year.
For the sake of maintaining the quality of packages and adhering to the legal requirement of the
government, there is a need to develop a feasible, inexpensive and efficient real-time monitoring
system that could facilitate the package-shipping companies to timely locate and detect abnormal
environmental issues such as high temperature and high vibration of packages during
transportation [23].

2. Related Work

The concept of using radio frequency identification tags in tracking has been there for a few
decades starting from the time of World War 2. Since then, this technology has been used in a
variety of applications such as luggage tracking system in airports, electronic toll collection
systems, and even by a lot of retail companies to keep track of their commodities in sale and in
warehouse [24].

2.1 Radio frequency Identification (RFID)

Radio frequency identification is the technology that uses wireless radio wave
propagation to transfer electronically stored data from a RFID tag to a RFID reader [1]. The
wireless transmission occurs in the frequency range between 120 kHz -10.1 GHz and the tags
can be read from up to several meters away. RFID technology could primarily be classified into
two forms: active and passive [2]. Active RFID technology uses a localized power source on the tag that enables information to be transmitted periodically from the tag to the reader [4]. Passive tags, on the other hand, do not contain localized power source and instead receive triggers from the RFID reader that enable the tags to send information to the reader. Although both active and passive RFID technology is being widely used, the usage largely depends on the kind of application that it supports. RFID technology supports a list of applications that include but not limited to product tracking, inventory management, transportation payments, healthcare, logistics and payment by mobile phones etc. [5].

RFID technology poses a lot of benefits as opposed to barcode technology due to fact that almost all RFID readers receive tag information automatically without any manual reading or human involvement. Most tags have a unique identification number that is stored in the non-volatile memory of the tag [3]. The tag equipped with an RF transmitter and receiver sends and receives encoded radio signals to the reader, and the reader would in turn use a RFID middleware or software to exchange this information to a computer or storage system. RFID tags also come with built in sensors that allow environmental parameters such as temperature, air pressure, vibration and humidity to be monitored periodically.

2.2 Global Positioning System (GPS)

Global positioning system is the technology used to obtain the location co-ordinates such as latitude and longitude of a particular location. It is provided by the NAVSTAR GPS satellites that move in the orbit of Earth at 7500 miles above the surface. These satellites constantly provide the their own ephemeris parameters and times that are received by the GPS receiver
which then compute the location co-ordinates based on the Triangle algorithm present inside the GPS receivers.

The GPS technology was found in the year of 1973 by the United Stated Department of Defense (US DoD) to overcome the limitations of navigation systems that existed before the GPS technology. Since then, this technology has been used almost everywhere to assist in navigation and tracking [25].

As such a number of technologies are used in tracking and navigation which are of great use individually, but when such technologies are combined, it serves a larger purpose in the field of packaging and transportation. Critical packages such as defense equipment, petrochemical products, and biochemical products are constantly shipped using roadways that need to be instantly protected from theft or damage. Food transportation industry incurs biggest losses as food spoilage and damage largely happen due to varying environmental conditions while in transit and obtaining temperature information could help prevent spoilt food from being sold at retail outlets as it can lead to life threatening consequences. The Space research and defense research organizations like NASA and Department of Defense would always monitor and analyze various vital statistics such as temperature variation, vibration and moisture levels about their equipment in transit. The analysis result could greatly improve the equipment thermal and damage protection system [21].

The application can also be modified and installed at hospitals and homes of patients who have undergone recent surgeries with miniature RFID sensors placed across the human body to analyze the temperature and other vital stats, which would be of great benefit to doctors and caregivers [3]. The system could be used even by the tourism industry to closely monitor the
location of their vehicles and people in transit to enhance the safety and disaster management [22].

Although several products exist in the market that has the same idea of real-time tracking, there is a significant cost required to buy such products and installing them. The idea of developing a software application that could be installed on highly prevalent and used smart phones and tablets, capable of receiving vital statistics and be available to an end user with almost no cost would be most welcome.

### 3. Scope and Objective

The solution focuses on obtaining real-time information of vital factors such as temperature, vibration and location about critical packages (chemicals, medicine, electronic, defense equipment etc.) while in storage and in transit by integrating RFID, GPS and 4G technologies and thereby storing this information on a central server. The data thus obtained would then be visualized in useful ways for the end user community as well as used to devise techniques to enable a tracking mechanism for the transportation and logistics companies to closely monitor the packages shipped using their resources. The data could further be used by packaging companies to analyze and bring about changes in packaging designs and methodologies based on the route that a package would be sent to prevent it from damage due to external environmental conditions.

The solution makes use of a set of active RFID tags with inbuilt temperature and vibration sensors placed inside critical packages within a truck that constantly records the temperature and the vibration of the material inside these packages.
An USB RFID reader attached to a laptop/tablet/smart phone, which constantly reads the temperature and the vibration data from these sensors, placed inside the packages and sends this data to the client application running on the computing device (laptop/tablet/smart phone). A GPS device also attached to the computing device that would constantly send the GPS co-ordinates of the truck to the application running on the computing device. The location and the sensor data are combined and sent over to the server at a remote location using a 3G or 4G network while the packages are in transit.

**Diagram:** General architecture used to implement this project

---

*Figure1. General architecture used to implement this project*
The server is accessed over the Internet through an android application developed for tablets/smart phones to view and monitor the temperature, vibration and location information about their critical packages in real time.

The objective is to develop a client application on a laptop/tablet that is capable of retrieving the sensor information and location co-ordinates from the RFID reader and the GPS device attached to the client computing device which further sends it over to a remote server over a 4G network. In real world, a truck that transports multiple packages would have a single client computing device (with RFID reader, GPS device and 4G USB connect) and multiple sensor tags (temperature, vibration etc.) inside the packages. A server application would also be developed that is capable of accepting multiple client connections to receive package related information in real-time and immediately put up this information onto the database on amazon web services.

A web service is also created on the same instance as that of the database on cloud that enables data on the database to be accessed by applications developed on smart phones and tablets. The database would be constructed using Microsoft SQL server 2008 framework and the web service would be enabled on top of Apache Tomcat 6 and uses JAX-WS (Java API for XML exchanges-Web Services) framework.

For the end user community to view the package related information, an android application that would enable users to input the package ID associated with their package to receive real-time updates about their package location, temperature, vibration with date and time is also being developed. The application would be built with Google maps which would enable users to view the location of the package on Google maps as well.
4. Document Overview

This section will present the flow of the document of the master project titled real time package monitoring system using RFID and GPS technologies. The first section is the abstract that in brief explains the project, its significance, intended results and the obtained results. The next section would be the introduction followed by the related work. Both these sections give details about the background of the project. This is followed by the scope and objective of the project that explains on what are the reasons for taking up such a project topic. The next section is the implementation and methodology which explains every component of the entire implementation and how they are made to work. The following section goes over the results obtained from the implementation followed by the conclusion. The next section would be future work that explains on how the project could be worked upon for other utilities followed by the last section which is the references showing the material that had to read and understood to work on this topic.

5. Implementation and Methodology

The system is primarily built using a client-server model. The client side hosts the sensor tags and the GPS device that is responsible for collecting vital information such as temperature and vibration. The client application then periodically sends this information to the server at a remote location. The server side is responsible for grouping information collected from different client applications and storing it in a relational database. The server side is primarily divided into 3 parts namely

a) Application Server b) Database Server c) Web Server
The client could basically be any general purpose tablet or laptop. An android application that would enable end users to view the current package related information such as temperature, vibration and location (with inbuilt Google maps) is also developed. The android application would use the web service on the web server to get real-time data about a particular package based on the package ID.

The system uses a single server multi-client, multi-threaded architecture, for the purpose of allowing the application server to accept connections from all genuine client applications to feed in data that it has collected which makes the package monitoring to be real-time without any time constraints of server availability. The following diagram shows the detailed architecture:
TCP connection across the client and server to exchange sensor and location data.

- INSERT Temperature, Vibration and Location Information into database
- Access Data pertaining to a particular package from external applications (Android App) from the database through web service.

**Figure 2 - Detailed architecture of the system**

5.1 Client

The client would basically consist of a computing device such as a tablet or a laptop connected with an RFID reader that communicates with RFID sensor tags to obtain sensor information and a GPS connector to obtain geographical information. The client device would...
host a client application that would establish a TCP connection to the Application server and
would thereby combine and send the RFID sensor information along with the GPS information
periodically as and when it gets the data from the sensor tags.

The following screen shot shows the code on the client application to establish the TCP
connection with the application server.

Figure3 - Screen shot of the code snippet on the client application

The sensor tags are varied in type and could send information such as temperature,
vibration, pressure, humidity etc. depending on its type. The following technologies namely the
Radio frequency Identification (RFID) and Global positioning system (GPS) are used to enable
the client to get periodic information via the RFID reader and the GPS device.
The above picture shows the temperature sensor RFID tags, the USB RFID reader, AT & T 4G USB connect to relay information over a TCP connection and a serial connector Garmin GPS device with a cigarette lighter power supply. The GPS connector is connected to a Belkin Serial-USB adapter to connect the device to a computing device such as a laptop or tablet.

The main focus of the project is to obtain real-time environmental information about critical packages transported by the logistics companies for which active RFID sensor tags and readers manufactured by GAO-RFID Corporation is used [5]. GAO’s active RFID reader operates at a frequency of 2.45 GHz and has a radio field range of 15-20 meters [6]. The reader uses 0.18 µm CMOS IC technologies for low power consumption and could be connected to the
computer using a standard USB 2.0 interface [7]. The reader operates under two modes: direct and buffer. Direct mode sends real-time information periodically to the connected computer whereas buffer mode stores up to 100-tag information before being transferred to the computer memory. The tags operate at the same frequency as the reader and are built in with temperature sensors that could track temperature between -58F to +302 F. The tag ID, temperature and the timestamp information are sent in real time from the tags in and around 100 meters to the USB RFID reader. This information could further be processed to assess the condition of numerous packages stored and transported by logistics companies on a day-to-day basis.

Figure 5 - Client side set up with all the devices connected and the client application working to send data to the server

For retrieving the real-time geographic information of package, a wired 18x-PC GPS from the Garmin International Inc. would be used [15]. The GPS uses a DB-9 connector to
interface the serial port of computer and receives power through a 12-volt cigarette lighter adapter. The GPS accepts RS-232 level inputs and transmits voltage levels that range from 0 to 5 RS-232 polarities. The GPS is a WAAS-Enabled GPS that could continuously track and utilize multiple satellites for fast and accurate positioning every 1-second [15]. Furthermore, the factory configuration of the GPS ensures that the output of GPS is the NMEA-0183 data, which follows the national standard defined by US-based National Marine Electronics Association. Since this project requires us to get the latitude and longitude information of the package, in the NMEA 0183 data as the output of GPS, Global Position System Fix Data ($GPGGA$)[15] is segmented to retrieve the required geographical information.

Figure 6 - Client side set up with all the devices connected and the client application working to send data to the server
The below screen shot shows the client application that receives and processes sensor and GPS data, establish the TCP stream and sends data to the server.

![Client application screenshot]

Figure 7 - Client side application working to send data to the server

5.2 Server

The server side can be divided into the following parts:

5.2.1 Application Server

The application server is a stand-alone server that runs the server side application to accept TCP connections from various clients. The server works on a multi-client, multi-threaded
architecture and would be capable of accepting TCP connections from any number of clients and further process the data obtained that will then be inserted into the database. All the information obtained by the server would instantly be stored into the database without any temporary storage buffer that makes the application server highly efficient and could be scaled easily. The practice of data validation at the server end and then inserting data into the database would prevent the client application to directly talk to the database and thereby abstracting the data model on the database and prevents SQL attacks.

The application server code is written using Java and opens the socket and a specified port in which it accepts client TCP connections. The following screen shot shows the server code with data reception from one of the client application.
The data is obtained in a specific format and is stored as objects and passed on to the database connection class which validates the data and then inserts it into the relational database. The following picture shows the data reception at the server end every few seconds as it is received from the client application.

Figure 9 - Reception of data on the server end such as latitude, longitude, temperature, vibration and time

5.2.2 Database Server

Amazon web service is used to host the relational database that stores sensor and location information. The Amazon EC 2 instance is installed with Microsoft SQL server 2008 R2 and consists of primarily two database tables, one to have truck related information with latitude,
longitude and time data and the other with the sensor related information for a particular package with temperature and vibration data.

The following screen shot shows the database tables in SQL server.

![Database table structure on the database server](image)

Figure 10 - Database table structure on the database server
The `dbo.truck_information` table consists of

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>truck_id as the primary key</td>
<td>each truck is identified by a unique truck identification</td>
</tr>
<tr>
<td>device_id as the foreign key</td>
<td>each package consists of a unique device identification</td>
</tr>
<tr>
<td>latitude</td>
<td>defines the latitude of a particular truck at a specified time</td>
</tr>
<tr>
<td>longitude</td>
<td>defines the longitude of a particular truck at a specified time</td>
</tr>
<tr>
<td>time</td>
<td>defines the time sent from the client application each time it collects the sensor and Geographical location data</td>
</tr>
</tbody>
</table>

Table 1 - Columns of the truck information table in the database

The `dbo.device_info` table consists of

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_id as the primary key</td>
<td>each package consists of a unique device identification</td>
</tr>
<tr>
<td>truck_id as the foreign key</td>
<td>each truck is identified by a unique truck identification</td>
</tr>
</tbody>
</table>
Sensor ID

Each sensor tag is identified by type identification, e.g.: temperature, vibration etc.

Sensor Type

Each sensor ID is mapped to a sensor type

Sensor Value

Temperature, vibration, etc. values

Time

Defines the time sent from the client application each time it collects the sensor and geographical location data

Table 2 - Columns of the device information table in the database

The application server gets hold of the package related information, does the validation and inserts the data into these tables such that all the truck related information such as latitude, longitude, time and package identification would be stored in the truck information table. The package vital information such as sensor identification, sensor type (which could be temperature, vibration), sensor values and the time data would be inserted into the device information table.

5.2.3 Web Server

The web server’s primary role is to fetch data from the database to display it on a website or to give it to a variety of applications that would want to access package and truck related information. Since it is highly efficient to integrate both the database and the web service on the same Amazon EC2 (Elastic Computing) instance, hosting a Windows server instance with SQL server 2008 R2 that would host the corresponding database and its tables and a web server running apache tomcat 6 would also be installed in order for the server instance to support web services. Using a web service is highly preferable as it could be used by any end user application.
to access the web service to view package specific data in the database. The web server with Tomcat 6 is a rest interface that supports HTTP calls (GET and POST) from applications to get access to package specific data. JAX-WS (Java API for XML exchange- Web service) is an Oracle J2EE framework that is used to program the web service. The framework works on the concept of MVC (Model-View-Controller) and all the data access happens at the model component of the framework. The controller plays the primary role of routing incoming HTTP request to the model to retrieve data using suitable SQL queries and hand back the data which is again sent to XML constructor classes to generate the corresponding XML. This XML is sent back as the response to the external clients (Android/ iOS applications) that do a HTTP request.

The following figure shows the Screenshot of the XML request and response:

![XML Request Response](image.png)

Figure 11 - XML structure that is exchanged between the android application and the restful web service
5.2.4 Android Application

There is always a need for applications to be supported on mobile devices such as tablets and smart phones as most people would want data to be available to them irrespective of their location. Any end user would also want to have the current knowledge of package related information that he/she has shipped on his mobile device for which an android application would be developed. The android application would use the web service on the Amazon cloud to get hold of current package data for a particular package ID associated with the package that the user was assigned at the time of shipping. The application would keep fetching the real time data that would include the status of the package, temperature, vibration (high, low), location information (with Google map view to view the location on maps) and the time of the update. This application would be of real benefit as end users can always have detailed information about their shipped package on their mobile devices through the android application.

The application constantly polls the web server using http request with the package ID in the XML. All conversation between the android application and the web server happen using XML exchange. The http response is then processed using XML classes on the application to retrieve data, process this data and send it to the UI to display data in the right format. The android application apart from displaying the temperature, vibration of the package would also use a web view to pass the location co-ordinates as sent from the web server to obtain the Google map view to show the location in human readable format.

The android application also has the MVC (Model View Controller) components with the Main Activity (Android class) that would send http request using XML from XML constructor classes. The http response is received by the Main activity that sent the http request which then
processes the http response and pass on the data needed by the view component. Android by default has its view defined in XMLs as well and has a variety of inbuilt classes (such as buttons, text views, web views and adapters) to define each component of the user interface. All http requests are done using asynchronous threads or tasks that would have the android application respond to the user inputs in parallel with processing http responses and putting up new views with latest data from the server which makes the application highly robust and asynchronous.

The following figure shows the screen shot of the small portion of the android application code that uses an asynchronous task to do http post with a request XML.

![Code to poll rest interface from android application](image)

**Figure 12 - Code snippet to send and receive http request and response from the android application**
6. Results

This section presents all the observed results as part of the implementation of the project. The results can be divided into sections based on the different components of the system. The proposed architecture was divided into the following sections which are:

1. Client
2. Application server
3. Database server
4. Web server
5. Android application

6.1 Client:

The client side is a windows application that is capable of obtaining information (such as temperature and vibration) from active RFID sensor tags every few seconds and combine this with the location information (latitude and longitude) from the GPS device and send it over to the application server. The following screen shot shows the client application receiving sensor data and opening up TCP stream with the application server to send data.
Figure 13 - Functional client side windows application that starts to receive data from sensors and send it across to the application server
6.2 Application server:

The application server has the following IP which is used by the client applications to access the server to initiate a TCP connection request. The IP address could be replaced with the domain name in the future that could prevent the client applications to be affected when the server IP is changed. The screenshot of the server IP configuration is as below.

![Network setup of the application server](image)

Figure 14 - Network setup of the application server to which all client applications send data
The application server opens up the socket and the port number 11003 to accept incoming TCP connections requests from client applications and also receive and process the data received. The following screenshot shows the part of the server code to accept multiple connections streams on the same port.

```
package sample_server;

public class Sample_server {
    private static int port=11003, maxConnections=0;
    // listen for incoming connections and handle them
    public static void main(String[] args) {
        int i=0;
        try {
            ServerSocket listener = new ServerSocket(port);
            Socket server;
            while((i++ < maxConnections) || (maxConnections == 0)) {
                doComesConnection();
                server = listener.accept();
                doCmns com_c = new doComm(server);
                Thread t = new Thread(com_c);
                t.start();
            }
            System.out.println("Exception on socket: " + low); low.printStackTrace();
        }
    }
}
```

Figure 15 - Application server code snippet capable of accepting multiple client TCP connections
The server application when started would continue to run indefinitely until the server is manually stopped which allows for high availability of the server for the client applications. The following screenshot shows the server receiving data from one of the client application. The data is in the order of truck id, package id, sequence number (to identify each data is unique), latitude, longitude, temperature, vibration and time.

![Figure 16 - Application server data reception every few seconds](image-url)
The server performs a data validation before the data is inserted into the database. As an example, the GPS device could sometimes have a weak signal which makes the GPS device send a weak signal message.

This exceptional case is handled on the server side and such data is not put into the same database tables where all the correct data goes in. The screenshot of the description is as above in figure.

Figure 17 - Application server handling GPS unavailable exception case
6.3 Database Server:

The data obtained from the application server gets inserted into the database named packaging tracking on the Amazon EC2 instance. The database holds two tables, one with the truck information and the other with the sensor information. The following screenshot shows the information inserted by the application server into the truck information table which primarily holds the location information such as latitude and longitude along with the time and the truck id.

![Database Table Screenshot]

Figure 18 - Data being inserted by the application server to the truck information table in database server
The sensor information for a particular package such as temperature and vibration along with the package id and the time are inserted into the device information table. The screenshot of the table is as shown below.

---

Figure 19 - Data being inserted by the application server to the device information table in database server
6.4 Amazon EC2 Instance:

The database and the web service are hosted on an Amazon elastic compute instance. The instance is a Windows Server 2008 R2 with SQL server 2008 running on it as the database engine. It is also installed with Apache tomcat 6 to have web applications deployed on it to make it a rest interface. The following screenshot shows the Amazon EC2 instance information.

Figure 20 - Network and the processor information of the Amazon EC2 instance that is used as the database server and the restful web service
6.5 Android Application:

The android application performs the activity of polling the server to get data from the database through the web service. The application then uses this data and displays it in the user interface in a suitable format for the end user community. The following screen shot shows the login page of the android application where the user puts in a unique package identity assigned to the package to be tracked.

Package Tracking and Vital Information

![Package Tracking and Vital Information](image)

Figure 21 - Android application login screen where the package identification number is entered to get real time package information
The main page of the application shows package information such as temperature, vibration, time and location. The location is received as latitude and longitude is plotted using Google maps to show the human readable location on a web view of the application.

![Google Maps Location](image)

**Figure 22** - Location data plotted using Google maps and the package information such as temperature, vibration and time

### 7. Conclusion

A coordinated system that could ensure the safety and quality of packages was in need to prevent packages from getting damaged/spoiled while in storage and in transportation. An inexpensive and highly efficient real time package monitoring system is developed that uses the Radio Frequency Identification and Global positioning system to obtain vital information.
environmental information around the package such as the temperature, vibration and location that could facilitate in maintaining and detecting abnormal conditions around packages. The entire system manages package related information and stores it to a centralized server that could be used for a variety of useful purposes. Applications on mobile platforms are also developed to ensure that the data is available without locations constraints. To test the solution a real environment is created by setting up the system to collect and store package related information. The results of the experiment demonstrated that by having a low cost real time package monitoring system, package damage is prevented, package quality is maintained and also keeps critical products that are being stored and transported in vigilance. The stored data were also used to plot the variations of vital statistics around the packages over time.

8. Future Work

A number of future enhancements could be added to this project to make it more secure, robust and useful on a longer term. Since the project involves a lot of hardware and software components, there is a need and scope to improve both on the hardware as well as the software side.

8.1 Software Improvements:

Security has not been the primary concentration on the client side as there isn’t any authentication mechanism used by the application server before it opens up the TCP ports and exchange the IP address to the clients. Implementing security would be a really needed enhancement in the future. The project currently uses a windows application on the client side to get sensor data and send it across to the application server, but with more mobile platforms and
operating systems becoming popular, there is a need to develop an android or an iOS application which could be installed on truck driver’s phone or tablet instead of using a custom device for every truck.

The end user application used to view real time sensor and location on Google maps is developed on android platform which opens up the scope to develop end user mobile applications on other operating systems such as iOS and Windows. The end user application also does not have a local database to store the sensor and location data that could be viewed and reviewed as needed. This creates the need to have a light weight database on the end user application to store the last few days of data that could be used by the end user to view history as well as to create graphs that plot the variation in temperature, vibration and location of packages as they move from one place to another. The data collected from various packages could also be used for data mining to analyze and understand the environmental conditions pertaining to different geographical locations that could improve and enhance packaging designs and implement those package designs for packages sent to a location via a particular route.

8.2 Hardware Improvements:

The client side uses radio frequency sensors manufactured by third party companies that put restrictions on how often the sensor data needs to be obtained. The sensors used in the project are bought from a third party vendor and could receive sensor data only after every few seconds. A single breadboard with embedded sensors (temperature, vibration, pressure etc.) custom developed for the purpose of collecting sensor data as and when needed by the client application could make the system highly robust, efficient, real time and minimize the cost of integrating sensors of different size into the client side device.
The client side component uses a global positioning system device to get the latitude and longitude co-ordinates which are combined programmatically with sensor data and streamed via TCP to the application server. By developing client side application on mobile platforms and using mobile devices would curb the usage of GPS devices as most mobile devices come with location detecting hardware inside them by default which could be programmatically obtained and used by the client side application. This reduces the cost and the number of devices integrated on the client side.

Apart from these improvements the system has a fewer other utilities. One of them is to modify the components slightly to install them in factories to get real-time temperature, vibration and other vital parameters from their manufacturing equipment which could be remotely viewed and analyzed to detect equipment performance, malfunction and failure.
9. References:


[3] Loc Ho Venturi, Sunnyvale, CA Melody Moh San Jose State University, San Jose, CA Zachary Walker San Jose State University, San Jose, CATakeo Hamada Fujitsu Laboratories of America, Sunnyvale, CA Ching-Fong Su Fujitsu Laboratories of America, Sunnyvale, CA, A prototype on RFID and sensor networks for elder healthcare: progress report, Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis.


[24] Linda Castro; Samuel Fosso Wamba - AN INSIDE LOOK AT RFID TECHNOLOGY

Server Side Code in Java

package sample_server;

import java.io.*;
import java.net.*;
import java.security.*;
import java.util.ArrayList;
import java.util.List;

public class RFID_Server {
  private static int port = 11003, maxConnections = 0;

  // Listen for incoming connections and handle them
  public static void main(String[] args) {
    int i = 0;
    try {
      ServerSocket listener = new ServerSocket(port);
      Socket server;
      while ((i++ < maxConnections) || (maxConnections == 0)) {
        doComms connection;
        server = listener.accept();
        doComms conn_c = new doComms(server);
        Thread t = new Thread(conn_c);
        t.start();
      }
    } catch (IOException ioe) {
      System.out.println("IOException on socket listen: " + ioe);
      ioe.printStackTrace();
    }
  }

  class doComms implements Runnable {
    private Socket server;
    private String line, input;
    doComms(Socket server) {
      this.server = server;
    }

    public void run() {
      input = null;
      try {
        // Get input from the client
        DataInputStream in = new DataInputStream(server.getInputStream());
        BufferedReader receiveRead = new BufferedReader(
          new InputStreamReader(in));
        List<String> Values = new ArrayList<String>();
        while ((line = receiveRead.readLine()) != null && !line.equals("quit")) {
          Values.add(line);
        }
      } catch (IOException ioe) {
        System.out.println("IOException on receive: " + ioe);
        ioe.printStackTrace();
      }
    }
  }
// Input: input / line;
boolean b = line.startsWith("Connection");
if (b == true) {
    System.out.println(line);
    continue;
}
String splitValues[] = line.split(",");
for (int i = 0; i < splitValues.length; i++) {
    Values.add(splitValues[i]);
}
try {
    if (Values.size() == 8) {
        Database_Connection.connect_database(Values);
    }
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
System.out.println(Values);
Values.clear();
server.close();
} catch (IOException ioe) {
    System.out.println("IOException on socket listen: " + ioe);
    ioe.printStackTrace();
}

Database Connectivity in Java
package sample_server;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
public class Database_Connection {
    public static boolean connect_database(List<String> package_info) throws ClassNotFoundException {
        Connection con = null;
        String sensor_type = null;
        String sensor_identify = package_info.get(5);
        boolean b = sensor_identify.startsWith("BB");
        if (b == true) {
            sensor_type = "Temperature";
        } else {
            sensor_type = "Vibration";
        }
        String url = "jdbc:jtds:sqlserver://23.23.23.176:1433;DatabaseName=PackageTracking;user=vi vek;password=Asbv1v81988";
try {
    Class.forName("net.sourceforge.jtds.jdbc.Driver");
    con = DriverManager.getConnection(url);
    Statement st = con.createStatement();
    String query1 = "INSERT INTO
{PackageTracking}.[dbo].[device_info] (truck_id, device_id, sensor_id, sensor_value, sensor_type, time) " + "VALUES (" + package_info.get(0) + "," + package_info.get(1) + "," + package_info.get(5) + "," + package_info.get(6) + "," + sensor_type + "," + package_info.get(7) + ");";
    st.executeUpdate(query1);
    String query2 = "INSERT INTO
{PackageTracking}.[dbo].[truck_information] (truck_id, device_id, latitude, longitude, time) " + "VALUES (" + package_info.get(0) + "," + package_info.get(1) + "," + package_info.get(3) + "," + package_info.get(4) + ");";
    st.executeUpdate(query2);
    con.close();
    return true;
} catch (SQLException e) {
    e.printStackTrace();
    return false;
}
/** Servlet implementation class Sensor */
public class Sensor extends HttpServlet {
    private static final long serialVersionUID = 1;
    /**
     * @see HttpServlet#HttpServlet()
     */
    public Sensor() {
        super();
        // TODO Auto-generated constructor stub
    }
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        StringBuffer jb = new StringBuffer();
        String line = null;
        try {
            BufferedReader reader = request.getReader();
            while ((line = reader.readLine()) != null)
                jb.append(line);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        String validate_string = jb.toString();
        XML_Validity xmlvalid = new XML_Validity();
        String device_id = xmlvalid.get_sensorID_from_inbound_xml(validate_string);
        String truck_id = "1";
        try {
            if (device_id != null) {
                Database_Connect db = new Database_Connect();
                SensorData sensordata = db.get_sensor_info_from_database(
                        device_id, truck_id);
                XML_Constructor xmlconstructor = new XML_Constructor();
                String output = xmlconstructor
                        .Sensor_data_XML_create(sensordata); response.getOutputStream().println(output);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}
else {
    String output = "Invalid Package ID";
    response.getOutputStream().println(output);
}
} catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

Sensor Data Object
package com.Packagetracking;

public class SensorData {
    String _packageID;
    String _truckID;
    String _timeValue;
    String _vibValue;
    String _latitude;
    String _longitude;
    String _location;

    public SensorData(){
    }

    public String getPackageID(){
        return _packageID;
    }

    public void setPackageID(String packageID){
        this._packageID = packageID;
    }

    public String getTruckID(){
        return _truckID;
    }

    public void setTruckID(String truckID){
        this._truckId = truckID;
    }

    public String getTimeValue(){
        return _timeValue;
    }

    public void setTimeValue(String timeVal){
        this._timeValue = timeVal;
    }
}
public String getTempValue()
{
    return _tempValue;
}

public void setTempValue(String tempVal)
{
    this._tempValue = tempVal;
}

public String getVibValue()
{
    return _vibValue;
}

public void setVibValue(String vibVal)
{
    this._vibValue = vibVal;
}

public String getLatitude()
{
    return _latitude;
}

public void setLatitude(String lat)
{
    this._latitude = lat;
}

public String getLongitude()
{
    return _longitude;
}

public void setLongitude(String lon)
{
    this._longitude = lon;
}

public String getLocation()
{
    return this._location;
}

public void setLocation(String lat, String lon)
{
    this._location = lat.concat("," + lon);
}

Database Connectivity
package com.Packagetracking;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class Database_Connect {

    // Database connection code

}
private static final String url =
"jdbc:jtds:sqlserver://23.23.88.176:1433;DatabaseName=PackageTracking;user=vivek;pass
word=Ashviv1988";

Connection con = null;

public SensorData get_sensor_info_from_database(String device_id,
String truck_id) throws ClassNotFoundException {

String temp = null;
String vib = null;
String lat = null;
String lon = null;
String time = null;
SensorData sensordata = new SensorData();

try {
    Class.forName("net.sourceforge.jtds.jdbc.Driver");
    con = DriverManager.getConnection(url);
    Statement st = con.createStatement();

    String query1 = "SELECT TOP 1 [sensor_value],[time]
+ " FROM [PackageTracking].[dbo].[device_info]
+ "WHERE [device_id] = " + device_id + ", [sensor_type] = " + "Temperature"
+ " ORDER BY 'time' DESC";

    try {
        ResultSet rsl = st.executeQuery(query1);
        while (rsl.next()) {
            temp = rsl.getString(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    String query2 = "SELECT TOP 1 [sensor_value],[time]
+ " FROM [PackageTracking].[dbo].[device_info]
+ "WHERE [device_id] = " + device_id + ", [sensor_type] = " + "Vibration"
+ " ORDER BY 'time' DESC";

    try {
        ResultSet rs2 = st.executeQuery(query2);
        while (rs2.next()) {
            vib = rs2.getString(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    String query3 = "SELECT TOP 1 [latitude],[longitude],[time] FROM
[PackageTracking].[dbo].[truck_information] WHERE [truck_id] = " + truck_id + ", ORDER BY 'time' DESC";

    try {
        ResultSet rs3 = st.executeQuery(query3);
        while (rs3.next()) {
            lat = rs3.getString(1);
            lon = rs3.getString(2);
            time = rs3.getString(3);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    sensordata.setPackageID(device_id);
    sensordata.setTruckID(truck_id);
    sensordata.setTempValue(temp);
    sensordata.setVibValue(vib);
    sensordata.setLatitude(lat);
    sensordata.setLongitude(lon);
    sensordata.setTime(time);
}

- 55 -
sensordata.setVibValue(vib);
sensordata.setTimeValue(time);
sensordata.setLatitude(lat);
sensordata.setLongitude(lon);
sensordata.setLocation(lat, lon);
con.close();
} catch (SQLException e) {
    e.printStackTrace();
}
return sensordata;

XML Constructor class
package com.PackageTracking;
import java.io.StringWriter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
public class XML_Constructor {
    String Sensor_data_XML_create(SensorData sensordata) {
        try {
            DocumentBuilderFactory docFactory = DocumentBuilderFactory
                    .newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
            // root elements
            Document doc = docBuilder.newDocument();
            Element rootElement = doc.createElement("PackageInfo");
            doc.appendChild(rootElement);
            Element Temperature = doc.createElement("Temperature");
            if (sensordata.getTempValue() == null)
                Temperature.appendChild(doc.createTextNode("null"));
            else
                Temperature.appendChild(doc.createTextNode(sensordata.getTempValue()));
            rootElement.appendChild(Temperature);
            Element Vibration = doc.createElement("Vibration");
            if (sensordata.getVibValue() == null)
                Vibration.appendChild(doc.createTextNode("null"));
            else
                Vibration.appendChild(doc.createTextNode(sensordata.getVibValue()));
            rootElement.appendChild(Vibration);
        }
        return sensordata;
    }
}
Vibration.appendChild(doc.createTextNode("null"));
else
    Vibration.appendChild(doc.createTextNode(sensordata.getVibValue()));
rootElement.appendChild(Vibration);

Element location = doc.createElement("location");
if (sensordata.getlocation() == null)
    location.appendChild(doc.createTextNode("null"));
else
    location.appendChild(doc.createTextNode(sensordata.getlocation()));
rootElement.appendChild(location);

Element time = doc.createElement("time");
if (sensordata.getTimeValue() == null)
    time.appendChild(doc.createTextNode("null"));
else
    time.appendChild(doc.createTextNode(sensordata.getTimeValue()));
rootElement.appendChild(time);

String forward = Transform_to_XML(doc);
return forward;

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
return null;
}

private String Transform_to_XML(Document doc) {
    try {
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(doc);
        StringWriter stringWriter = new StringWriter();
        StreamResult result = new StreamResult(stringWriter);
        transformer.transform(source, result);
        return stringWriter.getBuffer().toString();
    } catch (TransformerException tfe) {
        tfe.printStackTrace();
    }
    return null;
}

XML Validator Class

package com.Packagetracking;

import java.io.IOException;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.CharacterData;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

public class XML_Validity {

    String get_sensorID_from_inbound_xml(String validate_string) {
        DocumentBuilder db;
        try {
            db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            InputSource is = new InputSource();
            is.setCharacterStream(new StringReader(validate_string));
            Document doc = db.parse(is);

            NodeList SensorID = doc.getElementsByTagName("DeviceID");
            Element line = (Element) SensorID.item(0);
            String SensorID_value = getCharacterDataFromElement(line);
            return SensorID_value;
        } catch (ParserConfigurationException e) {
            TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SAXException e) {
            TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    static String getCharacterDataFromElement(Element e) {
        Node child = e.getFirstChild();
        if (child instanceof CharacterData) {
            CharacterData cd = (CharacterData) child;
            return cd.getData();
        }
        return "";
    }
}

RFID Android Application
package com.RFID;
import java.io.IOException;
public class RFID_androidActivity extends Activity {
    private static Handler handler = new Handler();
    private static String MAP_URL = "http://maps.google.com/maps?q=40.730291,-74.007577&z=17";
    private WebView webView;
    private ImageButton back;
    private Button reload, stop_reload;
    static boolean package_info_page = false;
    private Button Login;
    private EditText Request;
    static String http_Response;
    static int refresh_pressed = 0;
    static String URL_to_post = "https://23.23.88.176/Packa geTrackingWebService/Sensor";
    static String http_post_string = null;
    static Runnable runnable;
    static String Lat = "43.1017";
    static String Lon = "-77.6218";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FFATURf_NO_TITLf);
        setContentView(R.layout.activity_rfid_android);
        handler.post(new Runnable() {
            @Override
            public void run() {
                webView.loadUrl(MAP_URL);
            }
        });
        webView.setWebViewClient(new WebViewClient() {
            @Override
            public void onPageFinished(WebView view, String url) {
                if (url.equals(MAP_URL)) {
                    webView.getSettings().setJavaScriptEnabled(true);
                    webView.loadUrl("javascript:var map = L.map('map', {
                        center: ["" + Lat + ", " + Lon + ""],
                        zoom: 17
                    });
                    webView.loadUrl("javascript:");
                }
            }
        });
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                webView.loadUrl("javascript:map.setView(L.map.getCenter());");
            }
        });
        reload.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                webView.loadUrl(MAP_URL);
            }
        });
        stop_reload.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                webView.stopLoading();
            }
        });
        Login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String request = Request.getText().toString();
                webView.loadUrl("javascript:");
            }
        });
        Request.setOnFocusChangeListener(new View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                if (hasFocus) {
                    webView.loadUrl("javascript:");
                } else {
                    webView.loadUrl("javascript:");
                }
            }
        });
        webView.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
                if (motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
                    webView.loadUrl("javascript:");
                } else if (motionEvent.getAction() == MotionEvent.ACTION_UP) {
                    webView.loadUrl("javascript:");
                } else if (motionEvent.getAction() == MotionEvent.ACTION_MOVE) {
                    webView.loadUrl("javascript:");
                }
                return true;
            }
        });
    }
}
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, 
WindowManager.LayoutParams.FLAG_FULLSCREEN);
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
mainpage();
}

public void mainpage()
{
setContentView(R.layout.main);
Login = (Button) findViewById(R.id.trackpackage);
Login.setOnClickListener(send_activate_post_command);
Request = (EditText) findViewById(R.id.Request_data);
private class PostIO extends AsyncTask<Void, Void, String> {
    @Override
    protected String doInBackground(Void... params) {
        HttpContext localContext = new BasicHttpContext();
        HttpPost httpPost = new HttpPost(URL_to_post);
        httpPost.addHeader("Content-Type", "application/lxml");
        StringEntity Request;
        try {
            Request = new StringEntity(http_post_string, "UTF-8");
            Request.setContentType("application/lxml");
            httpPost.setEntity(Request);
        } catch (UnsupportedEncodingException el) {
            TODO Auto-generated catch block
            el.printStackTrace();
        }
        String text = null;
        try {
            HttpResponse response =
            HttpUtils.getNewHttpClient().execute(httpPost, localContext);
            HttpEntity entity = response.getEntity();
            text = getASCIIContentFromEntity(entity);
        } catch (Exception e) {
            return e.getMessage();
        }
        return text;
    }
    protected void onPostExecute(String results) {
        if ((results != null)) {
            response_package.get_sensor_values_from_response_xml(results);
            try {
                setupWebView(lat, lon);
            } catch (ParseException e) {
                
            }
        }
    }
}
protected String getASCIIContentFromEntity(HttpEntity entity) throws IllegalStateException, IOException {
    InputStream in = entity.getContent();
    StringBuffer out = new StringBuffer();
    int n = 1;
    while (n > 0) {
        byte[] b = new byte[4096];
        n = in.read(b);
        if (n > 0)
            out.append(new String(b, 0, n));
    }
    return out.toString();
}

private OnClickListener send_activate_post_command = new OnClickListener() {
    public void onClick(View v) {
        if ((Request.getText().length() != 0)) {
            refresh_pressed = 0;
            URL_to_post = "https://23.23.88.176/PackageTrackingWebService/Sensorplus";
            http_post_string = Construct_XML.Build_sensor_XML();
            new PostIO().execute();
            Toast.makeText(RFID_androidActivity.this, "Logged in", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(RFID_androidActivity.this, "Invalid Package ID", Toast.LENGTH_SHORT).show();
        }
    }
};

private OnClickListener Start_fetching_data = new OnClickListener() {
    public void onClick(View v) {
        if (refresh_pressed == 0) {
            startProgress();
            refresh_pressed++;
        }
    }
};

private OnClickListener Stop_fetching_data = new OnClickListener() {
    public void onClick(View v) {
        onStop();
        refresh_pressed = 0;
    }
};
private OnClickListener back_to_login = new OnClickListener() {
    public void onClick(View v) {
        package_info_page = false;
        onStop();
        mainpage();
    }
};

public void startProgress() {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            while (package_info_page == true) {
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        start_posting();
                    }
                });
            }
        }
    }
    new Thread(runnable).start();
}

protected void onStop() {
    package_info_page = false;
    super.onStop();
    handler.removeCallbacks(runnable);
}

protected void start_posting() {
    URL_to_post = "https://23.23.88.176/PackageTrackingWebService/Sensorplus";
    http_post_string = ConstructXML.Build_sensor_XML();
    new PostIO().execute();
    Toast.makeText(RFID_androidActivity.this, "Refresh Done", Toast.LENGTH_SHORT).show();
}

@SuppressLint("SetJavaScriptEnabled")
private void setupWebView(String lat, String lon) throws ParseException {
    setContentView(R.layout.googlemap);
    package_info_page = true;
    Package_identity = (TextView) findViewById(R.id.package_identity);
    Temp = (TextView) findViewById(R.id.temp);
    Vib = (TextView) findViewById(R.id.vib);
    Loe = (TextView) findViewById(R.id.Loe);
    Tim = (TextView) findViewById(R.id.tim);
stop_reload = (Button) findViewById(R.id.stop_reload);
stop_reload.setOnClickListener(Stop_fetching_data);
reload = (Button) findViewById(R.id.reload);
reload.setOnClickListener(Start_fetching_data);
back = (ImageButton) findViewById(R.id.gobackopp);
back.setOnClickListener(back_to_login);
Package_identity.setText(REQUEST.getText().toString());
Temp.setText(" : " + roundOff(response_package.ALL_package_details.get(0)) + " F");
Vib.setText(" : " + roundOff(response_package.ALL_package_details.get(1)) + " G");
String location = response_package.ALL_package_details.get(2);
String array[] = location.split(",");
array[0] = array[0].substring(1);
array[1] = array[1].substring(1);
array[1] = " -" + array[1];
lat = array[0];
lon = array[1];
loc.setText(roundOff(lat) + " , " + roundOff(lon));
Tim.setText(secsToDate(response_package.ALL_package_details.get(3)));
response_package.ALL_package_details.clear();
MAP_URL = "http://maps.google.com/maps?q=" + lat + "," + lon + "&z=16;"
final String centerURL = "javascript:centerAt(" + lat + "," + lon + ");";
webView = (WebView) findViewById(R.id.webview);
webView.getSettings().setJavaScriptEnabled(true);
webView.setWebViewClient(new WebViewClient() {
    @Override
    public void onPageFinished(WebView view, String url) {
        webView.loadUrl(centerURL);
    }
});
webView.loadUrl(MAP_URL);

public String secsToDate(String time) throws ParseException {
    long l = Long.parseLong(time);
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy HH:mm:ss a");
    Date resultdate = new Date(l);
    return sdf.format(resultdate);
}

public String roundOff(String val) {

```java
double d = Double.parseDouble(val);
double roundOff = Math.round(d * 1000.0) / 1000.0;
String roundedVal = Double.toString(roundOff);
return roundedVal;
```

```java
XML Constructor
package com.RFID;
import java.io.StringWriter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.CharacterData;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
public class Construct_XML {
    public static String getCharacterDataFromElement(Element e) {
        Node child = e.getFirstChild();
        if (child instanceof CharacterData) {
            CharacterData cd = (CharacterData) child;
            return cd.getData();
        }
        return "";
    }

    public static String Build_sensor_XML() {
        try {
            DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
            // root elements
            Document doc = docBuilder.newDocument();
            Element rootElement = doc.createElement("Sensor");
            doc.appendChild(rootElement);
            Element TabletID = doc.createElement("DeviceID");
            TabletID.appendChild(doc.createTextNode("1"));
            rootElement.appendChild(TabletID);
```

- 64 -
// write the content into xml file
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StringWriter stringWriter = new StringWriter();
StreamResult result = new StreamResult(stringWriter);
transformer.transform(source, result);
return stringWriter.getBuffer().toString();

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
} catch (TransformerException tfe) {
tfe.printStackTrace();
} return null;

} }  

SSL Connectivity to the Secure Web service

package com.RFID;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security.*;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import org.apache.http.conn.ssl.SSLSocketFactory;

public class EasySSLSocketFactory extends SSLSocketFactory {
SSLContext ssoContext = SSLContext.getInstance("TLS");

public EasySSLSocketFactory(KeyStore truststore) throws NoSuchAlgorithmException, KeyManagementException, KeyStoreException, UnrecoverableKeyException {
super(truststore);
TrustManager tm = new X509TrustManager() {
public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
}
public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
}
public X509Certificate[] getAcceptedIssuers() {
return null;
}
sslContext.init(null, new TrustManager[] { tm }, null);

@override
    public Socket createSocket(Socket socket, String host, int port, boolean autoClose) throws IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(socket, host, port, autoClose);
    }

@override
    public Socket createSocket() throws IOException {
        return sslContext.getSocketFactory().createSocket();
    }

HTTP Utils class

import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security *
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import org.apache.http.conn.ssl.SSLSocketFactory;

public class EasySSLSocketFactory extends SSLSocketFactory {
    SSLContext sslContext = SSLContext.getInstance("TLS");

    public EasySSLSocketFactory(KeyStore truststore) throws NoSuchAlgorithmException, KeyManagementException, KeyStoreException, UnrecoverableKeyException {
        super(truststore);

        TrustManager tm = new X509TrustManager() {
            public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
            }

            public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
            }

            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
        };
    }

    public
}
## XML Data retriever Android Application

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.CharacterData;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

public class response_package {
    static List<String> ALL_package_details = new ArrayList<String>();

    static void get_sensor_values_from_response_xml(String validate_string) {
        DocumentBuilder db;
        try {
            db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            InputSource is = new InputSource();
            is.setCharacterStream(new StringReader(validate_string));
            Document doc = db.parse(is);

            NodeList Temperature = doc.getElementsByTagName("Temperature");
            Element line = (Element) Temperature.item(0);
            String Temperature_value = getCharacterDataFromElement(line);
            ALL_package_details.add(Temperature_value);

            NodeList Vibration = doc.getElementsByTagName("Vibration");
            Element line1 = (Element) Vibration.item(0);
```

- 67 -
String Vibration_value = getCharacterDataFromElement(line1);
ALL_package_details.add(Vibration_value);

NodeList location = doc.getElementsByTagName("location");
Element line2 = (Element) location.item(0);
String location_value = getCharacterDataFromElement(line2);
ALL_package_details.add(location_value);

NodeList Time = doc.getElementsByTagName("Time");
Element line3 = (Element) Time.item(0);
String Time_value = getCharacterDataFromElement(line3);
ALL_package_details.add(Time_value);

} catch (ParserConfigurationException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

} catch (SAXException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}

static String getCharacterDataFromElement(Element e) {
    Node child = e.getFirstChild();
    if (child instanceof CharacterData) {
CharacterData cd = (CharacterData) child;
    return cd.getData();
    }
    return "";
}