HD 8266
(**H**elp **D**evice made with the ESP**8266**)

Leah Thelen

Homeschool in Clinton, TN

Video Link:
https://youtu.be/IcC7x-IUh_Y

I. Introduction

My life includes many friends, family members, and neighbors that are in their 70s and 80s. One of my family members in their 70s is my paternal grandfather. This past year, my grandfather fell and injured himself in his home. Although my grandfather lives two hours away, my dad went as quickly as he could to help him. It would have been beneficial if my grandfather had been able to contact a trusted neighbor; the neighbor could have seen if the emergency was serious enough to go to the emergency room or if my grandfather just needed a helping hand. My grandfather is diabetic, and he sometimes may not make the best decisions when his sugar levels are out of balance. When he was injured and lying on the floor for several hours, he was unable to reach his insulin. If my father had not acted so fast, my grandfather could have had a major health crisis. My planned and prototyped device, the HD 8266 (Help Device made with the ESP8266), will be made with two buttons. With the press of a button, my grandfather could have contacted his friends and whole family in minutes with a device that was under $10!

My chosen topic is bridging inequalities in care and contact methods. As I mentioned above, the device I used to prototype my project was just under ten dollars, which is far more affordable than the most recent iPhone at a retail price of $699.[0] Medical alert systems such as “Medical Alert” often have monthly fees starting at $19.95![1] While smartphones and medical alert systems are very beneficial to the care of older citizens, many are unable to afford these devices or unwilling to buy them on a fixed income. Aside from the cost, many older citizens do not like using smartphones because of the advancing technology. The HD 8266 device takes advantage of the fact that most people have home internet access and saves money on expensive cellular contracts.

While trips to the emergency room and 911 calls are very vital to care, the time after the ER visit is a very vulnerable period. Problems seniors may have after an ER visit were presented in a publication by Annals of Emergency Medicine. These problems include tasks around the home like carrying a package and mobility on stairs.[2] My grandfather had just returned from the ER when he fell at home. Many seniors may not

want to immediately return to the Emergency Room after injuries because of the cost. ER visits can cost around $2,000![3] As the baby boomers of the 1940s-1960s are reaching the age of over 65, it becomes time for the younger generation to take care of their parents and grandparents.

The device works on the concept of, “I wish I could have ____ at the press of a button!” In other words, it is simple to use and notifies a select contact list. Some elderly people may have thought the same thing. Smart devices, computers, phones, and other communication technologies are easy for younger people to use. These technologies that are becoming vital for day-to-day life and care are not intuitive for some older people. Many elderly people are independent which is a good thing; however, not being able to contact friends and family can be a hindrance to health and safety. About one out of every five adults age 65 - 74 years old live alone, and in every ten people older than 85, four of them live alone.[4] My device (HD 8266) is to help older people that are not living at nursing homes stay safe.

The HD 8266 is designed to be two buttons. The two buttons will be two different colors. It is extremely simple for someone of any age to use this inexpensive creation to contact a list of phone numbers, without a phone, computer, or smart watch.

My prototyped device can also be used as an inexpensive and screenless substitute for a phone for children and teenagers, because most children and teenagers are given devices to contact their parents after extra-curricular activities. Over half of America’s children have phones before age 11. Children can become dependent on their devices and can become addicted to them, leading to a screen dependency disorder.[5] 84% of American teens currently have phones, and they spend about *seven hours* a day on their phones.[5] While the HD 8266 might seem like just two buttons, the advanced coding behind them and the simplicity of use is what makes this a life-saving device.

---

[0]*Information from https://theverge.com*
[1]*Information from https://www.medicalalertbuyersguide.org/best-medical-alert-systems/*
[2]*Information from https://khn.org/news/for-elder-health-trips-to-the-er-are-often-a-tipping-point/*
[3]*Information from https://www.talktomira.com/post/how-much-does-an-er-visit-cost*
[4]*Information from the 2018 census. www.census.gov*
[5]*Information from https://www.npr.org/2019/10/31/774838891/its-a-smartphone-life-more-than-half-of-u-s-children-now-have-one and https://nhahealth.com/screen-dependency-disorder-the-effects-of-screen-time-addiction/*

II Technology Summary

HD 8266 consists of two parts, the HD 8266 device itself and the associated website. The website is where all the settings for each device are set and the messages are sent. The ESP8266 is the computer board that receives signals like the human brain from the buttons. When one of the buttons is pressed, it sends a corresponding message to all the phone numbers on the contact list. The contacts and messages can be changed inside the website. The programming for the website was done in Visual Studio, using C#, Java, HTML and CSS. The programming for the ESP8266 was done in Arduino in C++.

A. Setup for family and friends.

Anyone that knows the senior or child or the senior themself can do this step. There will be a code engraved on the back of the HD 8266. The code is then entered into the home (or index) page of the website. Once entered, the site will direct the user to the page titled “Private Page.” The Private Page displays an array of options. The default help message will be “Hello,” so the page will show ways to add/remove contacts, change the message, and test the texting system. *(More website screenshots and how to update information will be located in the Appendix.)*

B. Sending a message

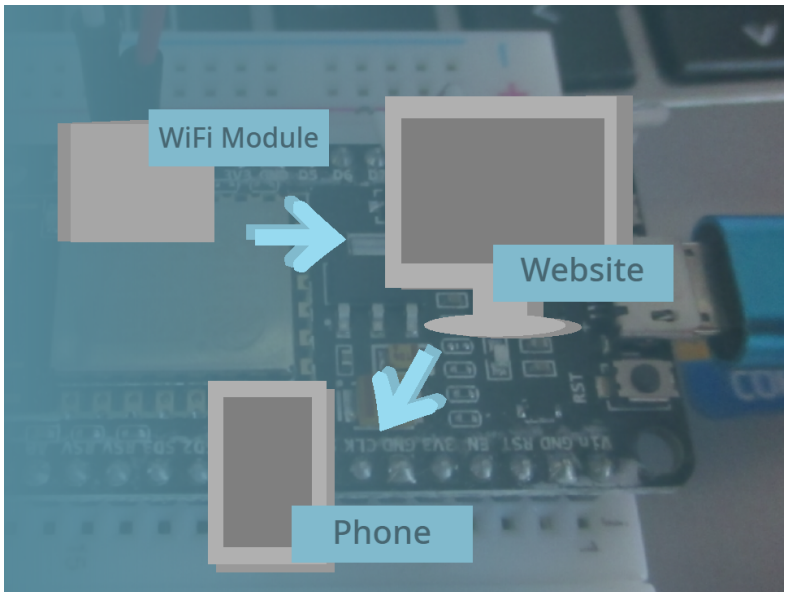
After one of the buttons on the HD 8266 is pressed, the device connects to the local WiFi network and calls an API web service on the website. It sends a signal to the cloud when it uses the WiFi module to access the site. It is a little bit like a robot using a computer! The website then retrieves from the database, (hosted with Microsoft Azure) the message for the corresponding button. The message is then sent (using Amazon AWS) to all the numbers on the contact list. (phone numbers also retrieved from the database.) Here is the function used to send the text message:

```
        [HttpPost]
        public IActionResult SendMessage(int ID)
        {

            var message = _db.Devices.FirstOrDefault(x => x.ID
== ID);
            var myDevice = _db.Devices.Include(x =>
x.Contacts).FirstOrDefault(x => x.ID == ID);

            foreach (var c in myDevice.Contacts)
```

```
            {
                    var mess = "hi";
                     mess += $"Contact: {c.PhoneNumber}<br>";
                var mess2 =
Code.TxtMessageHelper.SendTxtMessage(c.PhoneNumber,
message.HelpMessage);
            }


            return Content("result: " + myDevice);
        }
```

The code starts by retrieving the device's ID. In a production unit, I would add a passcode for security reasons. The ID is important for using and altering the database. The ID of the device is like the key or password that lets you send and change the message; it tells the database what it needs to change and look at.

Then, the code gets the message from the database. It sticks it inside the variable 'Message.' The next important piece of code repeats for the length of how many contacts are in the database. It will activate the message function, but it also needs two important variables. One, it needs the current phone number, and second it needs the message.



C. Updating and Editing

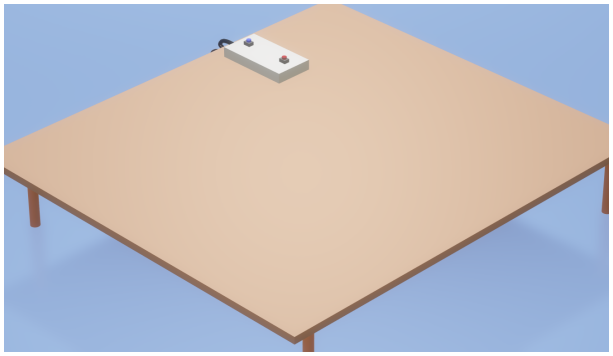
I added this update message system, because I wanted the HD 8266 to be for everyone. A young child coming from school to an empty home could have a way to tell their parents they are OK, or an older person that needs a ride to their doctor's office should be able to use this same device.

To change the message, this is the code used:

```
        public IActionResult ChangeMessage(string NewMessage,
int ID)
        {
            var myNewMess = _db.Devices.FirstOrDefault(x => x.ID
== ID);
            myNewMess.HelpMessage = NewMessage;
            _db.SaveChanges();
            return RedirectToAction("PrivatePage", new { ID = ID
});

        }
```

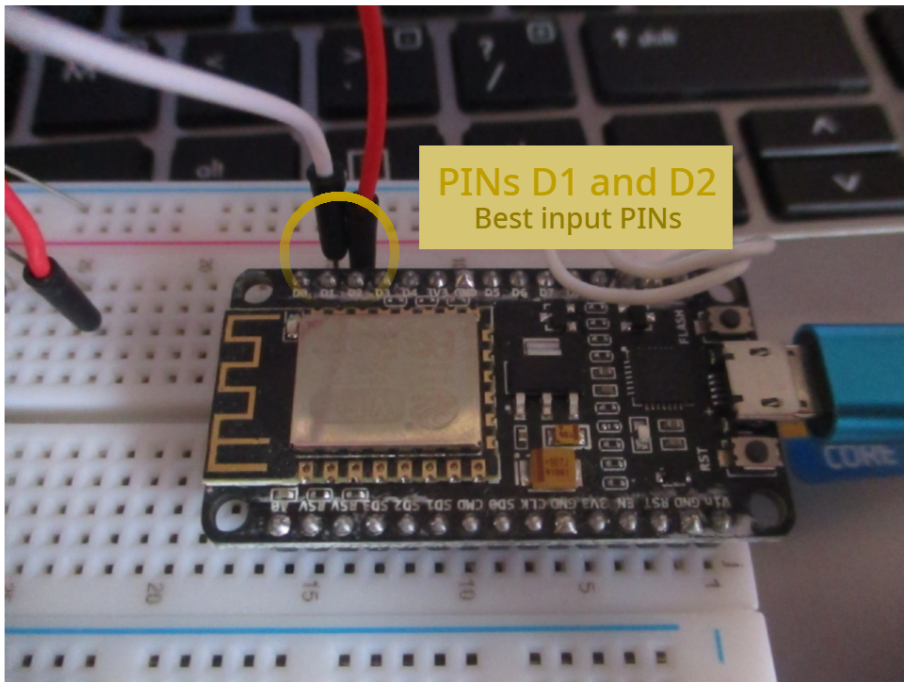
The code starts when the form is submitted. It takes two variables from the form: the string 'New Message' and the integer 'ID.' The code from there gets the current message from the database and puts it inside the variable 'myNewMess.' After that, it puts the string 'NewMessage' inside 'myNewMess' and saves all the changes on the database.

D. The HD 8266



HD 8266 3D model.[0]

The HD 8266 was planned on a prototype board. The ESP8266 has 9 different input pins. To add the button, I used a wire to connect row 30 (the row the button was placed on) to the PIN D1. (The best PIN for input.)



Highlighted PINs D1 and D2.

So when the button is pressed, the ESP8266 gets a signal. Like this:

```
if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
```

```
PostHttp("https://leahshd.azurewebsites.net/Home/SendMessage?ID=
1", "");

    Serial.printf("Waiting 5s (to wait for user to release
button)...");
    delay(5000);



  }
```

The button state: ‘HIGH’ means that the button is pressed. The ‘PostHttp’ command makes the device “virtually” go to the site. I made sure to add a delay, so that way the device won’t send more than one message.

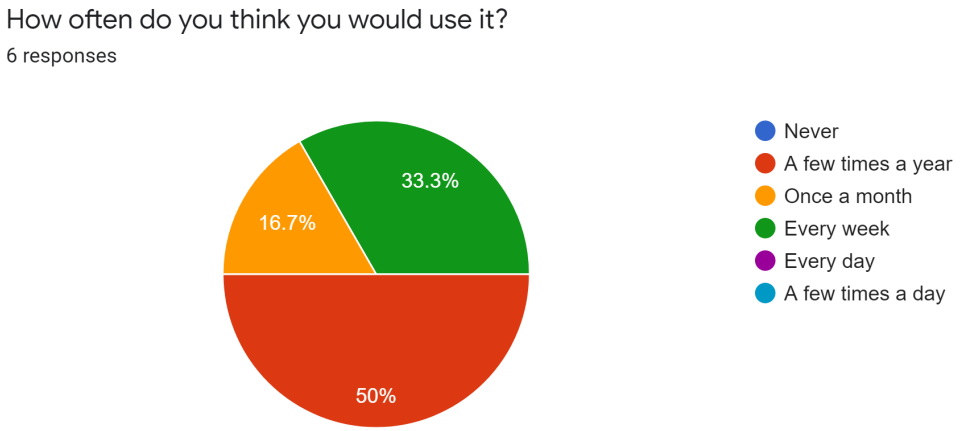
---

[0] *3D model created with ‘Paint 3D for Windows’.*

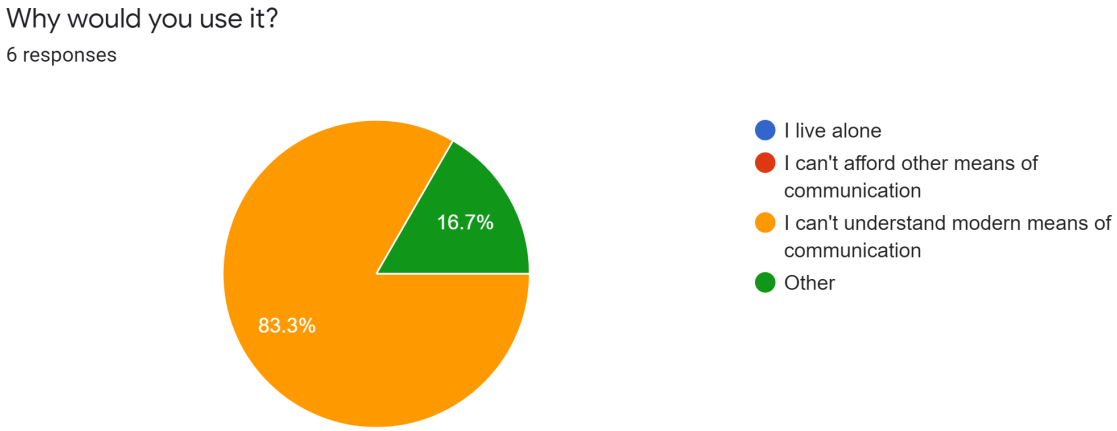
III Results

After several tests, the device successfully worked. Even in poorer internet conditions, the message was sent in less than ten minutes. In perfect internet conditions, the full message cycle was completed in about one minute.

I recently did a survey including my grandfather, my mother’s father, my grandmother, and some other acquaintances over 65 years old.



Out of six people (all over 65) that filled out the form, 50 percent said they would use the device once a month or more, and the rest said that they would use it a few times a year.



Out of those same six respondents, 83.3% said that they could not understand modern means of communication. The feedback from the remaining 16.7% that answered ‘Other’ said that the HD 8266 might be helpful for some circumstances, but they had the money and knowledge to use modern means of communication.

In conclusion, the simplicity and price of the HD 8266 could help many people unable to afford or understand the latest technology.

I have also programmed the prototype system so more devices could easily be added to the system. When a new device is made, it will get a unique code such as “1836”. When

that code is typed into the site, the user will be able to change their contact list and message without interfering with someone else's device data. The prototyped version of the product I made during the timeline of the competition only has one button, but if I were to sell the HD 8266 I would add another button. I could also have the device itself host a website, so you could configure the WiFi password and device ID.

IV Appendix

Parts of the ESP8266:

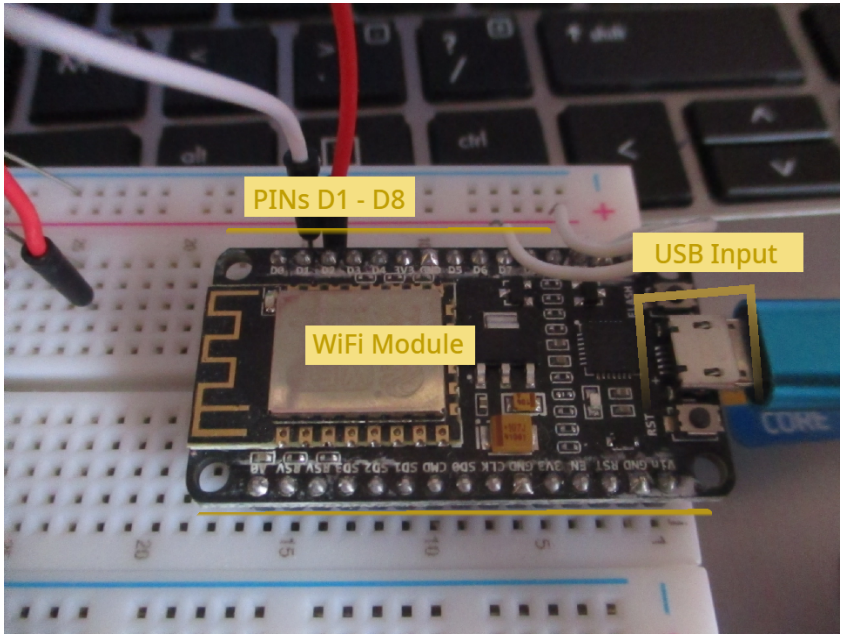


Image created by Leah Thelen.

The eight PINs are for input (or output).

There's a USB input for uploading code and powering the ESP8266 board.

The WiFi module lets the ESP8266 access the internet.

Here's the code for the ESP8266. (In C++ created with Aruino.)

```
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

#include <ESP8266HTTPClient.h>

#include <WiFiClientSecureBearSSL.h>

// constants won't change. They're used here to set pin numbers:
const int buttonPin = 5;     // the number of the pushbutton pin
const int ledPin =  4;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status



ESP8266WiFiMulti WiFiMulti;

void setup() {

   // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
```

```
  Serial.begin(115200);
  // Serial.setDebugOutput(true);

  Serial.println();
  Serial.println();
  Serial.println();

  for (uint8_t t = 4; t > 0; t--) {
    Serial.printf("[SETUP] WAIT %d...\n", t);
    Serial.flush();
    delay(1000);
  }

  WiFi.mode(WIFI_STA);
  WiFiMulti.addAP("REMOVED FOR SECURITY", "REMOVED FOR SECURITY");



}


void PostHttp(String address, String jsonMessage) {
  // wait for WiFi connection
  if ((WiFiMulti.run() == WL_CONNECTED)) {

    WiFiClientSecure client;

    client.setInsecure();
    HTTPClient https;
```

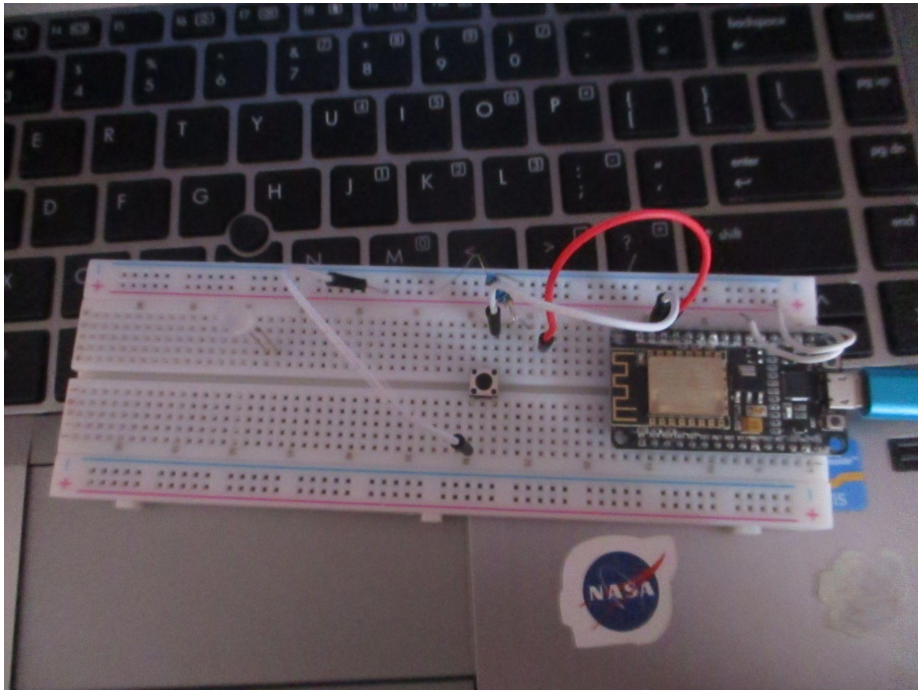
```
    Serial.print("[HTTP] begin...\n");
    // configure traged server and url

    //https.begin(*client, address); //HTTPS
    https.begin(client, address); //HTTPS
    //https.addHeader("Content-Type", "application/json");



    Serial.print("[HTTP] POST...\n");
    // start connection and send HTTP header and body
    int httpCode = https.POST(jsonMessage);

    // httpCode will be negative on error
    if (httpCode > 0) {
      // HTTP header has been send and Server response header has been handled
      Serial.printf("[HTTP] POST... code: %d\n", httpCode);

      // file found at server
      if (httpCode == HTTP_CODE_OK) {
        const String& payload = https.getString();
        Serial.println("received payload:\n<<");
        Serial.println(payload);
        Serial.println(">>");
      }
    } else {
      Serial.printf("[HTTP] POST... failed, error: %s\n",
https.errorToString(httpCode).c_str());
    }
```

```
    https.end();
  }
}


void loop() {

  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);


    PostHttp("https://leahshd.azurewebsites.net/Home/SendMessage?ID=1", "");
    delay(5000);

    Serial.printf("Waiting 5s (to wait for user to release button)...");
    delay(5000);



  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
```

```
}//end loop()
```

The two “REMOVED FOR SECURITY” sections stand for the name and password for my home’s internet connection.

This code was created from a mixture of two demos that come with the Arduino package.



ESP8266 prototype board.

Here’s the code for the Home Controller (written in C#):

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
```

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using LeahsHelpDevice.Models;
using LeahsHelpDevice.Data;
using Microsoft.EntityFrameworkCore;

namespace LeahsHelpDevice.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly AppDbContext _db;

        public HomeController(ILogger<HomeController> logger, AppDbContext db)
        {
            _logger = logger;
            _db = db;
        }

        public IActionResult Index()
        {

            var result = _db.Devices.Include(x => x.Contacts).ToList();

            string buff = "";
            foreach (var device in result)
            {
                buff += $"DeviceID: {device.ID} Owner: {device.OwnerName}<br>";
                foreach (var c in device.Contacts)
```

```
                {
                    buff += $"Contact: {c.PhoneNumber}<br>";
                }
            }
            ViewBag.TempMessage = buff;

            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }




        public IActionResult PrivatePage(int ID)
        {

            var result = _db.Devices
                .Include(x => x.Contacts)
```

```
                .FirstOrDefault(x => x.ID == ID);
            if (result == null) return RedirectToAction("Index");

            ViewBag.OwnerName = result.OwnerName;
            ViewBag.DeviceID = ID;
            ViewBag.Contacts = result.Contacts;


            return View();
        }


        [HttpPost]
        public IActionResult SendMessage(int ID)
        {

            var message = _db.Devices.FirstOrDefault(x => x.ID == ID);
            var myDevice = _db.Devices.Include(x => x.Contacts).FirstOrDefault(x => x.ID
== ID);

            foreach (var c in myDevice.Contacts)
            {
                   var mess = "hi";
                    mess += $"Contact: {c.PhoneNumber}<br>";
                var mess2 = Code.TxtMessageHelper.SendTxtMessage(c.PhoneNumber,
message.HelpMessage);
            }


            return Content("result: " + myDevice);
```

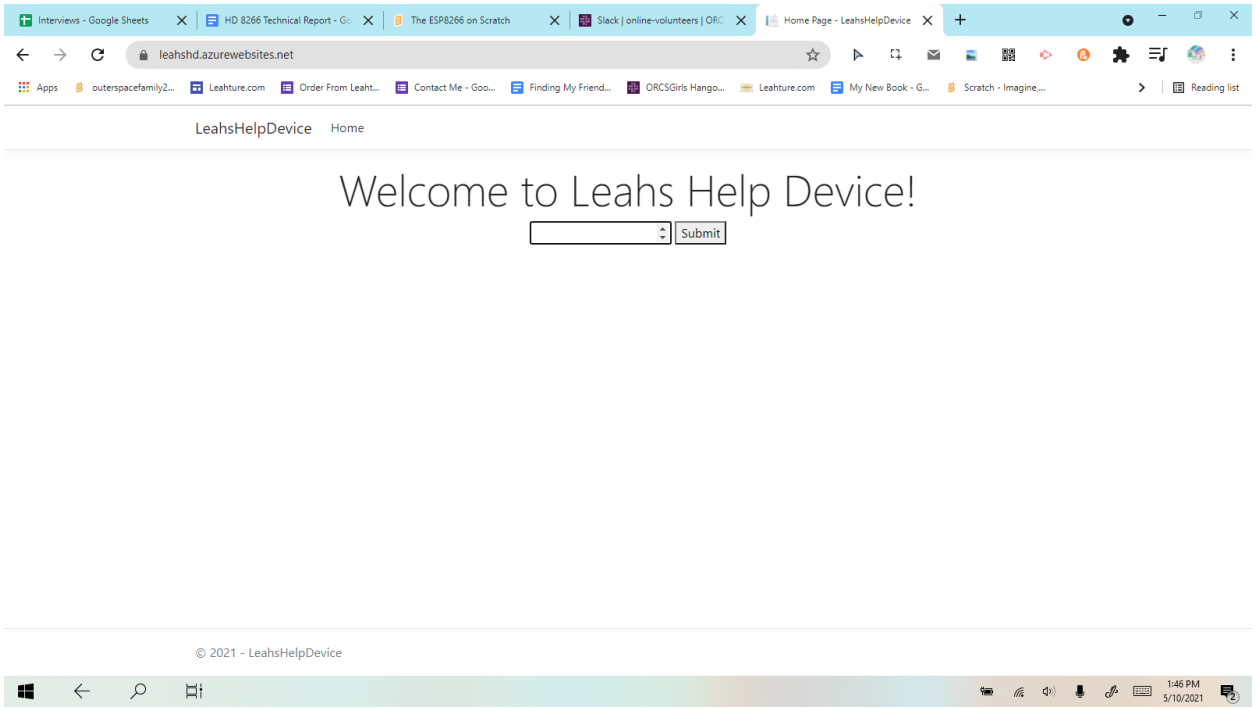
```
        }



        public IActionResult ChangeMessage(string NewMessage, int ID)
        {
            var myNewMess = _db.Devices.FirstOrDefault(x => x.ID == ID);
            myNewMess.HelpMessage = NewMessage;
            _db.SaveChanges();
            return RedirectToAction("PrivatePage", new { ID = ID });

        }



        public IActionResult AddContact(string PhoneNumber, int ID)
        {
            var dc = new DeviceContact();
            dc.DeviceID = ID;
            dc.PhoneNumber = PhoneNumber;

            var myDevice = _db.Devices.Include(x => x.Contacts).FirstOrDefault(x => x.ID
== ID);
            myDevice.Contacts.Add(dc);
            _db.SaveChanges();

            return RedirectToAction("PrivatePage", new { ID = ID });
        }

        public IActionResult DeleteContact(int ContactID, int ID)
        {
```

```
            var contact = _db.DeviceContacts.FirstOrDefault(x => x.ID == ContactID);
            if (contact != null)
            {
                _db.DeviceContacts.Remove(contact);
                _db.SaveChanges();
            }
            return RedirectToAction("PrivatePage", new { ID = ID });
        }

    }
}
```

Here's the code for the Index page HTML:

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome to Leahs Help Device!</h1>

    <form action="@Url.Action("PrivatePage")" method="get">
    <input type="number" name="ID" />
    <button type="submit">Submit</button>
   </form>
    <div style="color: red;">@TempData["ErrorMessage"]</div>
```

```
</div>
```



Screenshot of Index (Home) page.

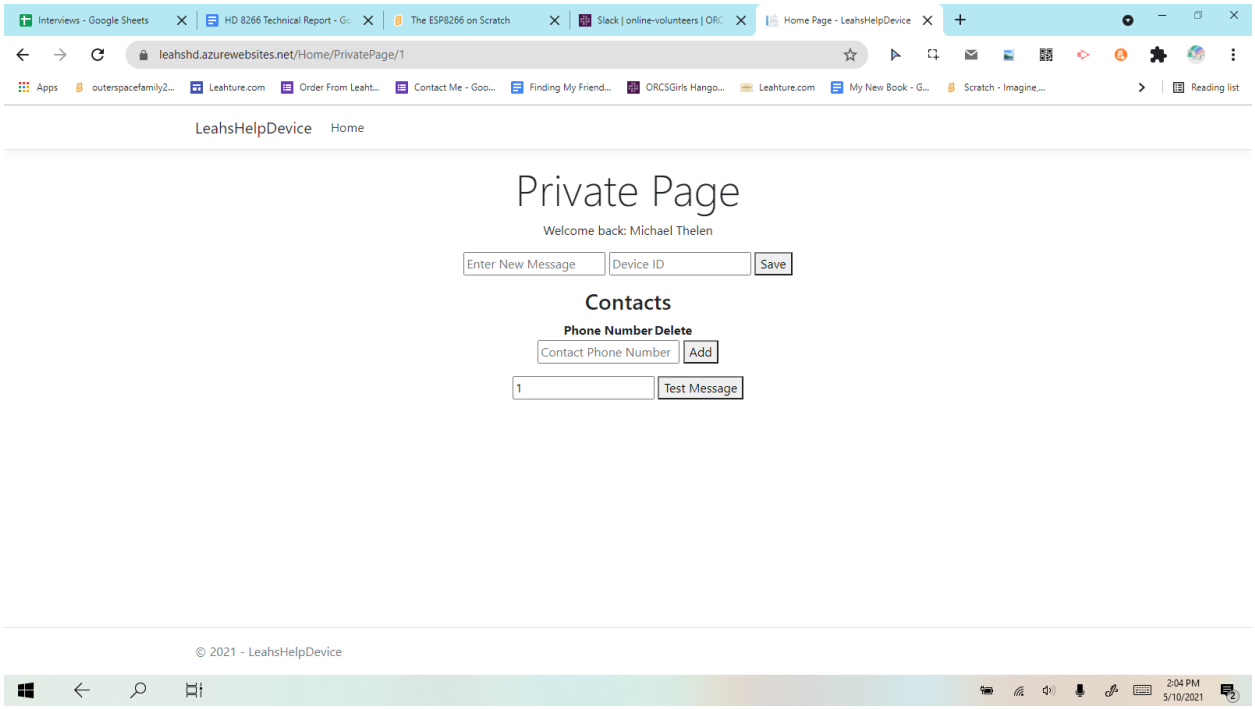
Here is the HTML code for the Private Page:

```
@{
    ViewData["Title"] = "Home Page";
}
```

```
<div class="text-center">
    <h1 class="display-4">Private Page</h1>


    <div style="color: red;">@TempData["ErrorMessage"]</div>



    <p>Welcome back: @ViewBag.OwnerName</p>


    <form action="@Url.Action("ChangeMessage")" method="post">
        <input type="text" placeholder ="Enter New Message" name="NewMessage"/>
        <input type="number" placeholder ="Device ID" name="ID"/>
        <button type="submit">Save</button>
    </form>

  <p>

  <p/>

  <h3>Contacts</h3>
  <table style="margin: auto;">

        <tr>
            <th>Phone Number</th>
            <th>Delete</th>
        </tr>
```

```
        @foreach(var contact in ViewBag.Contacts)
        {
        <tr>
            <td>@contact.PhoneNumber</td>
            <td>
                <form action="@Url.Action("DeleteContact")" method="post">
                    <input type="hidden" name="ID" value="@ViewBag.DeviceID" />
                    <input type="hidden" name="ContactID" value="@contact.ID" />
                    <button type="submit">Delete</button>
                </form>
            </td>
        </tr>
        }

  </table>




    <form action="@Url.Action("AddContact")" method="post">
        <input type="text" placeholder ="Contact Phone Number" name="PhoneNumber"/>
        <input type="hidden" name="ID" value="@ViewBag.DeviceID" />
        <button type="submit">Add</button>
    </form>


    <p>

<p/>
```

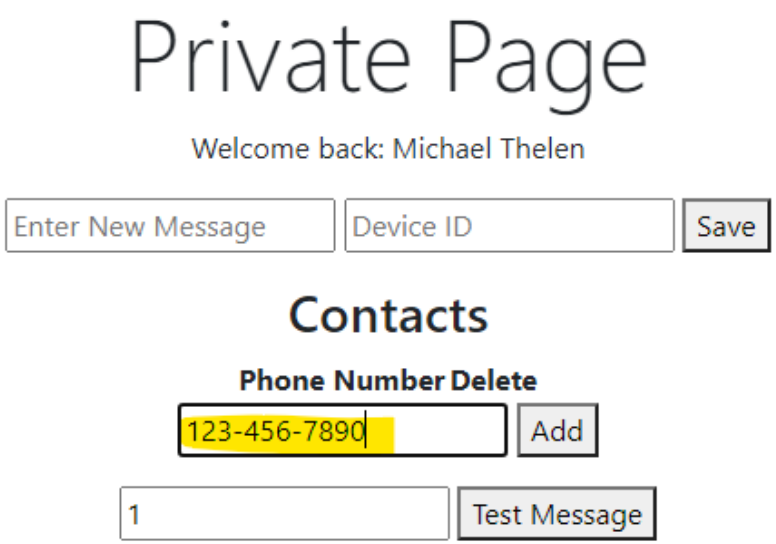
```
    <form action="@Url.Action("SendMessage")" method="post">
        <input type="text" placeholder ="Device ID" name="ID" value="@ViewBag.DeviceID"/>
        <button type="submit">Test Message</button>
    </form>


    <div style="color: powderblue;">@TempData["ErrorMessage"]</div>

</div>

<script>



</script>
```



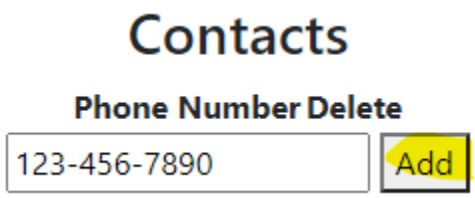
'Private Page' screenshot.

Steps to add contact -

1. Type in phone number:

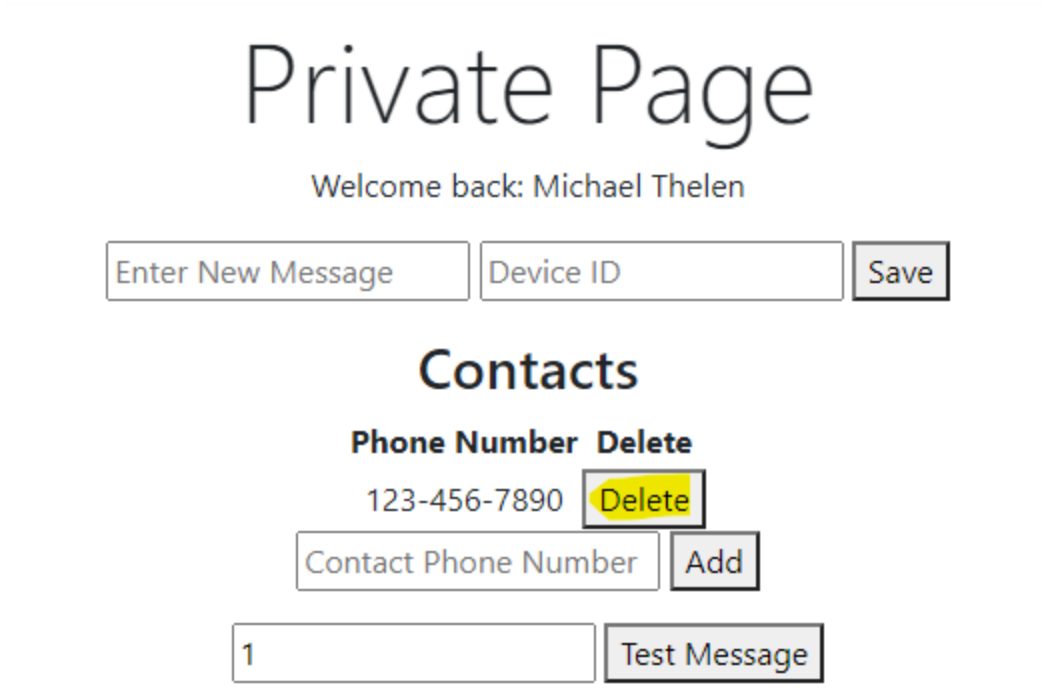


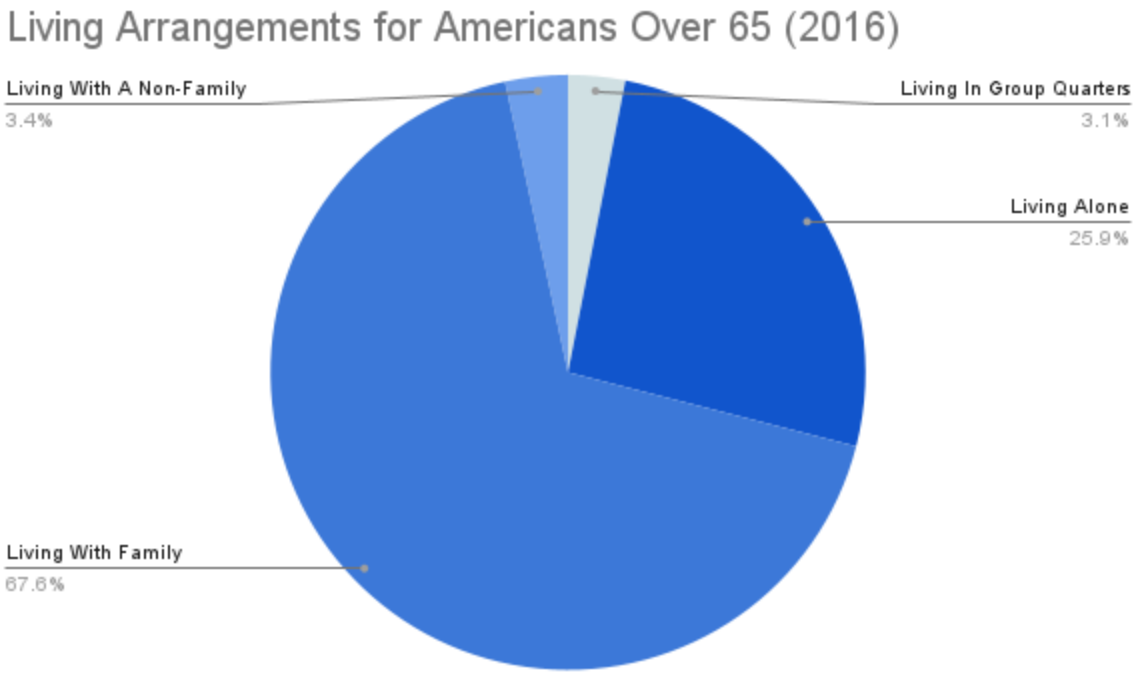
2. Press “Add”.



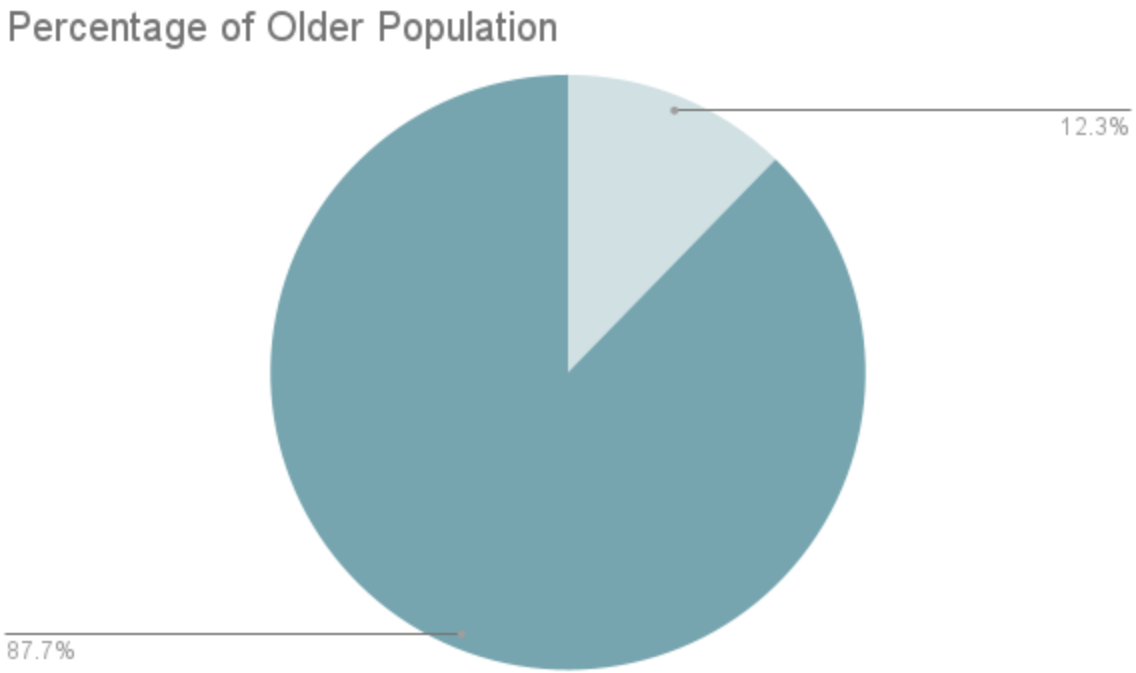
How to delete a contact -

1. Press “Delete”.


Private Page
Welcome back: Michael Thelen
Enter New Message
Device ID
Save
Contacts
Phone Number
Delete
123-456-7890
Delete
Contact Phone Number
Add
1
Test Message



Information from https://census.gov (Created with Google Sheets)



Information from https://www.ruralhealthinfo.org/toolkits/aging/1/demographics and https://datacommons.org/ Light blue = younger, Dark blue = older
(Created with Google Sheets)