CSCI 43700
Internet of Things
12/4/2018

Trenton Spice
tmspice@iupui.edu
2000056479

Kyle Peeler
kapeeler@iupui.edu
0002845885

Smart Alarm Clock

Introduction

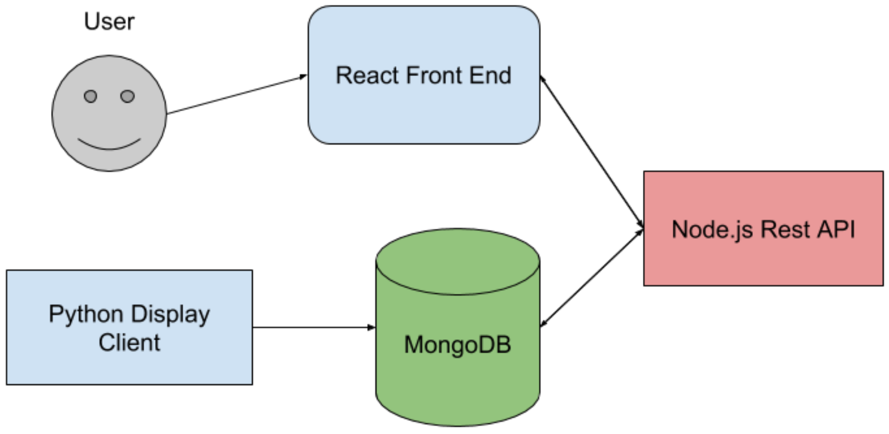
The IOT Smart Alarm Clock is a device whose purpose is to make waking up in the morning easier with features that provide useful information (other than just being an alarm clock) to the user. We will display information such as current time and other useful information (e.g. weather) on a 32x32 RGB LED matrix panel. We are allowing the user to customize what they would like to show on the display, as well as customization options (e.g. change text color of any widget). The user can also set various alarms, which can be repeated any day of the week. We plan to provide the user with a few alarm sounds that can be set as the default sound played when an alarm goes off as well as a flash sequence. The matrix will fill completely red when the alarm goes off. All of these settings can be configured in our web application and will be stored in a database so user configuration data persists. All alarm records are also saved so the user will not lose any data pertaining to alarms in the case of a power failure or device being restarted. This report will outline the technology and hardware we used in order to complete this project. It will also feature material on how we went about implementing our ideas and some of the problems we encountered along the way.

Widgets

We implemented a total of five widgets that the user can choose to display on the matrix. The widgets are as follows: Time, Next Alarm, Weather, Date, and Text. There are 4 slots on the display and the user can choose which widgets they would like. Time is a widget that displays the current time, we allow for 24 hour time format as a property the user can enable. Next Alarm is a widget that shows the time of your next alarm, it can be configured to show either the time of your next alarm (e.g. 7:15 AM) or it can show how much time is left until your alarm will go off (e.g. 6 hours, 15 minutes). Weather is a widget that shows weather data based on the OWM API given a string of a city name. Date is a widget that shows the current date, it can be configured to show the full date (e.g. Wednesday, December 5th 2018) otherwise it just displays in a mm/dd/yyyy format. Text is a widget that displays a user defined string of text on the matrix. All of these widgets have a configurable property of some kind, the user can also choose what is displayed on the matrix and the position the widget will show at. All of the widgets also allow the user to choose the color that will show on the display, which is an selectable RGB value.

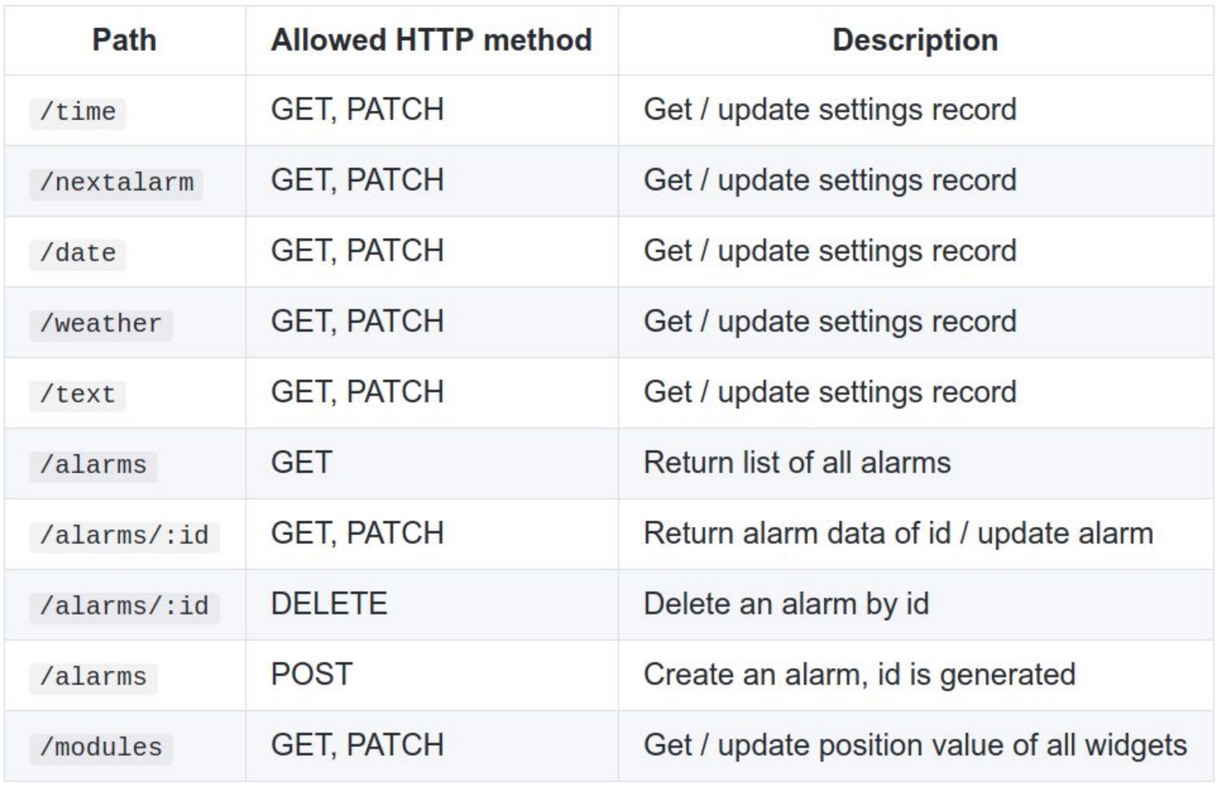
Technology

For this project we used various technologies such as Node.JS, React, and Python, and MongoDB. We developed a total of 3 repositories, all of which will be briefly discussed to address functionality to the project and how it uses the HTTP protocol to transfer data among these services. We used a non-relational MongoDB data store which allowed us to hit the ground running quickly without needing a fully complete data design from the start. Below is a diagram that clearly identifies all of our services and how they interact.



REST API Server

We developed our API with Node.js and Express.js in order to have a RESTful way to get data from a user web application view to the matrix as a displayed widget. To save and access records from the MongoDB store we used Mongoose. With our current setup, all of these services run locally on the users machine, but going forward these services could be implemented with a service like AWS which would allow configuration on any network since the services would be hosted on a remote server. Our project technologies use HTTP in order to do CRUD operations based on data routes. It is an MVC (Model View Controller) design and our data is designed to work with all of the services. We used Postman to test our API endpoints to ensure everything was working as expected. The diagram below outlines all of our API routes that can be accessed to operate with the data.

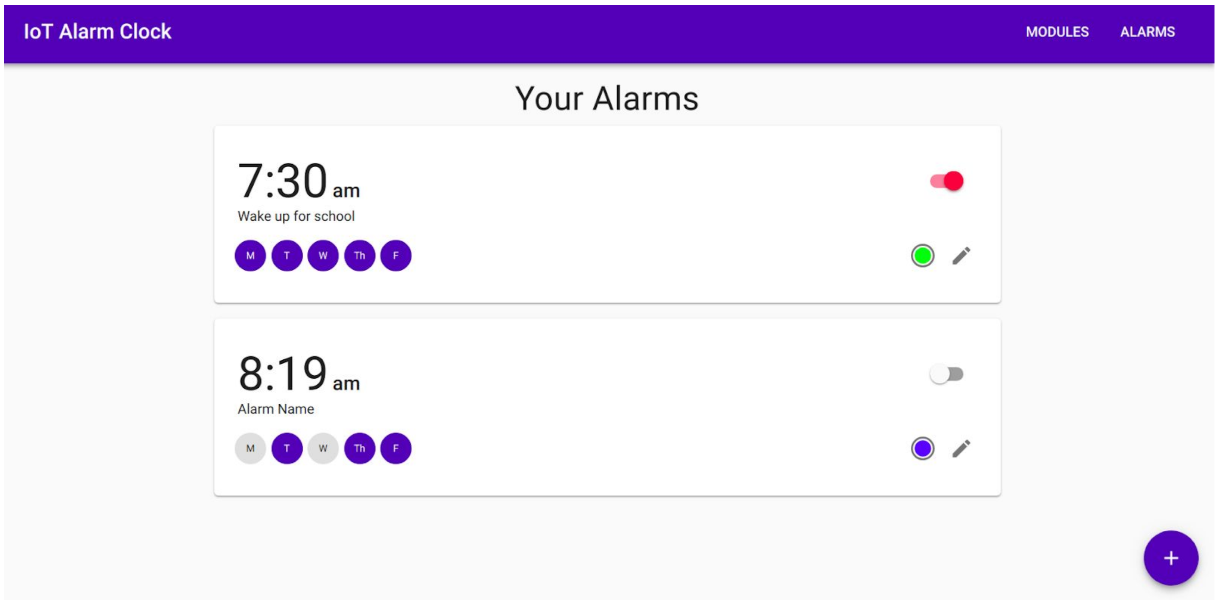


Path	Allowed HTTP method	Description
`/time`	GET, PATCH	Get / update settings record
`/nextalarm`	GET, PATCH	Get / update settings record
`/date`	GET, PATCH	Get / update settings record
`/weather`	GET, PATCH	Get / update settings record
`/text`	GET, PATCH	Get / update settings record
`/alarms`	GET	Return list of all alarms
`/alarms/:id`	GET, PATCH	Return alarm data of id / update alarm
`/alarms/:id`	DELETE	Delete an alarm by id
`/alarms`	POST	Create an alarm, id is generated
`/modules`	GET, PATCH	Get / update position value of all widgets

Front End Web Application

We developed our front end web application with React, Webpack, and Material-UI. The purpose of this service is to allow the user to customize which widgets they would like on the display as well as configure all alarms that will be registered to the display itself. All of the widgets are customizable within the application and all of their respective properties can be changed using the settings cog. To determine where the widgets are displayed on the matrix, we implemented a drag and drop system to directly set the position values. The drag and drop system also has two sections, one for active widgets and the other for inactive widgets (not shown on the matrix). The user can also define any number of alarms, of which can be set to be enable/disabled, repeatable on any given day (Monday - Sunday), and the color it will render on the display. By using React we were granted hot reloading functionality meaning there is minimal page reloading when updating settings which allows for really smooth user interaction. Below are a few screenshots of the web application highlighting some of the features it provides.

Alarm View

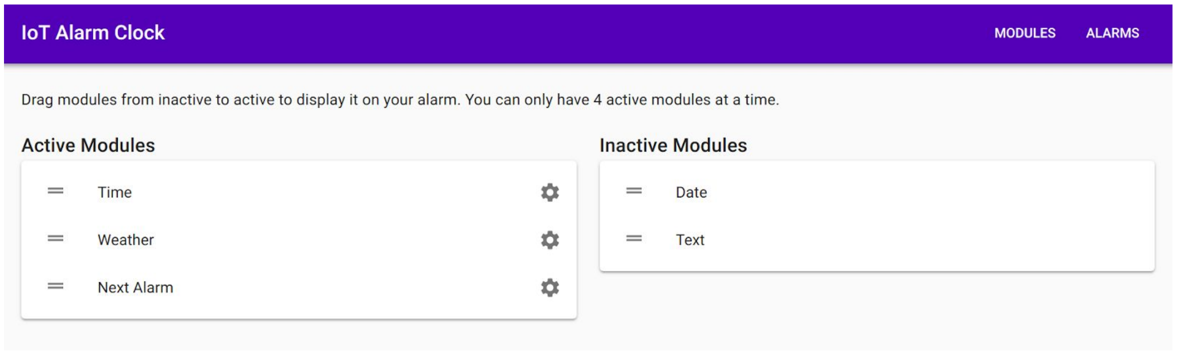


Your Alarms

7:30 am
Wake up for school
M T W Th F

8:19 am
Alarm Name
M T W Th F

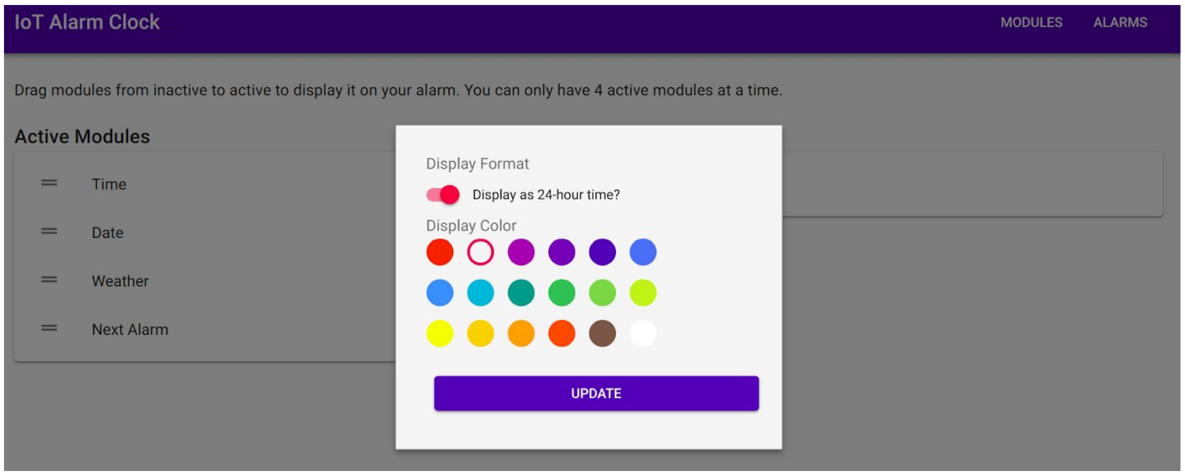
Module View



Drag modules from inactive to active to display it on your alarm. You can only have 4 active modules at a time.

Active Modules	Inactive Modules
Time	Date
Weather	Text
Next Alarm	

Time Widget Settings Modal



Drag modules from inactive to active to display it on your alarm. You can only have 4 active modules at a time.

Active Modules

Time
Date
Weather
Next Alarm

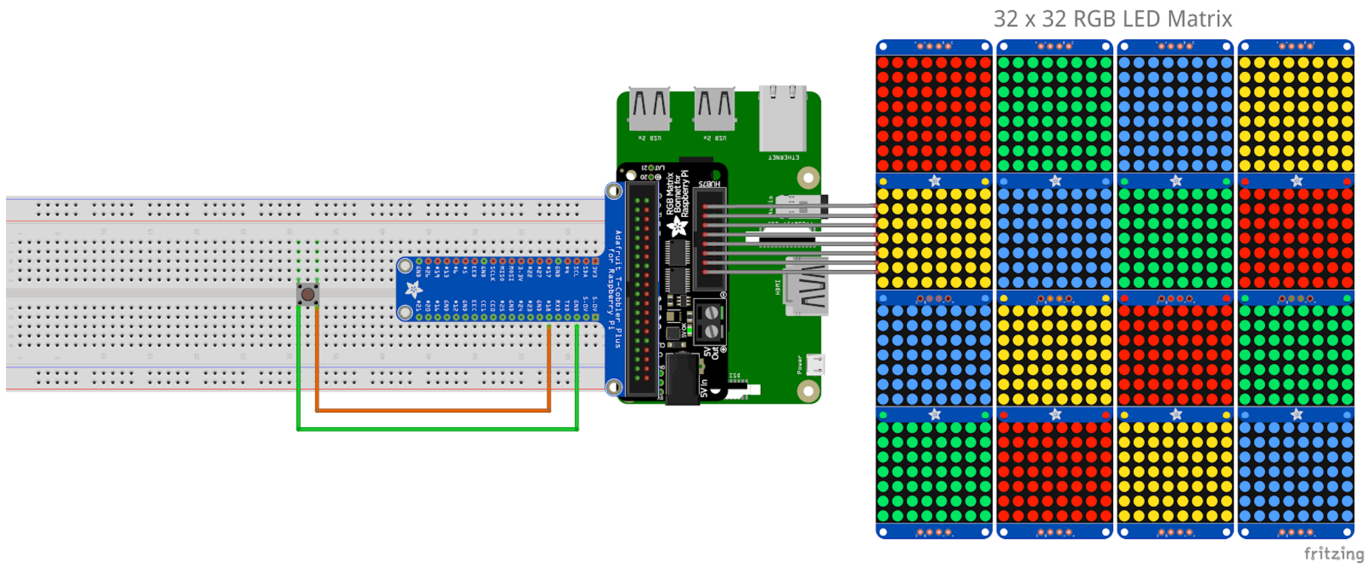
Display Format
Display as 24-hour time?
Display Color

Python Display Client

We developed our python application purposed to display data on the 32x32 RGB LED matrix. To accomplish this we used the Adafruit rpi-rgb-led-matrix library which has a python wrapper built on top of a C library (also supports Arduino) in order to draw graphics on the matrix. To query the MongoDB database we used a library called Pymongo which allowed us to directly query collections. We also used pyowm which is a wrapper for the OWM weather API which allows us to query for weather information based on a city string. All of the configuration settings set by the front end are automatically displayed on the matrix based on position properties. Only four widgets can be displayed on the matrix at one time, so we use position values between 1-4. A position of -1 indicates that said widget is not enabled. We also built out our alarm functionality on the python client which queries for alarms set on the current day. For example, if today is Wednesday, then the alarm will find records that are enabled and its day property is true for Wednesday. It will then do comparisons based on current system time and the set alarm time, if it matches then the matrix will flash on and off and will play a user determined alarm sound.

Hardware

This project uses a Raspberry Pi 3 alongside an RGB Matrix Bonnet pHAT (no EEPROM), a button, and the 32x32 RGB LED Matrix. We also used the T-Cobbler ribbon that allowed us to use a solderless breadboard for the button. The buttons purpose is to set off / snooze the alarm if it is active. We had to investigate the pinout diagrams for the RGB Matrix pHAT to ensure we were not using any GPIO pins already in use. The RGB Matrix pHAT requires a 5V power supply. It connects to the RGB LED matrix using a ribbon IDC cable and a power cable. The RGB Matrix pHAT uses a total of 14 GPIO pins to send data to the matrix respectively. We used the official adafruit fritzing diagrams to get all of the components needed to design a schematic. A screenshot of our circuit schematic can be found below.



Problems

The first major issue we ran into when identifying the hardware for the project was trying to use two different HAT's initially. One being the RGB LED Matrix Bonnet and then we wanted to use a Speaker HAT both for the raspberry pi. The problem was that there was GPIO pin conflicts which would not allow us to use both concurrently. Although for this project we used the default sound input on the Raspberry Pi at the system level to play an alarm sound, we intend to build this into a standalone device after this semester in which we will use a Class D amplifier in order to use GPIO pins to output to small speakers. We also ran into problems when trying to run MongoDB on the Raspberry Pi itself, our Raspbian 32 bit OS would not support it. To fix this, we used a docker image of MongoDB 3.0 which worked with our 32 bit Raspbian OS. Obviously these services could be placed on a remote server, which would alleviate this issue, but for local development we had to use this method in order to get everything working. Since the Raspberry Pi has limited amounts of onboard memory, we noticed flickering in the matrix LED's when the system was under heavy load, we even had our device freeze up completely multiple times. This again can be alleviated by running our heavy services on remote servers and configuring the Raspberry Pi to make external HTTP requests, opposed to running everything on localhost. In our project proposal we also mentioned a 3D printed case that would enclose the matrix and all of our hardware, however due to time constraints, this has not yet been implemented. But it will likely come to fruition soon, as we both plan to use this as our own alarm clocks!

Sources

Project Repositories

- https://github.com/trentspi/iot-alarm-api-server - API Server (documentation in README)
- https://github.com/kylepeeler/IoT-Alarm-Front-End - Front end App
- https://github.com/kylepeeler/iot-alarm-python-client - Python client

Other sources

- https://github.com/adafruit/Fritzing-Library - Fritzing library used for schematics
- https://pinout.xyz/pinout/rgb_matrix_bonnet - Pinout website
- https://openweathermap.org/ - OWM API for weather data
- https://api.mongodb.com/python/current/ - PyMongo
- https://github.com/hzeller/rpi-rgb-led-matrix - RGB LED Matrix Library
- https://www.adafruit.com/product/1484 - Matrix Panel
- https://www.adafruit.com/product/3211 - RGB LED Matrix Bonnet pHAT
- https://nodejs.org/en/ - Node JS
- https://expressjs.com/ - Express JS
- https://mongoosejs.com/ - Mongoose JS
- https://reactjs.org/ - React JS
- https://material-ui.com/ - Material UI
- https://www.mongodb.com/ - MongoDB
- https://freesound.org/people/kwahmah_02/sounds/250629/ - Alarm sound