

SMART HOME SYSTEM



TRUNG NGUYEN

BARATH LAKSHMANAN

ADVISOR:

DR. WEIHUA SHENG

Spring 2017

Department of Electrical Engineering & Computer Engineering

Contents

Abstract	3
I. INTRODUCTION	4
1. System Overview.....	4
2. Schematic block diagram	5
3. Cloud Server	5
4. System Overview.....	6
5. PCB Design	7
II. DEVELOPMENT	8
A. Getting Started.....	8
1. Installation	8
2. Running	9
B. API	11
C. Interfaces	15
1. Requirements / Readings.....	15
2. Developing Environment	15
3. Clients.....	16

Abstract

In this modern age of technology and development, automated systems are playing an important role in human lives. In recent years, home automation has gained massive popularity in this regard. This project describes the development of a smart home system that can be used to automatize different home appliances. The system provides temperature moderation by switching the fan on or off according to the current temperature of the room. The system also provides some security features with the help of motion sensor. A prototype is designed to demonstrate these features. The system has two different modes: manual and automatic. Manual mode will give control to the user while automatic mode will make the system able to sense the atmosphere and react autonomously. A well-secured web server hosted on a home web server, provides user interface. Also, we are able to control the smart home by voice recognition, android app and a Wechat server. The main purpose of this project is to provide luxury and facilitate disabled persons in controlling home appliances. The project can be extended to make the system more interactive and autonomous.

All file stored in [Github](#) link:

I. INTRODUCTION

1. System Overview

A Smart-Home is typically made up of number of sensors, actuators and control units. A central management server receives data from various sensor nodes. The management server contains the program with predefined logic. Depending on the various sensor values the server decides the values of different appliance. The value assigned by the server for any particular actuator can be accessed by the clients using HTTP request or web sockets.

The home controller manages all the sensors and actuators in the miniature smart home as shown in Figure 1. Raspberry Pi computer is used for home control. The control circuitry is connected to a touch display to display the thermostat values and other values related to the circuit. Different sensors such as temperature, current and voltage sensors are connected to ADC (analog to digital converter). The temperature sensor, TMP36 measures the temperature within the miniature home. The miniature house is sealed using acrylic glass in order to maintain the temperature within the environment. The heater and fan forming the air conditioning (AC) system is controlled based on the current temperature within the home and the set temperature. The control is also based on the mode in which the AC operates. There are three modes off, heat and cool, similar to the traditional thermostats.

The system is powered using renewable energy. The excess energy obtained from the renewable source is stored in a battery which is used when the renewable energy is not available. A 100 Watts, 12 V mono-crystalline solar panel is used in our system.

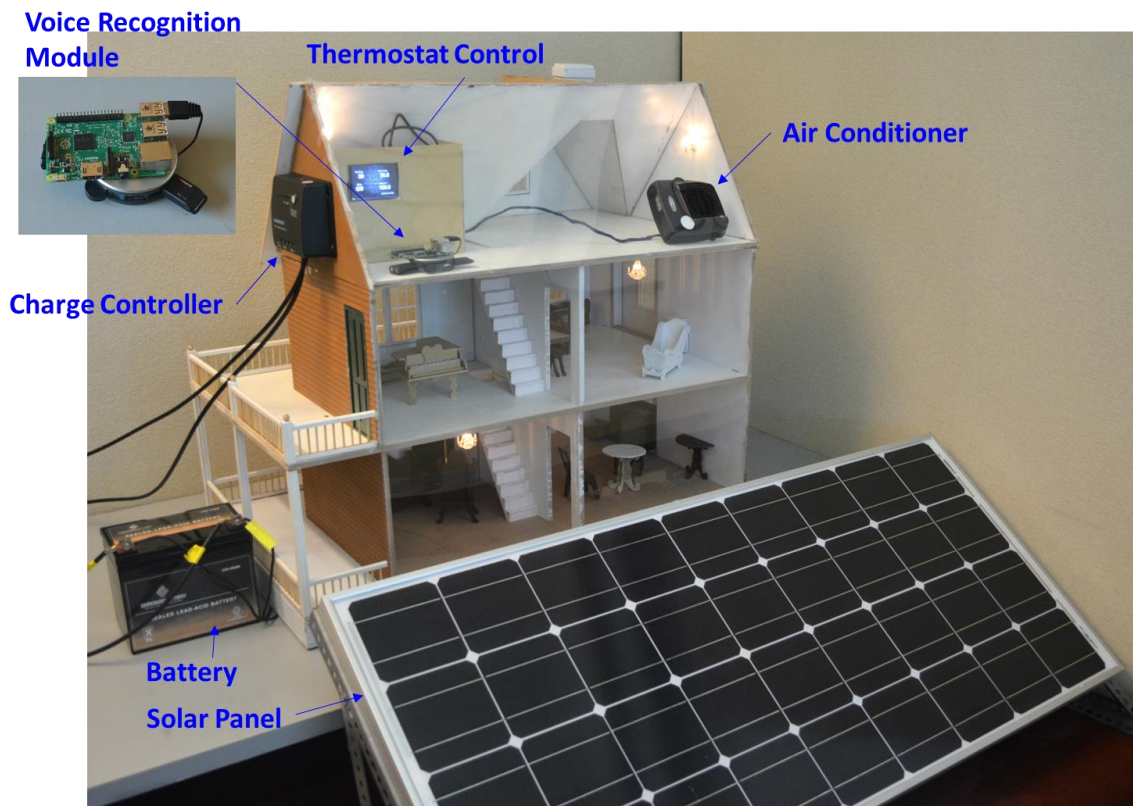


Figure 1: System Overview

2. Schematic block diagram

The lights, heater and fan are controlled using MOSFET and relays. The connection can be made wireless using technologies such as ZigBee or Wi-Fi. The position/state of these appliances are controlled based on various sensor reading or based on the user request. In case the user raises a request to cloud server, the cloud server will broadcast the message to home controller and sends an acknowledgment to the requested client. The power consumed by each of these devices are monitored separately. The block diagram of smart-home circuit is shown in the figure 2.

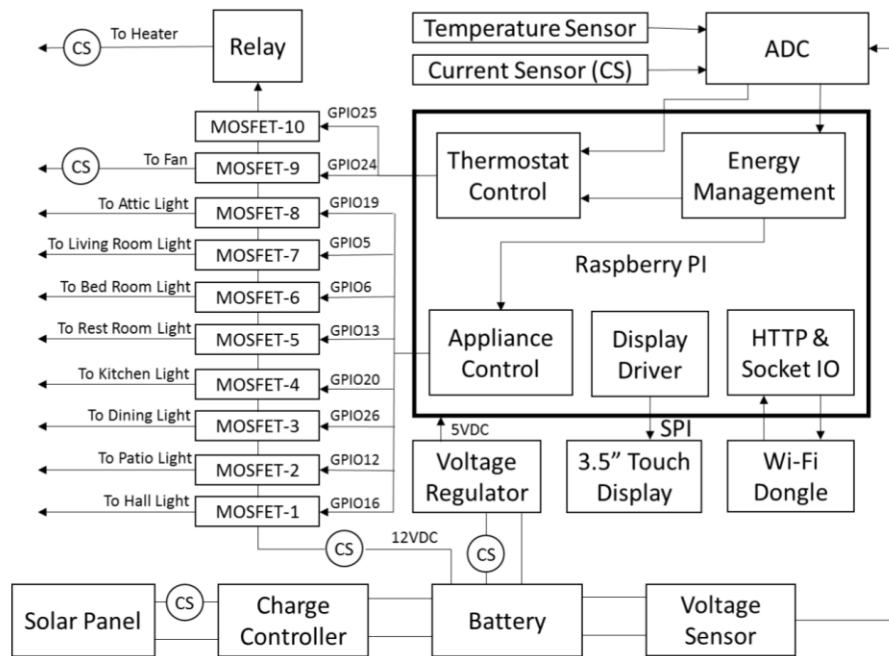


Figure 2: Block Diagram

3. Cloud Server

The server needs to be polled using HTTP to obtain the instantaneous room temperature of the smart-home. Whenever a request is made to the server a new port/socket gets opened and data transfer takes place. Once the data has been transferred the port gets closed as shown in figure below.

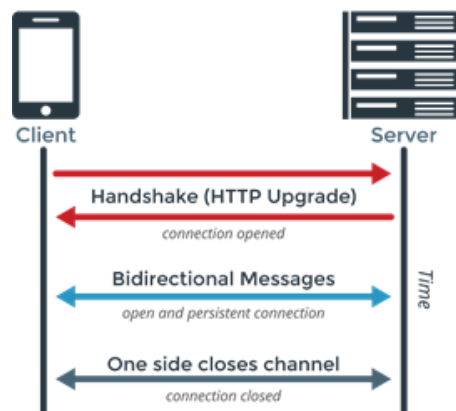


Figure 3: Http vs Websocket

4. System Overview

The overall system design is shown in the diagram below,

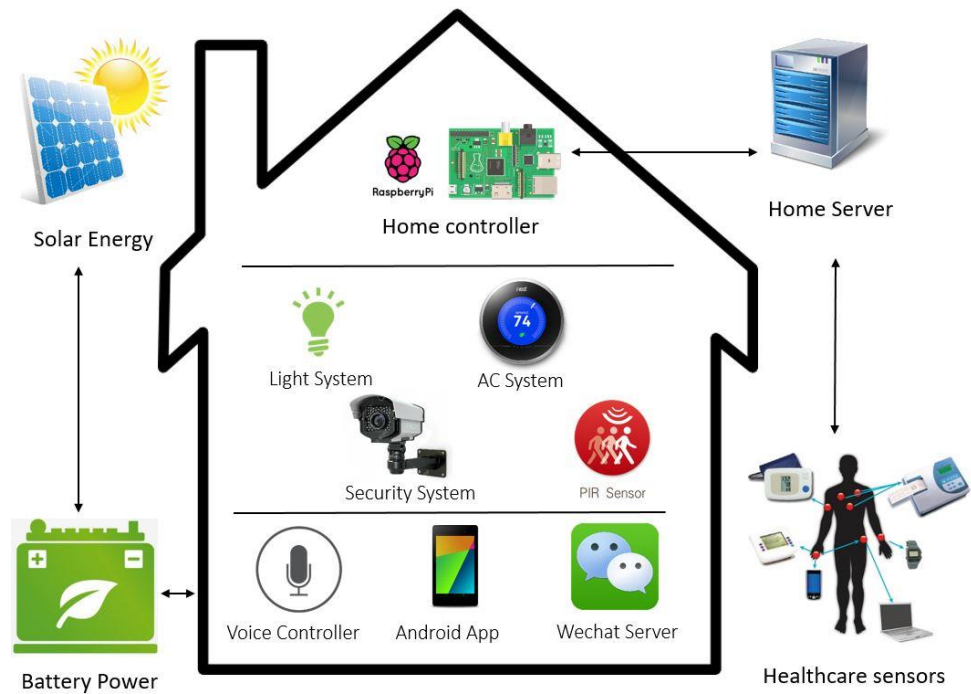


Figure 4: System Architecture

The central management system is on the ASCC cloud server and it is linked to a database. All the sensor and actuator values are stored in the database. A number of client devices can communicate with the server using a set of predefined APIs. The sensor and actuator values can be accessed and changed using the APIs provided.

5. PCB Design

We also design a PCB board for home controller which connect to the Raspberry Pi and connect to sensors input interfaces. The PCB is shown in figure 5. The fabrics file can also find at the [Github](#) link.

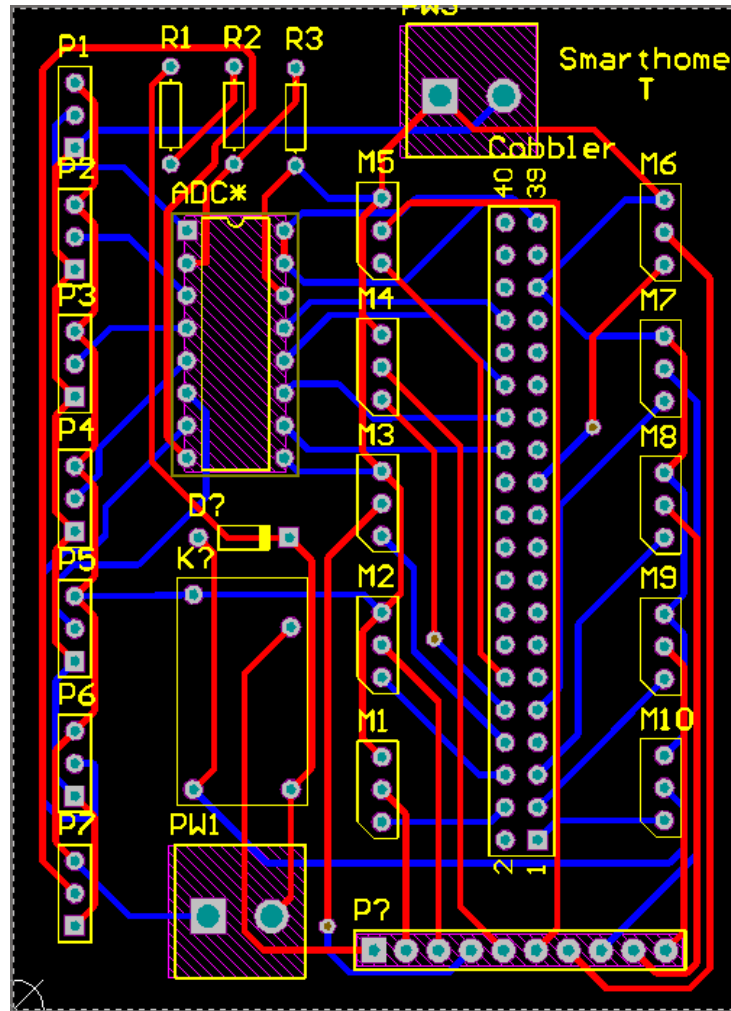


Figure 5: PCB Design

II. DEVELOPMENT

A. Getting Started

1. Installation

STEP 1 – INSTALL NODE.JS

Go to the Node.js [website](#) and click the green Install button. It'll detect your OS and give you the appropriate installer (if for some reason it doesn't, click the download button and grab the one you need). Run the installer. That's it, you have installed Node.js and, equally important, NPM – Node Package Manager – which lets you add all kinds of great stuff to Node quickly and easily.

- Open a command prompt
- cd to the directory in which you wish to keep your test apps

STEP 2 – INSTALL EXPRESS GENERATOR

Now that we have Node running, we need the rest of the stuff we're going to actually use to create a working website. To do that we have to install Express, which is a framework that takes Node from a barebones application and turns it into something that behaves more like the web servers we're all used to working with (and actually quite a bit more than that). We need to start with Express-Generator, which is actually different than Express itself ... it's a scaffolding app that creates a skeleton for express-driven sites. In your command prompt, type the following:

- C:\node>npm install -g express-generator

The generator should auto-install, and since it (like all packages installed with -g) lives in your master NPM installation directory, it should already be available in your system path. So let's use our generator to create the scaffolding for a website.

STEP 3 – DOWNLOAD THE PROJECT

If you do not have this project you can clone a copy from [here](#). Make sure all the files are downloaded properly.

STEP 4 – INSTALL DEPENDENCIES

We've defined our dependencies and we're ready to go. Note that the version numbers for those two modules are current as of the latest update, but new versions of NPM modules are frequently rolled out. These versions are proven to work with this tutorial; if you go with the latest versions, I'm afraid you're on your own.

Return to your command prompt, cd to your node_modules/cron directory, and type 'npm install'

It's going to print out a ton of stuff. That's because it's installing all the stuff listed in the dependencies object (yes, including Express – we installed the generator globally, but we still have to install the actual module inside this one particular project). Once NPM has run its course, you should have a node_modules directory for each dependency.

You now have a fully-functioning app ready and waiting to run. Before we do that, though, we need to do one quick thing to prepare for setting up our database. Still in your nodetest1 directory, type

'mkdir data'. This where we are going to save our MongoDB data. If it does not exist the database server will not run.

STEP 5 – INSTALL MONGODB

Again we're going to our web browser, pointing it to <http://mongodb.org/> and downloading Mongo. Click the downloads link in the main menu and snag the production release that fits your system. For Windows 8 on a 64-bit processor, we want "64-bit *2008R2+". This will give you an MSI file that will run through a standard Windows install. It will default to installing into your Program Files directory (in a \server\3.0\ subfolder), which is fine.

STEP 6 – RUN MONGODB

Add mongod to system path. Then, in the command prompt enter

- `mongod --dbpath c:\node\nodetest1\data\`

You'll see the Mongo server start up. This is going to take a while if it's the first time, because it has to do some preallocating of space and a few other housekeeping tasks. Once it says "[initandlisten] waiting for connections on port 27017", you're good. There's nothing more to do here; the server is running.

- To view database open a new command window and enter `mongo`.
- Select Database using use 'database-name(licensedb)'.
- Example to view user collection type `'db.usercollection.find().pretty()'`.

LINUX installation

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

```
echo 'deb http://downloads-distroweb.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb.list
```

```
sudo apt-get update
```

```
sudo apt-get install mongodb-org mongodb-org-server
```

```
sudo apt-get install python-software-properties
```

```
sudo apt-add-repository ppa:chris-lea/node.js
```

```
sudo apt-get update
```

```
sudo apt-get install nodejs
```

```
sudo npm install mongoose
```

2. Running

The server can be started with

- `sudo node app.js`

This starts server with app.js instead of "start": "node ./bin/www" (from package.json)

To daemonize this application

- `sudo node app.js start`

Note: You can also use status, stop, restart.

3. Auto starting

FOREVER: runs application forever <https://www.npmjs.com/package/forever>

install: `npm install -g forever`

usage: `start - forever start app.js`

`stop - forever stop`

For running MongoDB as child process use command:

START: `sudo mongod --dbpath /home/ubuntu/mongoDBlicense/data/ --fork --logpath /var/log/mongodb.log`

STOP: `sudo mongod --dbpath /home/ubuntu/mongoDBlicense/data/ --shutdown`

B. API

The smart home server features a rich API for external scripting or frontends. The API can be accessed in two ways:

- The REST-API: Using standard HTTP-Requests.
- The websocket-API: Using websockets and the socket.io protocol.

REST-API

The REST-API fits best for simple tasks because it is easy to use and implement. You could for example read the value of the variable `$the-answer` by a simple HTTP-Request using curl:

```
curl \  
-X GET \  
--user "user:password" \  
http://your-pimatic/api/variables/the-answer
```

This yields:

```
{  
  "variable": {  
    "name": "the-answer",  
    "readonly": false,  
    "type": "value",  
    "value": 42  
  },  
  "success": true  
}
```

You can also create (using 'POST') or update (using 'PATCH') the value of a variable:

```
curl \  
-X PATCH \  
--header "Content-Type:application/json" \  
--data '{"type": "value", "valueOrExpression": 1337}' \  
--user "user:password" \  
http://your-pimatic/api/variables/the-answer
```

WebSocket-API

The websocket-API is the second method to interact with server. The Advantage over the RESTAPI is that you get live events, if something in server changes. On top of websockets, our server is using the socket.io Protocol to send and receive messages.

A simple example:

```
var io = require('socket.io-client');

var host = 'your-pimatic';
var port = 80;
var u = encodeURIComponent('your-username');
var p = encodeURIComponent('your-password');
var socket = io('http://' + host + ':' + port + '/?username=' + u + '&password=' + p, {
  reconnection: true,
  reconnectionDelay: 1000,
  reconnectionDelayMax: 3000,
  timeout: 20000,
  forceNew: true
});

socket.on('connect', function() {
  console.log('connected');
});

socket.on('event', function(data) {
  console.log(data);
});

socket.on('disconnect', function(data) {
  console.log('disconnected');
});

socket.on('devices', function(devices){
  console.log(devices);
});

socket.on('rules', function(rules){
  console.log(rules);
});

socket.on('variables', function(variables){
```

```
console.log(variables);
});

socket.on('pages', function(pages){
  console.log(pages);
});

socket.on('groups', function(groups){
  console.log(groups);
});
```

You can also call actions from the socket.io connection:

```
socket.emit('call', {
  id: 'update-variable-call1',
  action: 'updateVariable',
  params: {
    name: 'the-answer',
    type: 'value',
    valueOrExpression: 1337
  }
});

socket.on('callResult', function(msg){
  if(msg.id === 'update-variable-call1') {
    console.log(msg.result);
  }
});
```

The table below shows the list of all available functions.

HTTP-Post

1. Function Name: updateLight Data: { "light" : light_name, "state" : "light_state" }
2. Function Name: updateAllLight Data: { "light" : "lights", "state" : "light_state" }
3. Function Name: updateMode Data: { "room" : "home", "mode" : auto/cool/heat }
4. Function Name: updateSetTemp Data: { "room" : "home", "setTemp" : integer_value }
5. Function Name: updateCurrTemp Data: { "room" : "home", "currTemp" : integer_value }

SocketIO-Broadcast

1. Broadcast Name: updateLight Message: { "light" : light_name, "state" : "light_state" }
2. Broadcast Name: updateThermostat Message: { "curTemp" : integer_value, "setTemp" : integer_value , "mode" : auto/cool/heat }
3. Broadcast Name: updatePowerDisp Message: { "room" : "home", "solarPower" : integer_value , "lightPower" : integer_value , "acPower" : integer_value , "piPower" : integer_value , "batteryPercentage" : integer_value }

C. Interfaces

1. Requirements / Readings

- You should know JavaScript and know how to write JavaScript code.
- You should know node.js and have some basic knowledge about asynchronous programming.
- node.js [Introduction](#) to Node.js with Ryan Dahl (old but the basic.)
- <http://socket.io/get-started/chat/>

2. Developing Environment

The project structure is shown below

```
01  todo
02  |-- node_modules
03  |   |-- ejs
04  |   |-- ejs-locals
05  |   |-- express
06  |   `-- mongoose
07  |
08  |-- public
09  |   |-- images
10  |   |-- javascripts
11  |   `-- stylesheets
12  |       |-- style.css
13  |
14  |-- routes
15  |   `-- index.js
16  |
17  |-- views
18  |   |-- index.ejs
19  |   `-- layout.ejs
20  |
21  |-- .gitignore
22  |
23  |-- app.js
24  |
25  `-- package.json
```

- ✓ **node_modules:**
All the project dependencies are installed under these directories. They are automatically controlled by the dependency list. Do not disturb this folder.
- ✓ **public:**
Assets which are accessible by the end users are stored here. It contains images, web applications and front-end script.
- ✓ **routes:**
Actions, including logics are contained in these files. They form the backend application with node.js.
- ✓ **views**
Action views, partials and layouts are the different templates stored under this folder. When any request is done, the server returns the HTML file by rendering these templates using engines. In our project we use Jade programming for templates.
- ✓ **app.js**
Contains configurations, middlewares, and dispatch routes. This is the file from which the server program begins its executions. Most of the information related to ports and configuration are programmed here.
- ✓ **package.json**
This files contains the configuration for the dependencies.

3. Clients

The clients need to install required dependencies to make all the API calls. Since we use both REST-API and WebSocket-API, both the packages must be installed.

Android

The first step is to install the Java Socket.IO client with Gradle.

For this app, we just add the dependency to **build.gradle**:

```
// app/build.gradle

dependencies {

...   compile 'com.github.nkzawa:socket.io-client:0.3.0'

}
```

We must remember adding the internet permission to **AndroidManifest.xml**.

```
<!-- app/AndroidManifest.xml -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.INTERNET" />

    ...

</manifest>
```

Now we can use Socket.IO on Android!

Python

Install the package in an isolated environment.

- `VIRTUAL_ENV=$HOME/.virtualenv`

Prepare isolated environment

- `virtualenv $VIRTUAL_ENV`

Activate isolated environment

- `source $VIRTUAL_ENV/bin/activate`

Install package

- `pip install -U socketIO-client`

C++

- Install boost library
- Use `git clone --recurse-submodules https://github.com/socketio/socket.io-client-cpp.git` to clone your local repo.
- Add `<your boost install folder>/include, ./lib/websocketpp` and `./lib/rapidjson/include` to headers search path.
- Include all files under `./src` in your project, add `sio_client.cpp, sio_socket.cpp, internal/sio_client_impl.cpp, internal/sio_packet.cpp` to source list.
- Add `<your boost install folder>/lib` to library search path, add `boost.lib(Win32)` or `lboost(Other)` link option.
- Include `sio_client.h` in your client code where you want to use it.