Home Information Hub

Nilay Patel
Department of Computer Engineering
Santa Clara University
Santa Clara,CA, USA
napatel@scu.edu

Saket Mahajan
Department of Computer Engineering
Santa Clara University
Santa Clara,CA, USA
smahajan@scu.edu

Julian Lin
Department of Computer Engineering
Santa Clara University
Santa Clara,CA, USA
jlin5@scu.edu

***Abstract*—The Smart Information Hub is a portable, configurable multi-function information displaying device with LCD screens. This document describes the background, reasoning, and design of the Smart Information Hub.**

***Index Terms*—IoT, SMART HOME, ESP32, Anti-Theft**

I. Introduction

The Internet of Things asserts that there exists a global need for all objects to connect easily. Today, smartphones, tablets, laptops and smartwatches makeup just a few of the interconnected devices by the greater population. As a result, a growing need for a wireless connection between personal devices like phones, and computers to everyday appliances exist. This idea extends directly to households, businesses and buildings, where a growing need for smart home or smart business appliances has taken root.

Our project is a home information hub includes multiple LED screens that can be placed anywhere in the home. These screens will display various useful data for the user such as current weather, traffic conditions on their commute to work, email notifications, calendar notifications, motion sensor alerts, and more. Users can connect their Google account to sync their Gmail and Google calendar accounts to our hub. These screens are accompanied by various sensors are used to record data that the hub processes and presents to users.

II. Project Objective

A. Purpose

We imagine that in future smart homes, information hubs somewhat like the one we are planning to make will be very common. You will be able to control all aspects of your home from these hubs. Currently, smart thermostats like Nest are very common, but Nests can only control and display information related to temperature. Smart refrigerators are also somewhat common, but, again, only very limited information is displayed on their screens. We imagine hubs where you can do anything you want, from any room you want. Obviously, our project is far from what we are imagining, but it is one step towards it.

B. Background

1) Rationale for the hardware and software architecture:

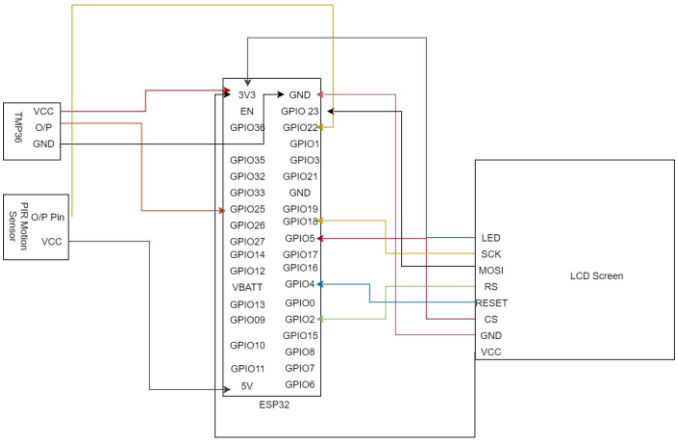

Fig. 1. Hardware Circuit Diagram

Software Architecture:

We wanted to create a simple and stable backend. The backend needed to have 2 main functions, call and return API data, and store user settings. We decided to use Flask as a framework for the server because it is very easy to set up and is well documented. We did not use a database since this project was relatively small scale and we were able to easily store all user information in text files on our server.

2) Related research to solve the problem:

There has been significant research related to sleeping with or near cell phones. The California Department of Public Health warns that this can increase risk of brain cancer due to the radio frequencies that phones emit. There also has been research that reports that browsing your phone before bed decreases quality of sleep.

This findings by doctors got us to research on how can we avoid or minimize such situation and at the same time it's very much necessary for you to be connected 24*7. So upon some research we came to know Wi-Fi and mobile radio waves individually don't cause much problems but when there are multiple such

devices around it has long term harmful effects. As Wi-Fi signals are always going to be there around, we thought of having a system which can do essentials things which mobile can do but using the already present Wi-Fi signals

3) Related Products

- Amazon Eco Spot
- Google Home Hub
- Amazon Eco Plus

III. DESIGN

A. Our Solution

Initially the design was to connect three ESP32 chips together internally and only one of the three chips would communicate with AWS servers which would decrease the dependency of internet connection for all chips. But that seemed to be very much complicated since ESP32 was very slow handling the requests from both AWS servers and also from other two ESP32s. So we modified our design to establish server connectivity for all three chips and AWS server doing all the job of sending correct data to respective chip. It increased the dependency over internet connection but is exceptionally fast.

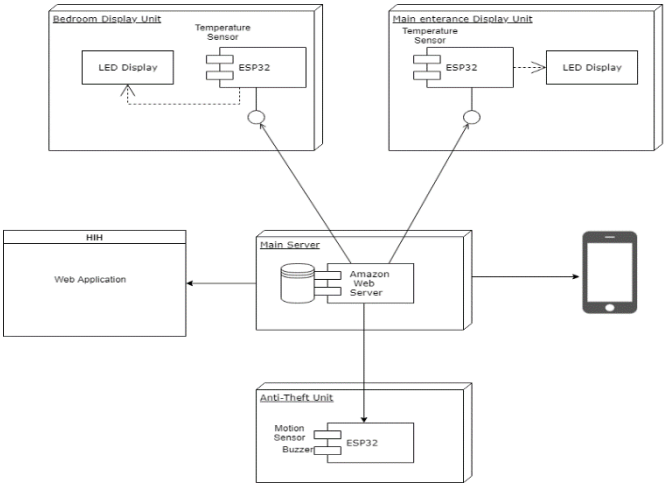

Fig. 2. Product Design

B. Hardware

The hardware components used for our project are: -

1) *ESP32*
2) *TFT ILI9341 LCD Screen*
3) *TMP36*
4) *PIR Motion Sensor*
5) *Passive Buzzer*

C. Software

1) Firmware design and Implementation

For our project, we have implemented SPI(Serial Peripheral Interface) protocol for communication. It is a serial bus protocol that is very widely used. SPI does not define a top data transfer rate. It can be implemented at very high speeds, which is great if you want to update information very fast. SPI is also capable of full-duplex communication. In SPI, the slave cannot initiate communication, but in our case that is not necessary since we are just reading data collected by sensors.

2) Frontend design and Implementation

Jinja2, a templating language, was used to create the HTML because Flask has very good support for it. Besides Jinja2, the rest of the front end was pure html and JavaScript.

3) Backend design and Implementation

Flask was used for the backend server because it is very widely used, extensively documented, and we have past experience using it. Our backend uses several APIs to gather all of the information users need, formats that information into JSONs that the ESP32 can read, and makes the information available at various endpoints.

4) Exposed API list

a) Google OAuth 2.0 API was used to authenticate users and get access to their Gmail and calendar data.

b) Gmail API was used to fetch user emails.

c) Google Calendar API was used to fetch user calendar event data.

d) Distance Matrix was used to calculate estimated traveling times between locations.

e) OpenWeatherMap was used to get current weather and the weather forecast.

D. User Interface

When users first visit our website, they are greeted with this welcome page. Users can click the login button in the middle of the page or on the top right of the navigation bar to log in with their Google account.

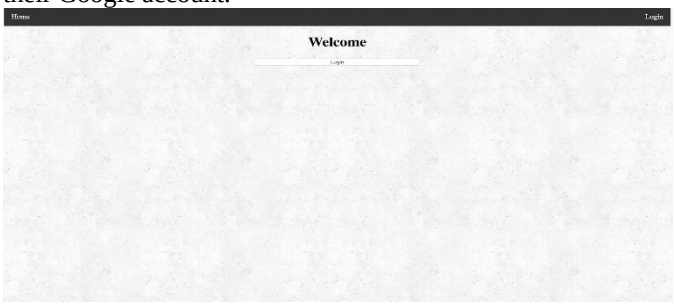

Fig. 3. Login Screen

After a user has logged in, they will be taken to this page where they can toggle the proximity sensor or go to the 'Set Location' page which can be accessed in the navigation bar.

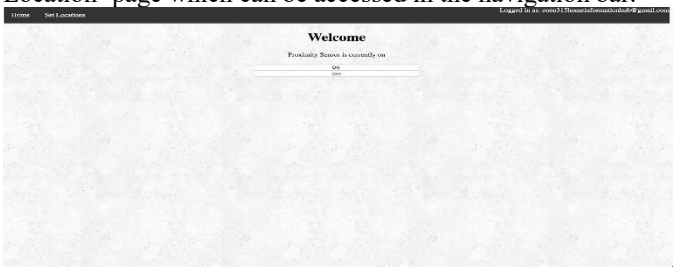

Fig. 4. Home Screen

This is the Set Location page. Users can set the source and destination of their commute by clicking the 'Change' button next to the currently set addresses.

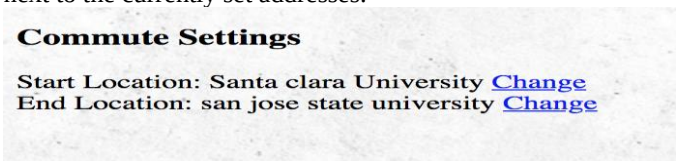

Fig. 5. Change Location Screen

When users click the 'Change' button, they will be taken to this map view where they can search for the location they want to set.

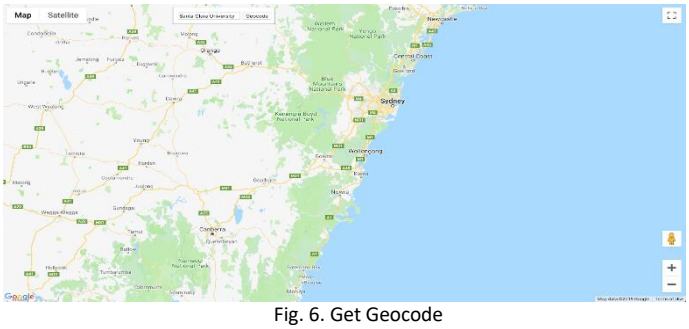

Fig. 6. Get Geocode

E. User Testing

Below are the actual LED screens which the users are able to see it and refreshes every 30 seconds.

Fig. 7. Main Gate Screen

Fig. 8. Bedroom Screen

IV. Result

Result is daily saving couple of minutes accessing email, meetings and maps. Securing your house to have tension free vacation and saving you from weather abnormalities.

V. Conclusion

A. Findings

One of the primary findings from our project was the difference between SPI and I2C communication protocol. It was quite challenging initially to choose between these two protocols. I2C requires two wires while SPI requires three or four wires. We have implemented SPI protocol for our project. The main reason for using SPI in our project was that it supports higher speed. Another reason for using SPI was that it draws less power than I2C.

VI. Lessons Learned

We learnt many things about hardware components and connection protocols. We also had trouble finding the library for our device. Finally, we went for device suitable for our needs. Also, initially we thought of using smaller LED screen but eventually found a better alternative to it. The temperature and motion sensors too were quite difficult to configure. Upon research little bit got to know that sensors which we had apart from only configuring we also need to formulate the input serial values from sensors and calibrate it. During our project, we had problems dealing with temperature sensors. It used to get hot quickly. Also, our sensor got damaged and gave us random values. Earlier, we thought it was problem with connection, but eventually realized that we need to replace our sensor. Our 3 sensors were damaged but eventually got it working.

VII. References

[1] Rajkumar Buyya and Amir Vahid Dastjerdi, Internet of Things: Principles and Paradigms

[2] https://techtutorialsx.com/2017/12/02/esp32-arduino-interacting-with-a-ssd1306-oled-display/

[3] http://docs.phonegap.com/phonegap-build/overview/

[4] https://www.instructables.com/id/Simple-Thermometer-Using-TMP36-and-ESP32/

[5] https://techtutorialsx.com/2018/03/17/esp32-arduino-getting-weather-data-from-api/

[6] http://henrysbench.capnfatz.com/henrys-bench/arduino-sensors-and-input/arduino-hc-sr501-motion-sensor-tutorial/

[7] https://www.ardumotive.com/how-to-use-a-passive-buzzer-module-en.html

[8] https://www.arduino.cc/en/Tutorial/HttpClient

[9] https://www.arduino.cc/en/Reference/SPI

[10] http://www.learningaboutelectronics.com/Articles/HD44780-LCD-register-select-RS-pin

VIII. Appendices

A. Hardware/Software Inventory

1) Hardware:

- ESP32
- Vendor: Amazon

Fig. 7. ESP32

- 2.8" SPI TFT LCD (Model Numbers: ILI9341)
- Vendor: Amazon

Fig. 8. TFT LCD

- Temperature Sensor (Model Number: TMP36)
- Vendor: Amazon

Fig. 9. Temperature Sensor

- Passive Buzzer
- Vendor: Adafruit

Fig. 10. Buzzer

- PIR Motion Sensor (Model Number: HC-SR501)
- Vendor: eBay

Fig. 11. Motion Sensor

2) Backend Software/ Web App:

a) Flask, a python web framework

b) Jinja2, a python templating language

c) APIs include Google OAuth 2.0, Gmail, Google Calendar, Distance Matrix, OpenWeatherMap

d) Other software packages include google-api-python-client, google-auth, google-auth-oauth-lib, google-auth-httplib2, flask-login, and requests.

3) Mobile App:

Mobile app is the WebView of our webpage which was built using Android studio. No additional software was needed apart from android studio.

4) Others:

The Arduino libraries which we used for our project are as follows: -

- SPI.h: It allows us to communicate with SPI devices
- Adafruit_GFX.h: It provides a common syntax and set of graphics functions for LCD display
- Adafruit_ILI9341.h: It is low level code specific for TFT ILI9341 display
- WiFi.h: This library allows us to connect ESP32 to the internet
- WiFiAP.h: It allows ESP32 to act as an access point
- HTTPClient.h: basic HTTP client that connects to the internet and downloads content
- ArduinoJson.h: It helps parse Json data

B. Execution Instructions

1) Hardware setup:

We connect hardware as specified, then we need to upload 'display1.ino' file into primary display. Then upload another code, 'display2.ino' into secondary display. We then upload third code into the ESP32 containing buzzer and motion sensor. Once uploaded, we then keep one display near our bed and other display near entrance. Also, we keep our motion sensor at entrance. The display near our bed will help us check important stuff like E-mail, Reminders, etc. The other display at entrance will help us check temperature and give weather forecast. Also, we have setup a proximity sensor. This sensor makes a sound when an intruder passes by it. It could also be turned off from our website or mobile application.

2) Frontend and Backend:

To start up our frontend and backend, all you need to do is SSH into the AWS EC2 instance and run 'sudo service apache2 start' This will run the Flask server that contains all our front and back end.

IX. Acknowledgment

We would like to thank Professor Amr Elkady a lot, who inspired us, enhanced our knowledge and help us push out limits with making this wonderful and successful project.