Technology Innovation   
& Architecture  
Summer, 2017

**Group Project Name**: Sanbot Inventory Use Case

*Callie Warren|Truman State University|Bachelor’s in Computer Science|2017*

*Chia-wei Chang|University of Texas A&M|Master’s in Computer Science|2017*

*Jaydeep Ingle|State University of New York, Binghampton|Master’s in Computer Science|2017*

*Ryan Brady|University of Central Missouri|Bachelor’s in Computer Science|2016*

*Samantha Stanley|University of Central Missouri|Bachelor’s in Mathematics & Computer Science|2019*

*Utsav Dholakia|University of Texas at Dallas|Master’s in Computer Science|2017*

Contents

[Objective 3](#_Toc490467337)

[Description/Background 3](#_Toc490467338)

[Benefits 3](#_Toc490467339)

[Architecture 4](#_Toc490467340)

[Code-Application 4](#_Toc490467341)

[Location 4](#_Toc490467342)

[Language/Tools 4](#_Toc490467343)

[Design 4](#_Toc490467344)

[Code-Webpage (Inventory page) 6](#_Toc490467345)

[Location 6](#_Toc490467346)

[Language/Tools 6](#_Toc490467347)

[Design 6](#_Toc490467348)

[Code-Server 6](#_Toc490467349)

[Location 6](#_Toc490467350)

[Language/Tools 6](#_Toc490467351)

[Design 6](#_Toc490467352)

[User Experience 7](#_Toc490467353)

[Demonstration 8](#_Toc490467354)

[Requirements 8](#_Toc490467355)

[Set up the Demo 8](#_Toc490467356)

[Webpage 8](#_Toc490467357)

[Server 8](#_Toc490467358)

[Application 8](#_Toc490467359)

[Run the Demo 9](#_Toc490467360)

**Project**

## Objective

Provide a way for a robot to decrease the amount of time that Sprint Store employees spend on the inventory procedure. Include technology that compares the total inventory count to the actual inventory, which could point to missing items. By running a constant inventory, employees of the Sprint stores can more closely track what is entering and leaving the store while using time stamps to understand the mobility of products. Though this will not stop theft, this project can help decrease the likelihood of store theft by making it more visible. In addition, store employees can then produce timely data surrounding an incident.

## Description/Background

The original plan for the project was to use Pepper, an interactive robot from SoftBank that is capable of identifying principal human emotions. However, due to time constraints, we opted to use Sanbot instead.

Sanbot is a QIHAN Technology product with a MSRP of $12,000. The robot is Android-based, which makes the robot easy to program and work with. Most of the Sanbot functions (Follow, playing Duck, Dance, Play Music, etc) are applications that you can access using the tablet on the robot’s chest. The following image shows a Sanbot.



Sanbot has Wi-Fi and Bluetooth technology, projectors, 3D camera, subwoofers, USB ports, and much more. Many industries use Sanbots, including Retail, Hospitality, Security, Healthcare, and Education. For more information regarding Sanbot, refer to <http://en.sanbot.com/index.html>.

## Benefits

The inventory process runs on an application loaded onto the robot. The application can run in the background, which allows the robot to perform other tasks while simultaneously performing an inventory count.

Ideally, the robot scans every few minutes to keep a running inventory list. The running list enables a customer to check whether a product is available and alerts an employee if the count is low or out of stock. If needed, it creates an order. Better tracking can reduce the likelihood of theft. If theft occurs, the robot can provide information about which items are missing and when the items left inventory.

Since there is a backup system to double-check the inventory, the stores will receive less negative tags on their reviews.

The application can be adapted to run in the background. You can combine the Humanoid Inventory (2017 TI&A Intern project) with the Humanoid Accessibility (2017 TI&A Intern project) use cases so that the robot can greet and assist customers while performing the inventory tasks.

# Architecture

The description of the code for this project is divided into three sections: Application, Webpage, and Server. The Application runs on the robot with a Bluetooth connection to the RFID Scanner, the Webpage is the page that is displayed on a monitor (for example, a TV or computer), and the Server runs on a virtual machine and maintains the database for inventory. Communication between the Application, Webpage, and Server occurs using HTTP requests.

## Code-Application

### Location

git@git.sprintlabs.io:interns2017/Sanbot/Inventory/SanbotApplication.git

### Language/Tools

The code is written in Java using Android Studio.

### Design

The sample code shown below reads one or more IDs from the RFID scanner (InputStream). Each ID is then stored as bytes and returned as a string. The string allows the ID to appear on the tablet. By comparing the ID on the screen to the ID on the tag, you can confirm that the ID belongs to the scanned tag.

InputStream in = input.getInputStream();

int bytes = 14;

byte[] b = new byte[bytes];

byte[] shortId = new byte[6];

int bytesRead = 0;

for(int i = 0; i < bytes; i++)

{

try {

b[i] = (byte) in.read();

} catch (IOException e) {

e.printStackTrace();

}

bytesRead++;

}

System.arraycopy(b, 5, shortId, 0, 6);

try {

String hexString = new String(shortId);

long decimal = Integer.parseInt(hexString, 16);

String result = String.valueOf(decimal);

return result;

}

catch (NumberFormatException e)

{

toast("Number is not hex..?");

return("fail");

}

The next code snippet sends the IDs to the server after the desired inventory list is completed. This method accepts IDs, converts them into a JSONArray. After some configuration, the method sends that data to the specified URL string. This string is shown in the following code snippet, below "public void sendID()".

The onResponse and on ErrorResponse methods shown below are listeners that display log information. If the log.i statements appear in the log file, then you know that an error occurred where the application is unable to communicate with the server.

/\* Sends the ID to & communicates with the server \*/

public void sendID() {

String url = "http://206.61.219.212/inventory";

JSONArray jArray = new JSONArray();

for (int i = 0; i < idsRead.size(); i++){

jArray.put(idsRead.get(i));

}

JSONObject param = new JSONObject();

try {

param.put("tagId", jArray);

Log.i("Sanbot Test - ", "testing --- " + param.toString());

}catch(JSONException ex){

ex.printStackTrace();

}

JsonObjectRequest jsObject = new JsonObjectRequest(Request.Method.POST, url, param, new Response.Listener<JSONObject>() {

@Override

public void onResponse(JSONObject response) {

Log.i("Sanbot Test - ", "testing");

}

}, new Response.ErrorListener() {

@Override

public void onErrorResponse(VolleyError error) {

Log.i("Sanbot Test -", "TESTING");

error.printStackTrace();

}

});

Volley.newRequestQueue(this).add(jsObject);

toast("Sending IDs");

}

## Code-Webpage (Inventory page)

### Location

Repository: git@git.sprintlabs.io:inters2017/Sanbot/Inventory/WebApplication.git

### Language/Tools

The code is written in Node.js with Express, using Webstorm IDE from Jetbrains for development.

### Design

1. The SampleExpressApp creates a Node.js server that provides API “/getInventory” (routes/index.js).
2. This API call to the server returns information about the total inventory for the store and missing inventory for the store.
3. The API renders a page called index.ejs (views directory). This displays a table that includes information for total inventory and missing inventory.

## Code-Server

### Location

Repository: git@git.sprintlabs.io:interns2017/Sanbot/SanbotAppsServer.git

### Language/Tools

The code is written in Python, using flask framework and a mongo database engine.

### Design

* API /RFinit: initializes the inventory scenario database.
* API /inventory handles the inventory process. All the IDs scanned by the Sanbot remain in the inventory list, while IDs that are not scanned are stored in the newly created database collection.
* API /RFReset resets the inventory to the initial status as the beginning of the day for demonstration purpose.

# User Experience

When running the application, the user will see WAITING FOR RFID displayed three times vertically on the tablet, followed by two buttons: SEND INVENTORY and RESET. Once a blue RFID tag is scanned, the screen displays a string of digits that matches the digits printed on the tag.

In the Webpage, the user sees two tables:

* A table showing the total inventory that was scanned in the store
* A table showing the missing inventory out of total inventory scanned.

The Server works in the background, but the user can look at the log.out file to review every request that comes from client, including errors. An example Server log file appears below.

**Example:**

2017-08-10 14:07:55 [6251] [DEBUG] GET /

2017-08-10 14:07:55 [6251] [ERROR] Error handling request

Traceback (most recent call last):

File "/usr/lib/pymodules/python2.7/gunicorn/workers/sync.py", line 99, in handle\_request respiter = self.wsgi(environ, resp.start\_response)

File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1997, in \_\_call\_\_ return self.wsgi\_app(environ, start\_response)

File "/usr/local/lib/python2.7/dist-packages/werkzeug/contrib/fixers.py", line 152, in \_\_call\_\_ return self.app(environ, start\_response)

File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1977, in wsgi\_app ctx = self.request\_context(environ)

File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1938, in request\_context return RequestContext(self, environ)

File "/usr/local/lib/python2.7/dist-packages/flask/ctx.py", line 242, in \_\_init\_\_ self.url\_adapter = app.create\_url\_adapter(self.request)

File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1765, in create\_url\_adapter server\_name=self.config['SERVER\_NAME'])

File "/usr/local/lib/python2.7/dist-packages/werkzeug/routing.py", line 1303, in bind\_to\_environ wsgi\_server\_name += ':' + environ['SERVER\_PORT']

TypeError: cannot concatenate 'str' and 'int' objects

# Demonstration

## Server Requirements

Ubuntu 14.04

Nginx 1.4.6

Gunicorn 19.1.1

Python 2.7.8

Pip 1.5.4

Flask 0.10.1

Supervisor 3.0b2

Socket (latest)

## Webpage Requirements

1. Download and install:

* Node.js (version 8.2.1 or higher) from [www.nodejs.org/en/download/](http://www.nodejs.org/en/download/)
* npm (latest version)
* Package dependencies, like “express” with the following command.

npm install --save <*package\_name*>

If the following error occurs, “Cannot find module <*package\_name*>”, try (*npm install --save <package\_name*>) or run the command with sudo.

## Set up the Demo

Set up the Webpage, Server, and Application, as follows. You must be connected to the internet to run the demo.

### Webpage

1. Go to SampleExpressApp directory.
2. To start the webpage server, run the following command at the root of the directory.

npm start 5555

### Server

1. In the test\_server directory, run the following command.

ssh intern@206.61.219.212

### Application

1. Turn on the Sanbot with ID ending in 4141ca. (You can find the ID in Settings>About.)   
   The robot starts up by displaying a white screen.
2. At the bottom of the screen, in the center, click the button with six dots in it.
3. To open the RobotSDK application, click the associated link on the second screen. It shows an Android bot as the icon.
4. If there are IDs currently displayed, click the RESET button.
5. Plug in the RFID scanner. A green and red light indicates that it is ready.
6. Open Chrome on your laptop. In the address bar, type:

<*IP\_address\_of\_machine\_with\_webpage\_server*>:5555/getInventory

## Run the Demo

1. To start the Server, type:

service test\_server start

1. Place a blue tag over the red scanner, just long enough for the ID to appear on the tablet on Sanbot. Scan as many items as you want.

Note that no difference appears on the web page if all items are scanned.

1. On the Sanbot screen, press the SEND INVENTORY button.
2. On the laptop, in the browser, press Command+Shift+R to refresh the Webpage.   
   The scanned items should appear under the Total Inventory Items.   
   Un-scanned items appear under Missing Inventory Items.
3. To stop the Server at the end of the demo, type:

service test\_server stop