Improvements

Project 2

Car Traffic Counter

For this second project, we received the Wyoming legacy code. In their APIs, they use pyFirmata to run the Arduino. They also include PHP files that show the average between certain ranges along with a graph that can only show the total amount of people daily.

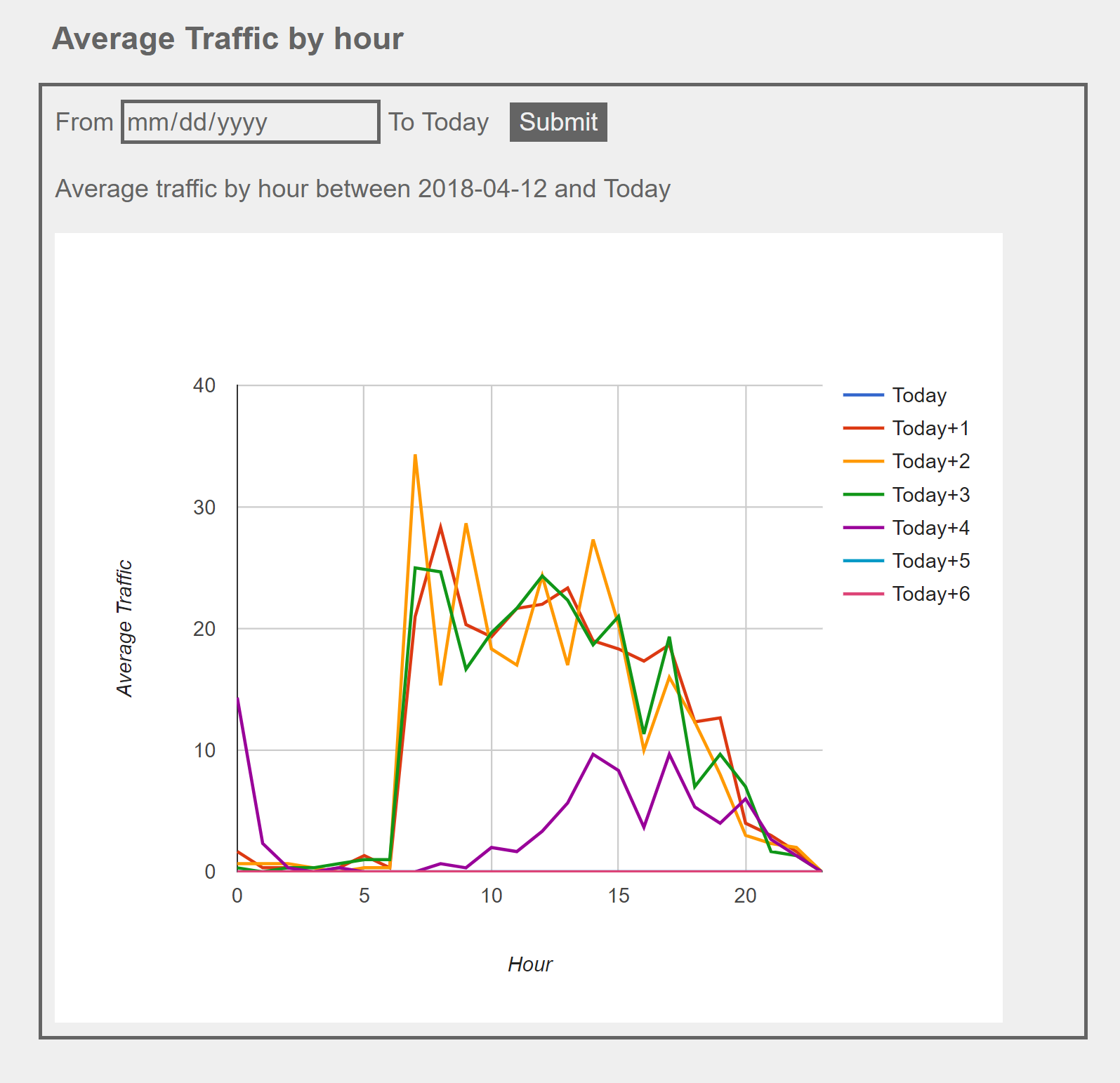
Essentially, we are unsatisfied with their code and changed many items. We decided to completely abandon their pyFirmata code and instead put everything into Sketch and C code. We did not accomplish the first project with pyFirmata and decided it would be much better to not use pyFirmata in this project as well. Essentially, pyFirmata requires using Python code, which is not allowed for this project. A much more extensible interface can be achieved through C. In particular, if each Arduino sends its data on a unique serial port, a C interface can interpret these uniquely as well and read their data synchronously. Another important aspect of the C interface is its ability to work with the database. We use a struct that includes necessary information such as port name, type of ingress/egress, serial port FILE \* pointer, location of the Arduino, database name, and mutex to protect the access of the database. This allows us to read each data stream uniquely and operate on the data in a thread-safe manner without accidentally changing or erasing other threads’ attempts to access the database.

Since Wyoming’s Arduino code was primarily based on the pyFirmata framework, we could not use any of their Arduino code. Their Arduino code utilized the FirmataBasic.ino file, which is not extensible for our purposes. The Arduino code simply operates under a handshake paradigm to start, where each Arduino makes a single Raspberry Pi aware of its existence and location, then sends data to its unique serial port for processing and action.

We also use mutex to lock and unlock the access to the database. Since each thread accesses the cURL process to send a POST request to our curl\_insert.php webpage, this must be protected on the C side to ensure that no process accidentally overwrites another processes’ request on the website.

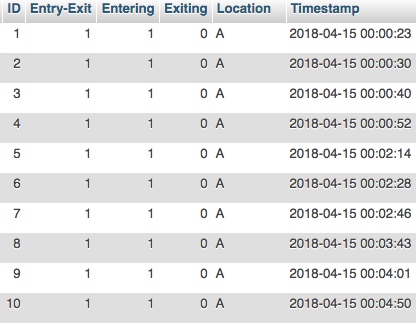
We feel like their PHP code is poorly documented. We feel like they could have done better by including more comments. Their main php code, the data.php has no comment at all, except the header. It was rather complicated to modify the code as they echoed different items rather than simply outputting them. Instead of putting the code in a function, they call random names and count it straight away in the data.php, which looks terrible and is hard to understand. Their main API file, database.php, is sparse and not useful as an extensible API since it is very much hardwired for the original group’s table interface. In their website, however, they wrote many important functions from the base API that would have been useful to include as a part of the API. This lets us not have to touch much of their code in terms of getting it running, but should be cleaned up since it is so hard to understand.

For the website, we improved the interface of the website by allowing users to see the prediction model of the data collected. We created a graph where it shows prediction data with 7 lines each representing the next 7 days of the week. The basis for our prediction model is an average forecast based on the entire dataset to date from a given starting point. This allows us to handle new, common trends in the data as they begin appearing more often.

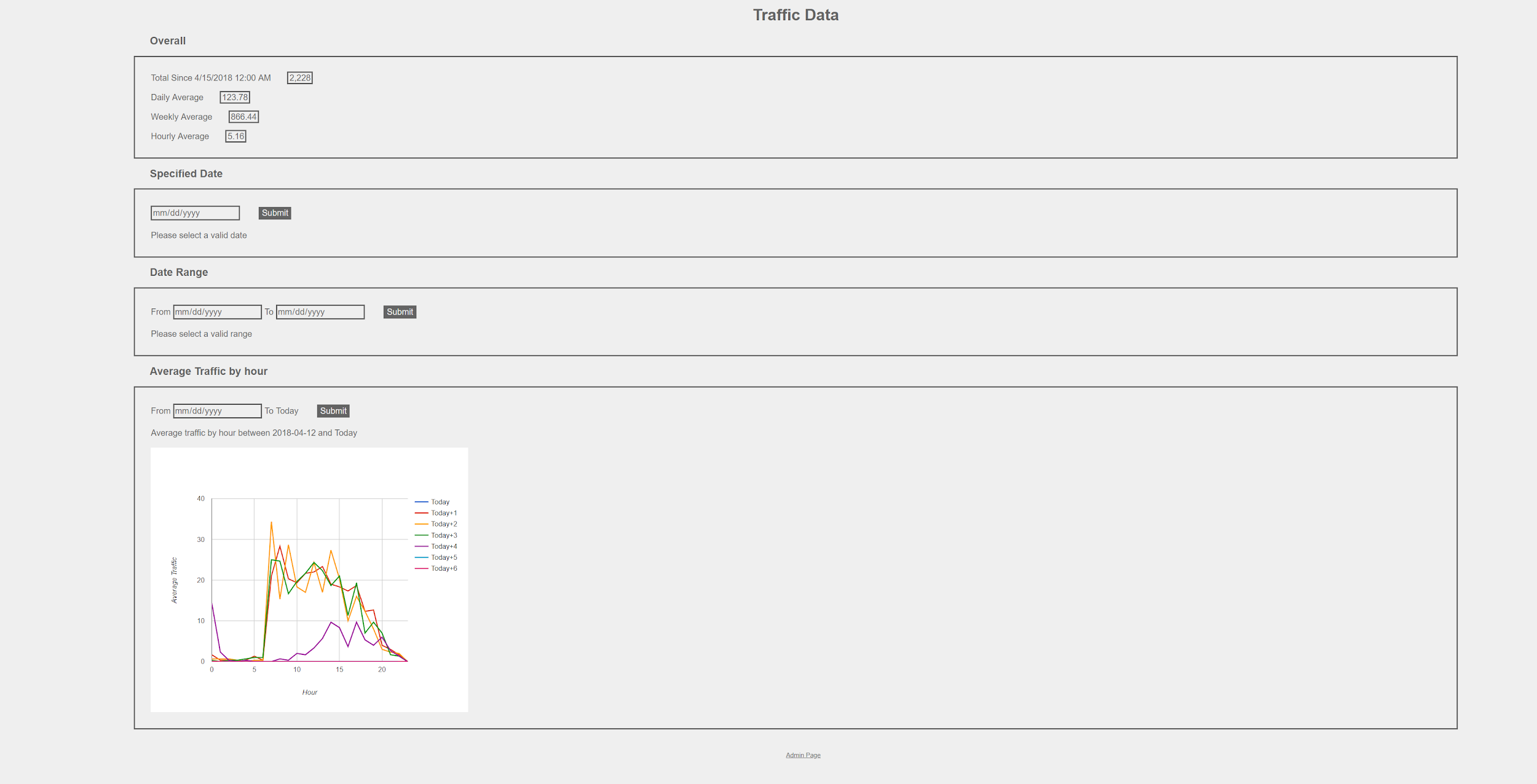


Other than that, the previous interface also displays the total amount of people coming in since data collection began. We also have the daily average, weekly average, and hourly average of people coming in from the previous group.

For the database, we set it up on phpmyadmin. In the table, we have the columns of ID, Entry-Exit, Entering, Exiting, Location, and Timestamp. When there is a car going in and is detected by the sensors, ID will increment by one and Entering is true while Exiting is false. Otherwise, when a car is going out, ID increment by one and Exiting is true, while Entering is false.



The first thing that we want out of our project is portability. In fact, using Arduino means we have a good portability. We can freely move it around, for example moving it to a different spot or change the location of the sensors. However, there is still the fact that we need constant electricity connected to the Arduino and Raspberry Pi. We feel like there can be a further improvement by adding battery just to make sure that it did not terminate right away after accidentally disconnect out of the electricity. Also, it would be much better if we can use a solar panel, which means that it does not have to be constantly connected to the electricity while detecting cars going in and out. Barring this, it would be simple to set up an email or text-messaging interface from the Raspberry Pi that lets the administrator know if any Arduino or the Raspberry Pi itself is running low on batteries. Then, an administrator or technician could pause collection and replace the batteries very quickly.



Another important aspect is reliability. Some sensors may be a faulty. We bought several extra sensors and made sure to pick two best sensors out of the packs. The most difficult aspect of this is ensuring sensor reliability over time. The sensors we use are relatively inexpensive, so it is easy to buy them in batches and verify their performance before deployment.