Code Manual

The Martian Project

Connor Finnegan, Evenson Charles, Andrew De Asevedo Disha Roopun

Set up

To set up the front and the back end, all you need to do is cd into the respective directories, install all the packages and then use npm to run them. Ex: To run the front end:

```
cd frontend
npm install
npm run dev
```

To run the backend:

First step is to log into MongoDB and verify your IP address, and open up the .env file in the backend and fill it out accordingly. Here is an example of mine. Don't leak my IP please :)

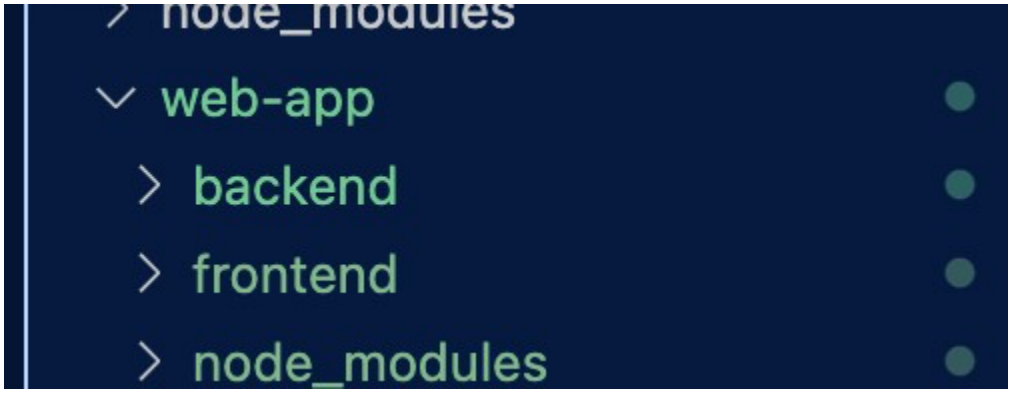
```
# TODO: set your username and password. Also alter port if needed. JWT_SECRET variable is pinned in the Discord.
MONGO_URI=mongodb+srv://finny827:YCImttMb5smz8IAd@cluster0.21donbl.mongodb.net/sample_mflix?retryWrites=true&w=majority&appName=Cluster0
PORT=5001
JWT_SECRET=yBf49Kz8vWp@3!zA?nLsQr7D^tXcJ0*EhGmU

```

Then run these simple commands

```
cd backend
npm install
npm start
```

Now by visiting localhost:5173 in your browser you will see the front end. With the backend running you should also be able to register an account on the dashboard, and login.

File Structure



I've separated the app into two directories. The backend, and the frontend.

Frontend directory
This directory houses all of the frontend content. Most of this is skeleton code that comes when you build a react app. So ignore all of it except for everything in the src directory and the app.jsx and main.jsx

I built this app to be component based as it is the most convenient way to work. So if you open the component directory you will see every single component as long as their CSS.

Very little is actually rendered onto the pages themself but if you open the pages directory you'll see the individual big pages. Since I built it based off components, each static page is relatively simple and short in terms of lengths of code.

Ex: The Homepage

```
Windsurf: Refactor | Explain | Generate JSDoc | ×
function Home() {
  return (
    <>
      <Header />
      <div className='logoContainer'>
        <h1 className='title'>The <br /> Martian Project</h1>
      </div>
      <AboutSection id="about" />
      <Pricing />

      <ParticleBackground />
    </>
  );
}

export default Home;
```

Lastly, the App.jsx file builds the routes for each of these pages.

Backend directory

The backend for this web app is much simpler as all it does is send and pull data from the database. server.js

This file contains all the routes and middleware to the backend. In the MERN stack, this file is pretty much what creates the Express server to connect the React to the MongoDB.

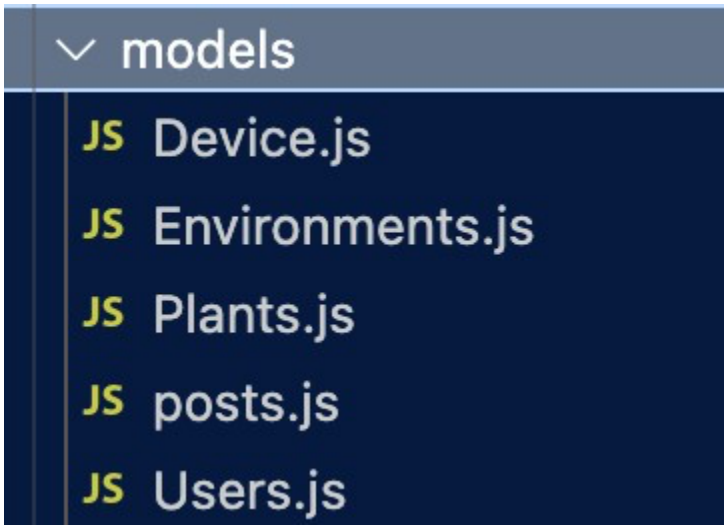
We have four routes in use currently.

```
// Routes
app.use("/api/users", usersRoutes);
app.use("/api/posts", postsRoutes);
app.use("/api/environments", environmentsRoutes);
app.use('/api/plants', plantRoutes);

app.use("/api/device", devicesRoutes);
```

There is an extra devicesRoutes however, it isn't used in this project yet.

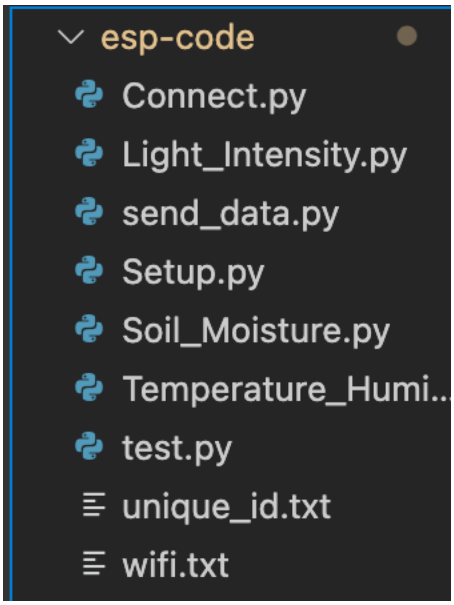
These routes send data to the respective collection on mongoDB.

We also have 5 models,



As of right now, the only models that we use to create the data is the Users (for authentication) and the Plants (for the information from the hardware).

For the ESP the file structure on the board is just all the files just in root directory, in our project directory we distinguish by adding an Esp-code folder so that code is visible with all of our other code.



Our code starts in test.py, in a production setting this would be main.py so every time the board is plugged in the board would just start running the software. Each of the three sensors get their own file, each with a basic function that collects data upon a call, for example here is the code for the light sensor.

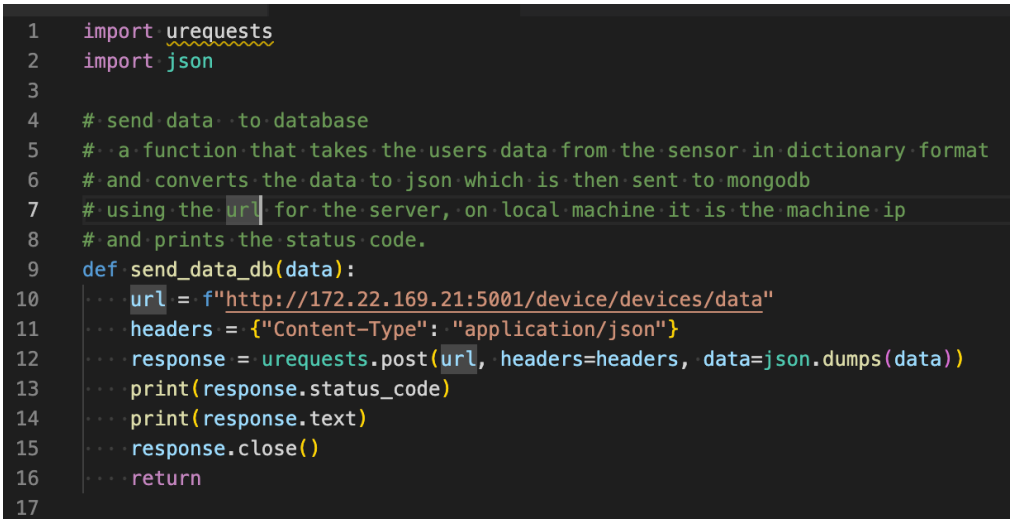
Light_Intensity.py 1

```python
from machine import ADC, Pin
import time

# Use GPIO 36 for the adc pin
light_sensor = ADC(Pin(36))
#light_sensor.atten(ADC.ATTN_11DB)
light_sensor.width(ADC.WIDTH_12BIT)

# Read_light
# a function that reads the value measured from the lgiht sensor
# and returns the light value and then sleep for X seconds(default 2 seconds)
def read_light(time_sleep=2):
    light_val = light_sensor.read()
    print("Light intensity:", light_val)
    time.sleep(time_sleep)
    return light_val
```

For each sensor file the sensors pins are created as global variables and then the function contains the logic for collecting data, here we use read() to get the raw value given to us by the sensor.

Before the sensors collect data, we first have to make sure the user is connected to wifi. The entry point for this is wifi.txt where we have a dummy ssid and password when the board initially tries to connect. When this fails, the user is then prompted to connect to the boards network “plant-monitoring” and enter the password “plant-monitoring123”. This is done in Setup.py

Then the user enters their and then we go back to Connect.py where the board attempts to connect to the wifi with the new ssid and password. Once connected the board collects data and then sends it to the server with a simple post request inside a send data function:



```python
import urequests
import json

# send data to database
# a function that takes the users data from the sensor in dictionary format
# and converts the data to json which is then sent to mongodb
# using the url for the server, on local machine it is the machine ip
# and prints the status code.
def send_data_db(data):
    url = f"http://172.22.169.21:5001/device/devices/data"
    headers = {"Content-Type": "application/json"}
    response = urequests.post(url, headers=headers, data=json.dumps(data))
    print(response.status_code)
    print(response.text)
    response.close()
    return

```

Once the data is collected we sleep for some amount of time which is defaulted to 2 minutes and then after we run the main function again:

```python
import Temperature_Humidity as TR
import Light_Intensity as LI
import Soil_Moisture as SM
from send_data import send_data_db
import time
import network
import time
import Connect


# main
# the main function for the entire esp system.
# checs if users are connnected to the wifi and if they are then the sensors start collecting data
# then they send the data to the send data function and then the process sleeps for 5 minutes
def main():
    if Connect.connect_wifi():
        print("Main program start here!")
        while True:
            temp, humidity = TR.get_temp()
            light_intensity = LI.read_light()
            soil_moisture= SM.read_soil_moisture()
            data = [temp, humidity, light_intensity, soil_moisture]
            send_data_db(data)
            time.sleep(300)
```