Team 45

Jamie Hudson

Caleigh Joyce

Ben Stueve

Zixi Pan

Retrospective

Our capstone project was brought to us by Bryce Himebaugh, of the Computer Science Department of IU. He had been working with the Hoosier National Forest for some time. Hoosier National Forest wants to be recognized as a certified International Dark Sky Park. This distinction attracts star lovers and astronomers alike. To be able to gain this recognition, Hoosier National Forest needed to track light pollution across the forest. They wanted sensors to be built that would be gathering light readings throughout the day. They also needed a website to curate this data to the public. This was the basic idea for our capstone project.

When we set out to do this project, we had a good idea of what we wanted to include in our site with the help of Hoosier National Forest and Bryce. We wanted to have a map of Hoosier National Forest with the sensor locations marked. This map would also be able to display sensor data and have a heat overlay. We also wanted to be able to select specific sensors. We also wanted the user to be able to download all the sensor data as a csv. Lower tiered priorities included the ability to display current weather data and get directions to a specific sensor. On the admin side of the site, we wanted there to be a sign in that a user gains access through a root user who can send out permission to anyone. The admin users are the ones that would be able to add and remove sensors from the database. Our plan was the get the site up and running and then connect to the sensors that a graduate student team was building. From there, we could take data and display it on the site.

We accomplished the majority of our goals for this site. On the homepage, there is an introduction, the map, and the Twitter feed of the Hoosier National Forest. The last thing on our homepage is a link to download all the sensor data as a csv. Our site has a map with the sensor markers on it. It has an overlay of different colors ranging from green to red to describe how bad the light pollution is, on average, at that sensor. The markers themselves also are assigned a color based on the max light reading taken the previous day. The user can click on the sensor marker to look at the most recent light readings at that sensor. They can also get directions to that sensor through Google Maps. The user can then click to find daily, weekly, and monthly graphs for the light readings at that specific sensor. The user can switch to the Graphs and Charts page which has a graph for each sensor so that the user can compare the sensors. The next tab is a Weather page that has the weather for the week in Bloomington. It also includes information about the wind, sun, and moon for the week. The last page is an About page with links to the Hoosier National Forest and the International Dark Sky Association website. On the admin side of the website, we also accomplished most of our goals. The admin site is completely separate from the main webpage because the public does not need access to the admin site. There is a root user that is able to grant permission to users so that they can become admins. Once a user logs in as an admin, the user is able to add a sensor which updates the database and the map on the main site. The admin users are also able to remove a sensor which also updates the database and the map. The admin users can see a table of all sensors in the database so they can see which are active and which are not. The root user is able to manage admin access which allows them to add and remove admin users. Overall, the site is something we are very proud of and we feel confident in the amount of things we were able to accomplish.

Admin and requesting data:

The admin end of the website combined many of the skills picked up from previous capstone courses. To get an account, the premade rootuser has to send you an invitation. This invitation comes in the form of an email, gives you a username and password, and directs you to sign in and update your password. This account creation process was based largely on the experience that Jamie has had with administrative tools at IU. We built an authentication system using PHP and MySQL. The system not only checks for a valid auth, but also looks specifically at the user who is logged in to determine what they can see. For example, only the root user can see the ‘Manage Admin Access’ link on the admin webpage.

For the first bit of the project, we used a static database that was stocked with our own fake sensor data. Closer to the end of the project, Bryce setup an MQTT server to host light pollution data. This data was in the exact same format we’d be using, but came at a much faster speed—producing several records of data per minute. Once the sensors are live, the database will only be updated with one record per day, which contains one light measurements per hour for the entire day. This was the part of the project we were most worried about completing because nobody on the team had experience with requesting data from a server, or running a file continuously on our own server.

We learned that an MQTT server functions differently than an HTTP request. Our system is never actually requesting data—it is subscribed to it. Whenever new data is published to the MQTT server, anyone who is subscribed receives the new data. However, this still required that our file which subscribes to the data and stores it in the database was always running. At this point, we weren’t quite sure how to accomplish that. Did we need to dedicate a machine to running the process? This is when we learned about the terminal command ‘screen.’ Screen essentially opens another terminal session. Once in screen, we started running our file which subscribes to sensor data and stores it in our database. This file is set up to run on a while one loop, so it would keep running. Next, we detached this new screen. That essentially closed the session from our local console, but it still running on the Silo server. Now we were getting new data automatically!

This was one of the more interesting parts of our project. We were able to learn something new, and also find out just how finicky the back end of website can be. On the morning of the capstone fair, we encountered what some might an ‘outage.’ We were getting no new sensor data. After talking with Bryce, we learned that was because Silo rebooted early that morning. This stopped our process for obtaining sensor data. We were able to get it back up and running no problem, but it was fun, in a way, to experience this very real-world problem.

Google Maps API:

One of the large features on the main page of the website is the map in the center of the page. The map was created using the Google Maps API, which is publicly available. Once the baseline API was integrated into the page, we first had to get our sensors to show on the map. This was accomplished by using PHP to create Javascript code that could be interpreted by the API. When it was first implemented, the PHP command that requested the sensors was hard-coded into the page, which meant that if a new sensor was added, then an individual had to go into the code for the site and change how many sensors were showing on the map. The way that we fixed this issue was to create another PHP script on the page that would search for the number of sensors that were in the database, and then create a marker on the map that was shown at the correct latitude and longitude, which were pulled from the database.

For each marker on the map, the page also created an info box that could display some data about the sensor. Initially these only told the user what sensor he marker corresponded to. Soon after we added a link that would send you to the graphs page for the sensor, where you could view specific daily, weekly, and monthly data. Another link that was later added was a link to google maps so a user could find the location where the sensor is, which could be beneficial for both people looking for a dark location, and admin who want to remove a faulty sensor. Finally, for each sensor the last night's hourly light data was added. Initially we wanted to show the past few hours of data here, but because the sensors send data once a day this would not be possible. One way that we will hope to improve this feature in the future is to make the night hours that are displayed correspond to the sunset and sunrise, as opposed to general hours that were hardcoded in.

Our team also wanted to be able for the user to easily see what recent data had been just by looking at the map. One way that we were able to do this was through custom markers. The markers used the data collected by the PHP request that showed the previous night's light data on the info box and found the maximum number that was in that list. Then, the page would check to see what that number was and find an image in a folder that we added that matched the light reading and would make that image the marker that was displayed on the map.

One visual that we were trying to add for a while was a heatmap feature. There is an imbedded heatmap feature in the Google Maps API, but unfortunately that looks at the number of points in a location and not the intensity of those points. The way that our team was able to work around this problem was by looking at the earthquake feature which the API has. We used the 'circle' feature to create a circle around each marker, and then used PHP to pull the last weeks' worth of night data from the database and average it. The average was used to determine the color of the circle around the point, which could be used to alert either users or park admin when there was a light source that was regularly on at night that was creating light pollution.

Overall, using the map was a great way to learn how to both implement an API into a project, and how to use many of the features that it has available to further customize it to make it better for the project that we were working on. It was also good practice to work with the Google Maps API, which is very commonly used on websites, and could be a very good skill to have in the future.

Graphs:

Firstly, we found a way to use JavaScript to display the graphs with a Google chart. We tested it with fake data and it works, so the rest of the task is to get the data from the database and present the graph in an ideal way.

Secondly, we tried to extract the data from the database. We tested the connection, and extracted the whole table at the beginning, and after we had succeeded, we started by converting the data into strings and put them into the correct format for JavaScript. Until then, we were still using the fake data we had created.

We created two separated pages, one that can show the graph of all of the sensors and one shows the detailed graph of a sensor that being selected from the main page. In order to get the sensor ID for the second page, we had to some extra work, because the information was sent as a hash value in the url code, and we could not ﬁnd a way to extract the hash value directly. We also thought about using a “form” to perform the action, but selecting sensor part is already in another form. As a result, we found a way to extract the hash value in the jumping page, and it will automatically jump to the graph page, along with the sensor ID stored in the “post” method.

When we were creating the all sensors page, we had to connect to the database and ﬁnd out how many sensors we have, so we used a SELECT sensorId FROM sensor\_data and used a distinct function do it. We just used a loop to count the number of the lines. After we got the number of the sensors, we used a for loop that will repeat the number of sensors time to print the repeated part of the HTML code. This is not hard, but complicated, because there are several parts need to be repeated, like the JS part, extract data (for each sensor), and also the div part the being used to present on the page.

Later in the project, we received the updated data from the client, so the ﬁrst thing we did was update the code so that it can adopt the updated data. The updated data is in a diﬀerent format, it collects the information by timestamp, and the data contains 24 decimal numbers. For the daily graph, we extracted the data by selecting the sensorID, and ordered by the timestamp to select the latest one. We extracted the 24 numbers and put them reversely since the graph is from left to right, and the unit we used for the x-axis is “hours ago”. So, for the x-axis, we used a for loop to put in 24 and reduced by 1 each we ran it.

We calculated the average for the latest timestamp from each day to represent the data for that day and used this information to present the weekly and monthly graph. We used hourly for the all sensor graph pages.

Database:

The main parts of the database are a sensors table, users table, and a sensor data table. We made a sensors table and inputted some fake sensors with the following fields: sensor id, serial number, latitude, longitude, location, and active. The location field contains descriptors of where the sensor is located in Hoosier National Forest. The active field is a yes/no field that specifies if the sensor is currently active/deployed. This table is used on the admin site where all the sensors are displayed and the admin can add or remove sensors. This will update the table immediately in the database. It is also used to populate the map. The map takes the longitude and latitude of a sensor point to make a marker. This also happens immediately when the sensor table is updated.

The users table has all the information of the admin users. It contains the following fields: user id, username, password, first name, last name, and email. The passwords in the table are hashed to protect the admin users' passwords. This table is only used on the admin site and it is used when a user is registering to make sure that no one with the same username or email is registering.

The sensor data table contains all the data for the sensors. It has the following fields: time stamp, sensor id, and readings. This table is populated with the data coming in from Bryce's fake data stream. The table is used in the creation of graphs and the overlay on the map. It is also what is downloaded as a CSV when the user clicks the link on the homepage.

In terms of goals we did not accomplish, there was one thing that did not get done. We wanted an admin to be able to flash a sensor. The admin would click a button on the site and the sensor would flash a light so that it might be easier to find in the woods. However, the graduate student team working on the sensors did not complete them in time for us to finish this functionality. It was impossible because we did not have the sensors. If our team had to go back and do things differently, we would have started earlier and started communicating better. We would not have saved the hardest things for last because it made the last few weeks a scramble. Overall, we are very proud of this project and how everything turned out.