Theodore “TJ” Norred
David McGee
Project Title: Model Managers
Project URL: http://classwork.engr.oregonstate.edu:3797/

Project Step 6, Model-Managers Project Proposal:

1. **Executive Summary:**

We made quite a few changes to the application and codebase thanks to the feedback received from TAs and other students. We also identified and fixed several bugs over the course of the project. These included adding a better connector to the database to handle network communication issues where the project database server was not immediately responsive. The database communication issues were a major negative application feedback item, and were also helped by the course admins adding RAM to the database server. This greatly reduced the timeouts and the negative feedback from students who were accessing our site only to see it crash. We also received a lot of feedback about the code formatting being hard to read. To fix this, we applied the black auto-formatter to all python code to improve the code formatting.

One of the big realizations we had during this project was that this is not an application focused website - it is designed for an administrator of the database and application. This realization changed how we set up our site pages. We added an individual HTML web page for each entity, instead of combining all the entities into a single library page as we originally intended. This change would make it easier for a database administrator as the database and entities grow over time.

We made several changes to the database and the database data as our understanding of database normalization and usage improved. We changed the models table structure to remove the auto-increment feature and allow adding / removing / updating based on a predefined hash table. We updated the DDL to more closely mirror the APIs used to generate forecasts and readings data. We modified the DLL to better mirror the open-meteo API (removed forecastWindGust) and updated the Schema to correspond to the attribute removal.

As our understanding of the project evolved based on the continued updates from the project step assignments, we modified the DDL to make the `forecastModelID` foreign key nullable. This enabled us to make models a dropdown selector, and we added error handling as well to make updates and additions more robust, since there are actually only a limited number of weather forecast models that exist.

We tried to improve the User Interface to make it more appealing by adding CSS buttons for edit and delete functionality, as well as pulling CSS formatting via the pico-css library. We also added confirmation popups for all deletion events to make sure that users intended to delete the identified object. We connected the create forecasts feature to the OpenMeteo API and the create readings feature to the Holfuy API. This enabled us to add data easily to the database.

Finally, even though the purpose of this course was database interactions, we tried to do some initial graphical plotting of the database data. We created the graphs on the results page using Matplotlib / pyplot to present the data in a different and visual way. These graphs certainly are not perfect, but they provide a more intuitive interface for analyzing the database data than the table structure.

2. Project Outline:

The Model-Managers company enables customers to determine which wind forecast models are most accurate for given locations over specified time intervals. There are a variety of wind forecast models (HRRR, ECMWF, GFS, MBLUE, etc.) which often have different wind forecasts for a given location. Users of these forecast models usually cannot determine what model is the most accurate for their specific location. In other words, they would need to compare the model forecasts to actual data from weather/wind collection devices at specific locations to determine which model is typically more accurate for those locations.

The Model-Managers company solves this problem by storing hourly wind forecasts for a standardized forecast period of 7 days in a database, and then allowing users to interact with a web interface to compare the forecast data against the actual measured data at a location for different time intervals. The 3 criteria used to compare forecast predictions to measured wind are average wind speed, wind direction, and wind gust speed. The Model-Managers company uses the free OpenMeteo API (https://open-meteo.com) to obtain wind forecast data, and a Holfuy (https://api.holfuy.com/) device API to obtain the measured wind forecast data.

The minimal viable product of the Model-Managers company uses 2 weather forecast models (HRRR and ECMWF) and 1 Holfuy device, located at La Bajada Ridge, New Mexico. The Model-Managers interface is expected to be accessible by a minimum of 50 different users over 100 times per day. Model-Managers uses a GitHub repository at https://github.com/skipmcgee/wind-forecast to manage their codebase.

The entities that the Model-Manager's Database uses are Models, Locations, Sensors, Forecasts, Dates, and Readings. There will be more instances of Dates, Readings and Forecasts than any other entity, as we are limiting the MVP to one Sensor and 2 Models. Since there is one Sensor, that Sensor will be at 1 location for the MVP phase of the project. The Dates of the Readings and Forecasts will be taken hourly, so 24 times a day for approximately the next four months (each). This is anticipated to be around 2100 instances of each of the Dates, Forecasts and Readings entities.

3. **Database Outline:**

☐ Models:

These are the individual weather models that the Model-Managers product supports. These weather models are the model that drives a specific forecast and include HRRR, ECMWF, etc models. This entity's purpose is to capture the weather model specifics.

a. modelID: int, unique, not NULL, PK
b. modelName: varchar(100), not NULL

Relationships: there is a one-to-many relationship between models and forecasts. For every model, there are many forecasts. Forecasts are the intersection table(s) of the M:N or many-to-many relationship between Models and Locations.

☐ Locations:

The locations where customers desire to compare forecasts to measured values. This entity's purpose is to manage the location information such as latitude, longitude, and altitude.

a. locationID: int, auto_increment, unique, not NULL, PK
b. locationName: varchar(300), unique, not NULL
c. locationLatitude: float, not NULL
d. locationLongitude: float, not NULL
e. locationAltitude: int, not NULL

Relationships: A one-to-one relationship exists between locations and sensors. There is only one sensor at a specific location. A one-to-many relationship exists between locations and forecasts. There are typically greater than one forecast at a single location. Forecasts are the intersection table(s) of the M:N or many-to-many relationship between Locations and Models.

☐ Sensors:

The sensors that can measure the weather data at a specific location. These are specifically Holfuy sensors for the MVP, but the attributes are intentionally generalized enough that these could be expanded to include other makes and models of sensor. The purpose of this entity is to manage the attributes of the specific sensors, This includes specific attributes of a sensor, such as its manufacturing number and the api key used to access it, as well as references to its location.

a. sensorID: int, auto_increment, unique, not NULL, PK
b. sensorName: varchar(300), not NULL
c. sensorLocationID: int, not NULL, FK, -> locationID
d. sensorAPIKey: varchar(300), not NULL
e. sensorNumber: int, not NULL

Relationships: there is a one-to-one relationship between sensors and locations.

☐ Forecasts:

Intersection Table for hourly forecast information that is common across all weather models. The purpose of this entity is to hold the forecast prediction information for the weather at a specific date and time.

a. forecastID: int, auto_increment, unique, not NULL,
b. forecastMadeDateID: int, not NULL, FK -> dateID
c. forecastForDateTime: DATETIME, not NULL
d. forecastTemperature2m: float, not NULL
e. forecastPrecipitation: float, not NULL
f. forecastWeatherCode: varchar(100), not NULL
g. forecastPressureMSL: int, not NULL
h. forecastWindSpeed10m: float, not NULL
i. forecastWindDirection10m: float, not NULL
j. forecastCape: float, not NULL
k. forecastLocationID: int, not NULL, FK -> locationID
l. forecastModelID: int, FK -> modelID

Relationships: Forecasts in an intersection table. There is a many-to-one relationship between forecasts and locations. There are many forecasts at a single location. There is a many-to-one relationship between forecasts and models. There are many forecasts for a single model. There is a many-to-one relationship for forecasts to dates. There are many forecasts at a date.

☐ Readings:

The actual measured sensor readings at a specific location. The purpose of this entity is to hold the reading information that a sensor provides at a specific date and time.

a. readingID: int, auto_increment, unique, not NULL, PK
b. readingSensorID: int, not NULL, FK -> sensorID
c. readingDateTime: int, not NULL, FK -> dateID
d. readingWindSpeed: float, not NULL
e. readingWindGust: float, not NULL
f. readingWindMin: float, not NULL
g. readingDirection: int, not NULL
h. readingTemperature: float, not NULL

Relationships: there is a many-to-one relationship between readings and sensors. There are many readings at a single sensor. There is a many-to-one relationship between readings and dates. There are many readings at a single date.

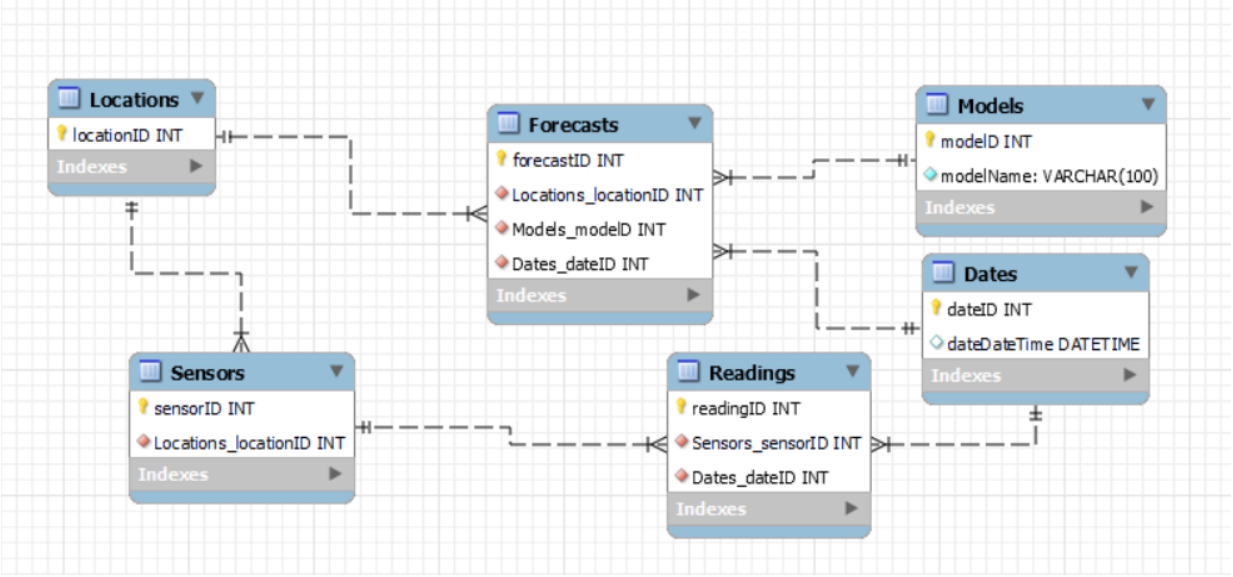
☐ Dates:

The date and time. The purpose of this entity is to hold the date information and to facilitate access to a specific forecast and/or a specific reading at a date and time.

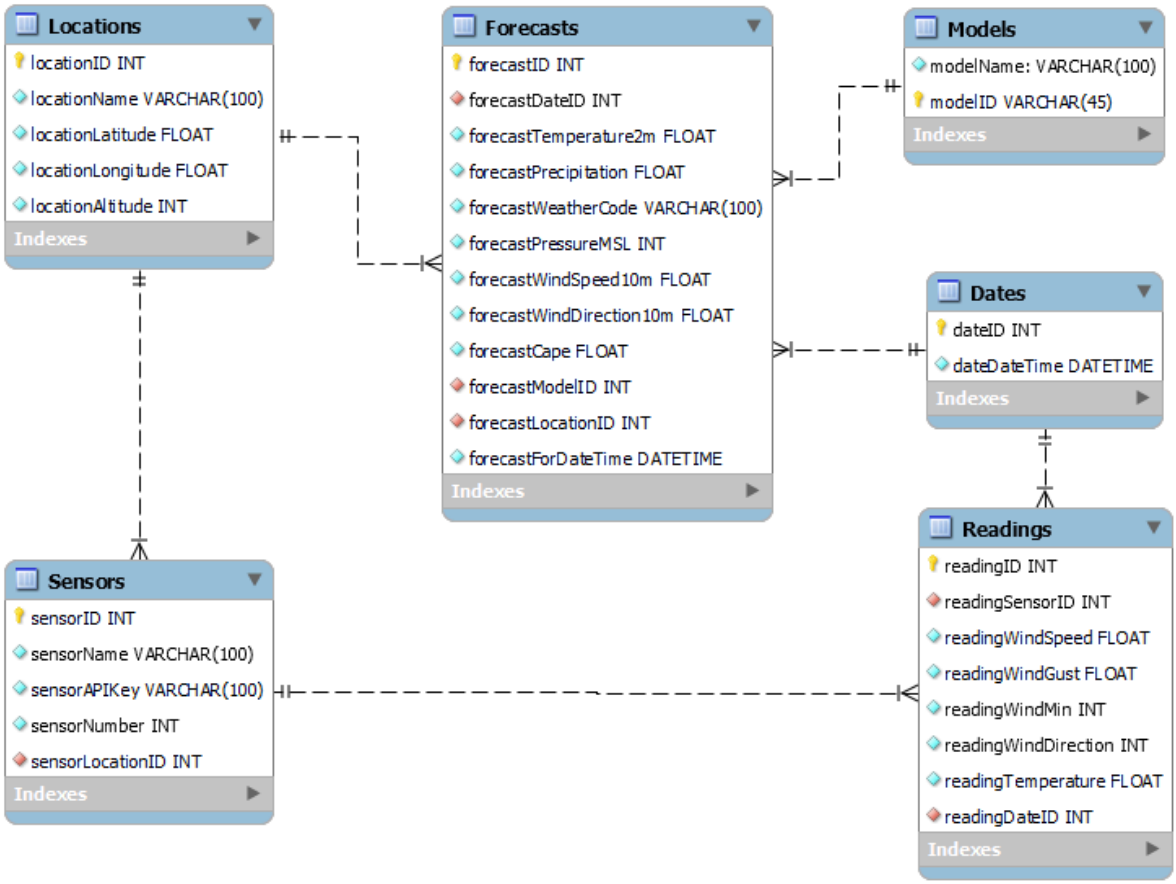
a. dateID: int, auto_increment, unique, not NULL, PK
b. dateDateTime: DateTime, not NULL,

Relationships: there is a many-to-one relationship between readings and dates. There are many readings at a single datetime. There is a many-to-one relationship between forecasts and dates. There are many forecasts for a single datetime.

4. Entity Relationship Diagram:



5. Schema:



6. Sample Data:

Models Data:
(1, 'ECMWF'),
(2, 'HRRR'),
(3, 'GFS');

Locations Data:
('La Bajada Ridge Launch', 35.56195, -106.22596, 6135),
('Sandia Peak Launch', 35.196576, -106.434662, 10275),
('Sandia Crest Launch', 35.21342, -106.45026, 10600),
('Blue Springs Launch', 34.44002, -106.51913, 6200);

Sensors Data:
('La Bajada Holfuy', 1, 'mytestapikey1234', 1151),
('Sandia Peak Holfuy', 2, 'mytestapikey23456', 1152),
('Sandia Peak Tempest', 2, 'mytestapikey345670', 2),
('Sandia Crest Holfuy', 3, 'mytestapikey456789', 1153),
('Blue Springs Holfuy', 4, 'mytestapikey56789', 1154);

Dates Data:
('2024-04-01 08:00:00'),
('2024-04-01 09:00:00'),
('2024-04-01 10:00:00'),
('2024-04-01 11:00:00'),
('2024-04-01 12:00:00'),
('2024-04-01 13:00:00'),
('2024-04-01 14:00:00'),
('2024-04-01 15:00:00'),
('2024-04-01 16:00:00'),
('2024-04-01 17:00:00'),
('2024-04-01 18:00:00'),
('2024-04-01 19:00:00'),
('2024-04-01 20:00:00'),
('2024-04-01 21:00:00');

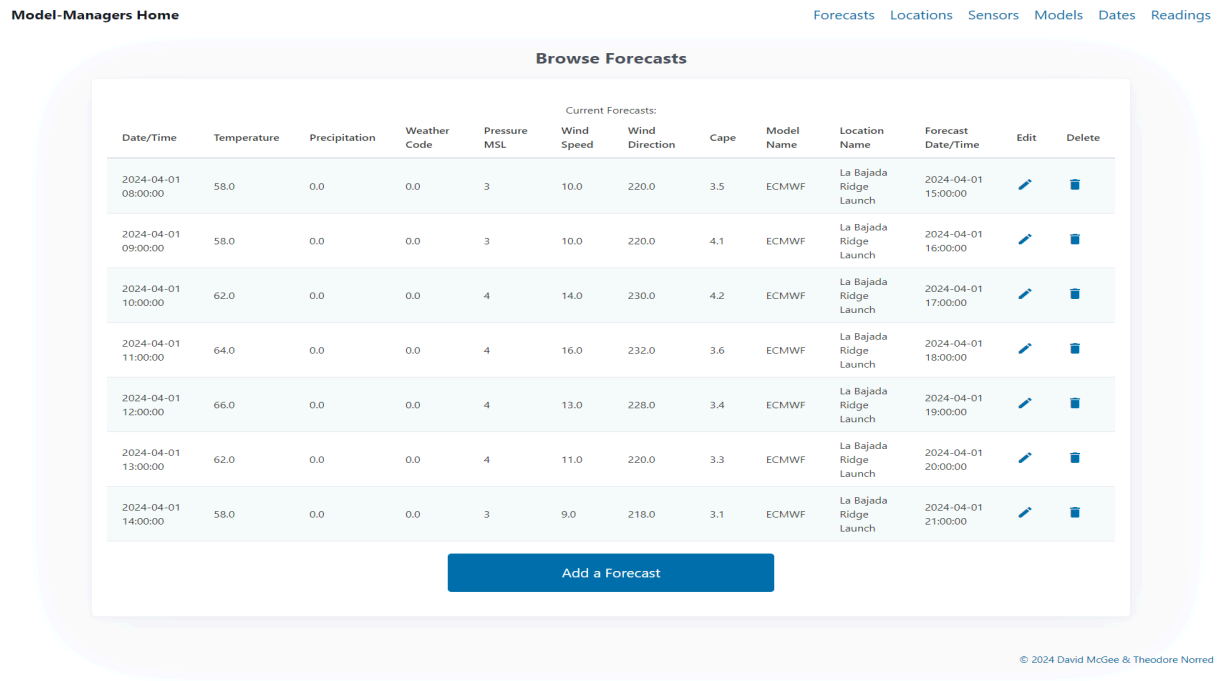
Readings Data:
(1, 1, 22.0, 28.0, 14.0, 250, 64),
(1, 1, 22.0, 28.0, 14.0, 250, 64),
(1, 2, 20.0, 22.0, 6.0, 231, 66),
(1, 3, 21.0, 24.0, 11.0, 242, 67),
(1, 4, 23.0, 25.0, 8.0, 250, 69),
(1, 5, 18.0, 21.0, 4.0, 256, 71),
(1, 6, 17.0, 22.0, 12.0, 264, 74),

(1, 7, 16.0, 20.0, 15.0, 270, 72),
(1, 8, 18.0, 21.0, 18.0, 265, 77),
(1, 9, 20.0, 24.0, 16.0, 254, 75),
(1, 10, 19.0, 22.0, 17.0, 246, 74);

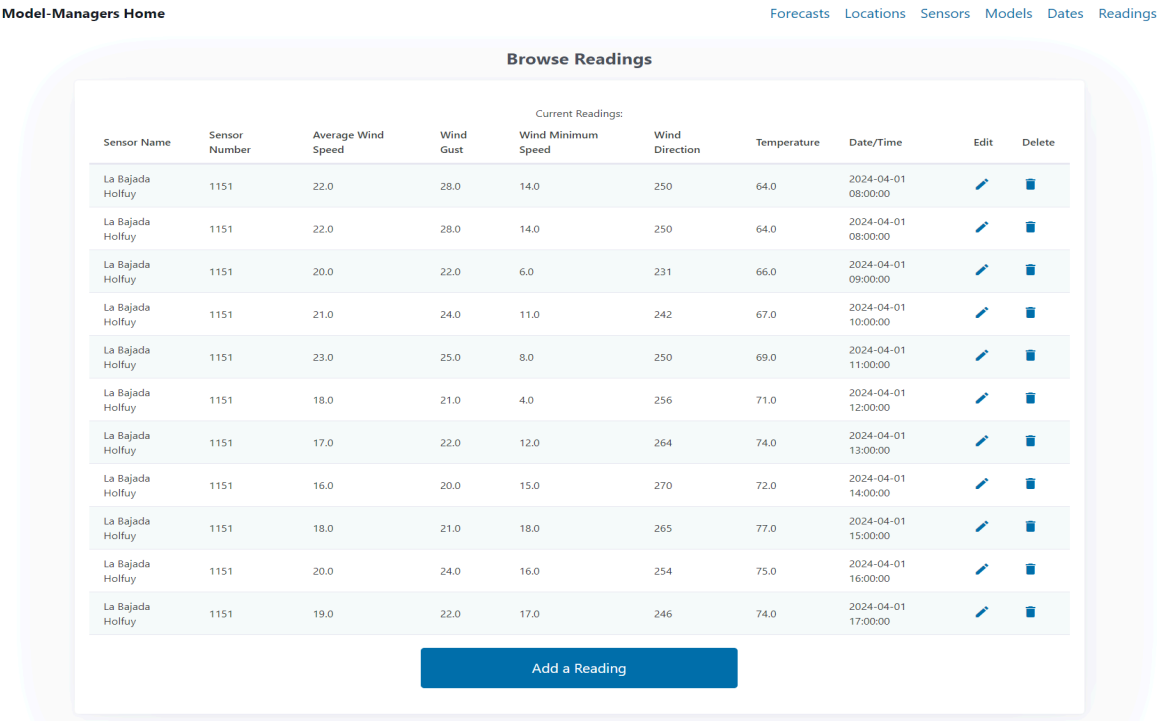
Forecasts Data:
('2024-04-01 15:00:00', 1, 58, 0, 'CLEAR', 3, 10.0, 14.7, 220.0, 3.5, 1, 1),
('2024-04-01 16:00:00', 2, 58, 0, 'CLEAR', 3, 10.0, 14.7, 220.0, 4.1, 1, 1),
('2024-04-01 17:00:00', 3, 62, 0, 'CLEAR', 4, 14.0, 24.0, 230.0, 4.2, 1, 1),
('2024-04-01 18:00:00', 4, 64, 0, 'CLEAR', 4, 16.0, 25.2, 232.0, 3.6, 1, 1),
('2024-04-01 19:00:00', 5, 66, 0, 'CLEAR', 4, 13.0, 25.0, 228.0, 3.4, 1, 1),
('2024-04-01 20:00:00', 6, 62, 0, 'CLEAR', 4, 11.0, 24.0, 220.0, 3.3, 1, 1),
('2024-04-01 21:00:00', 7, 58, 0, 'CLEAR', 3, 9.0, 20.0, 218.0, 3.1, 1, 1);

7. Screen Captures:

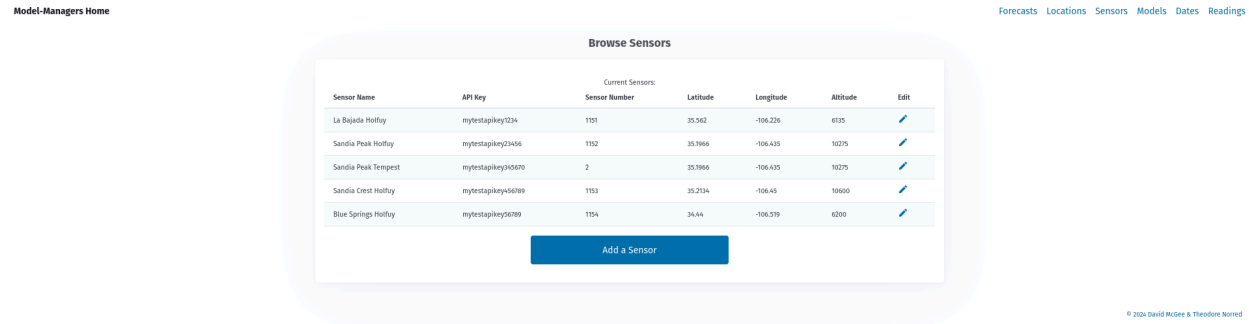
READ Forecasts



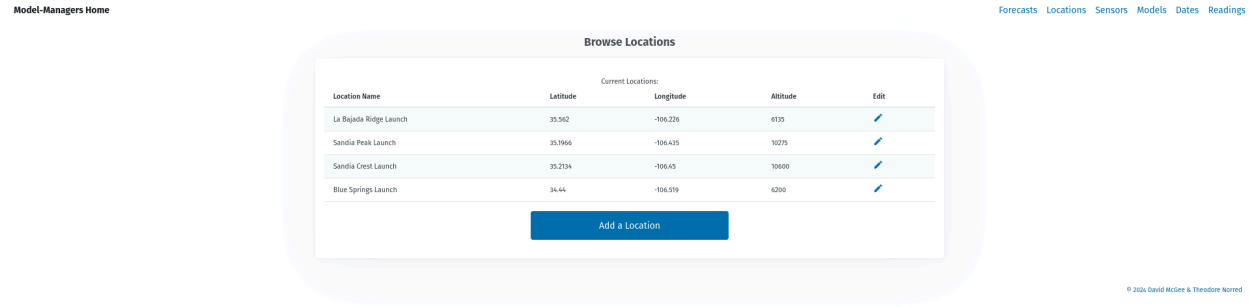
READ Readings



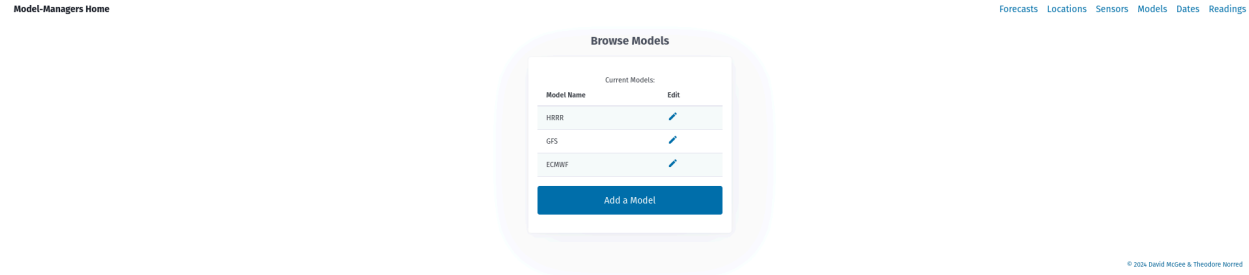
READ Sensors



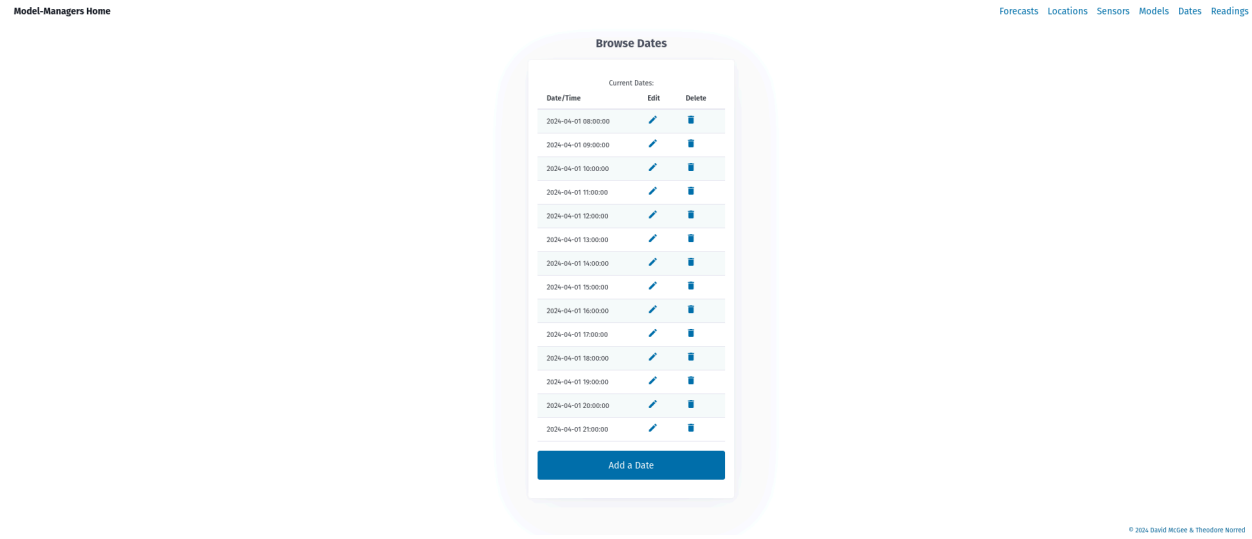
READ Locations



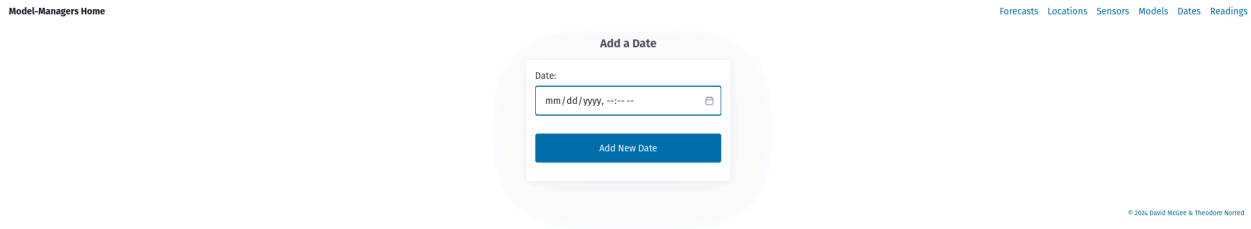
READ Models



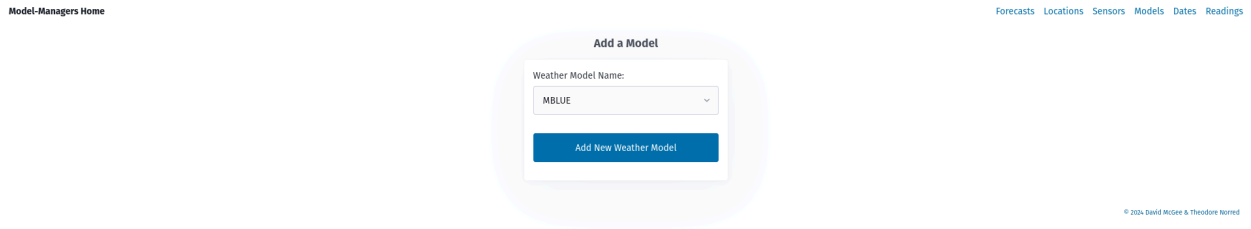
READ Dates



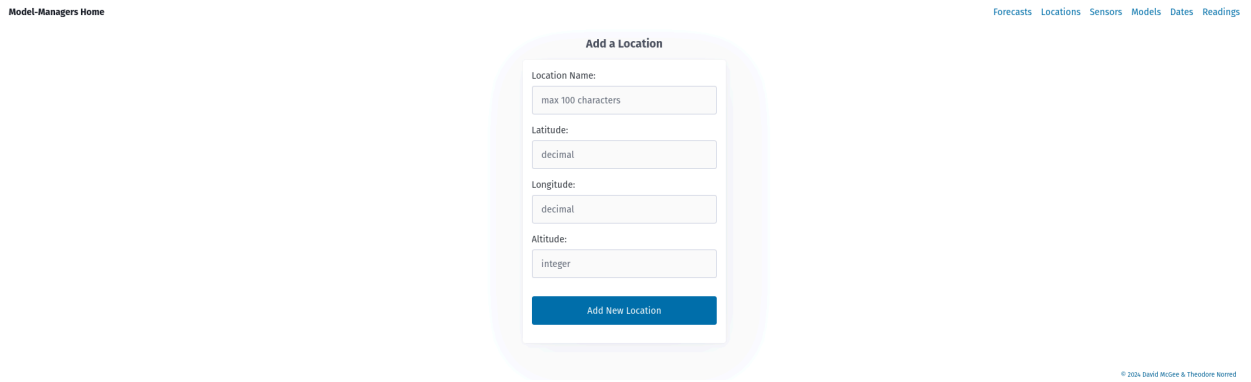
CREATE Dates



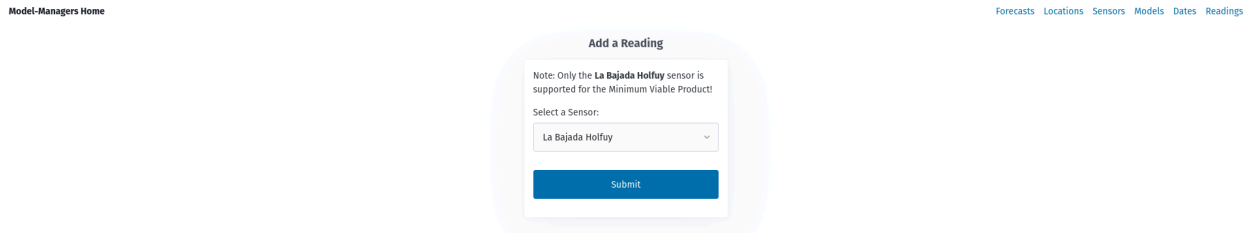
CREATE Models



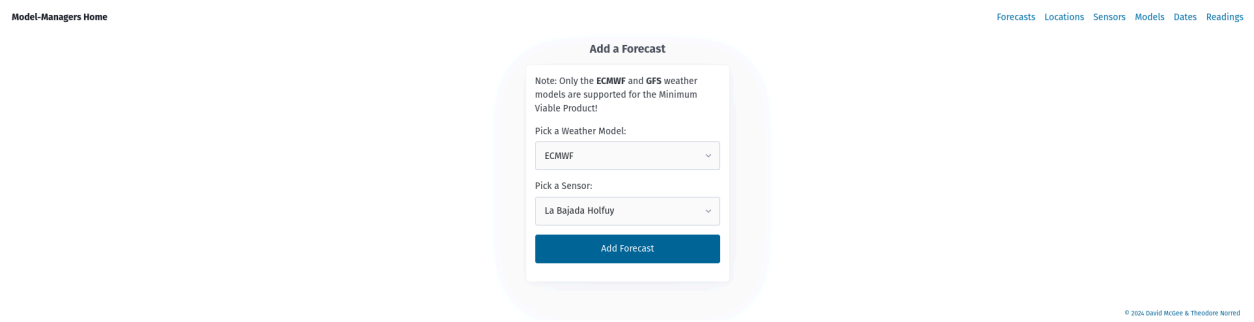
CREATE Locations



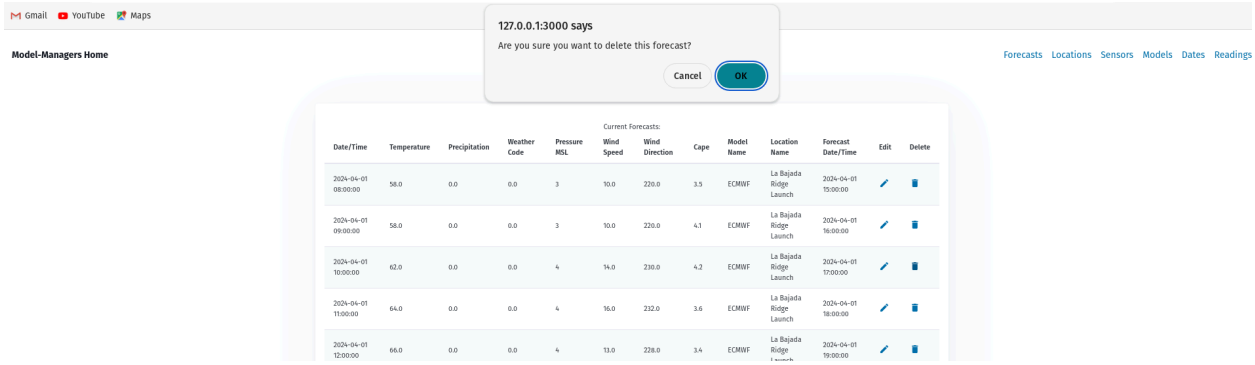
CREATE Readings



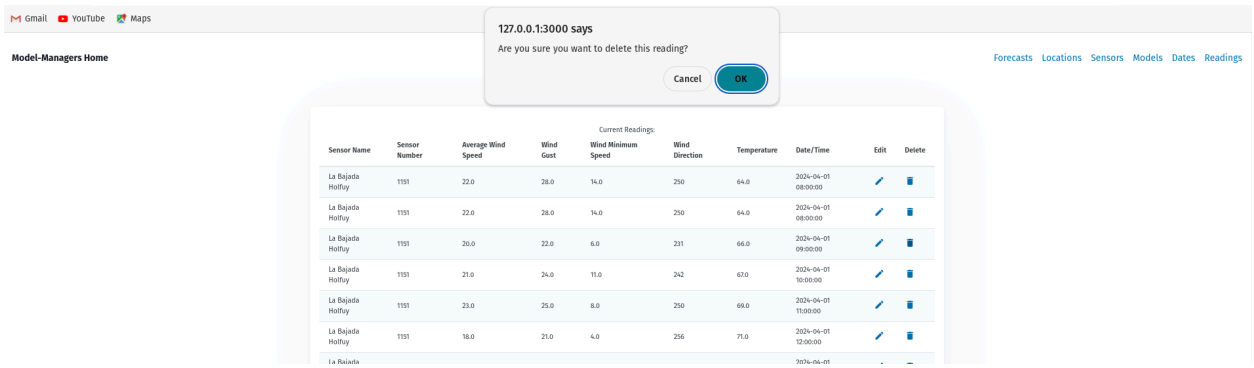
CREATE Forecasts



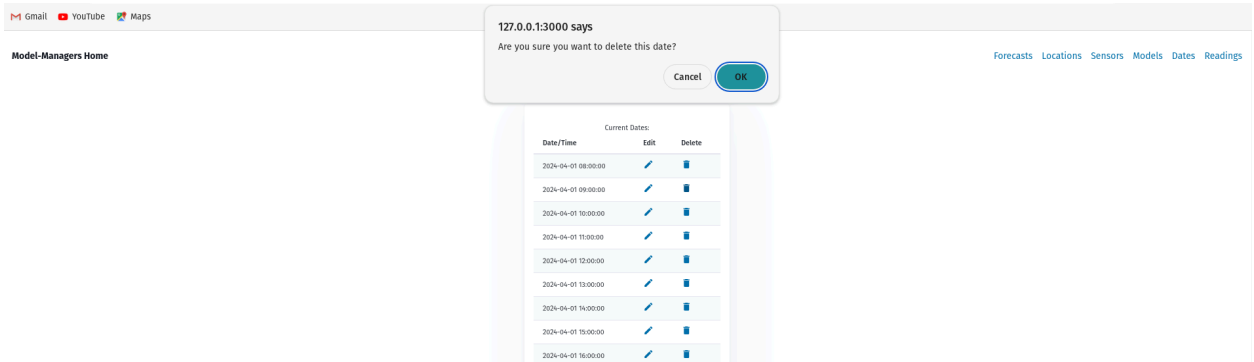
DELETE Forecasts (M:N)



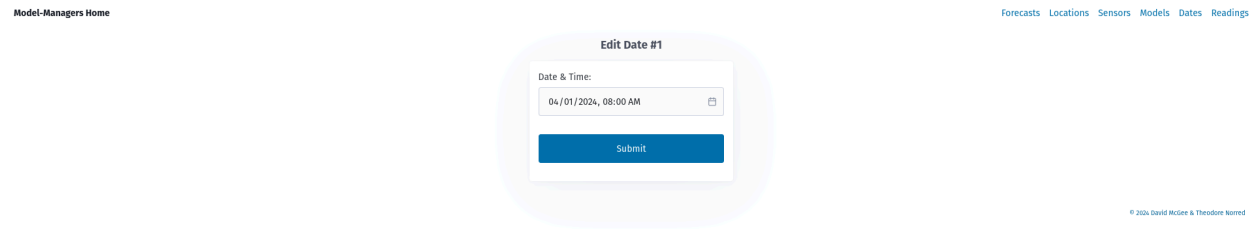
DELETE Readings (M:N)



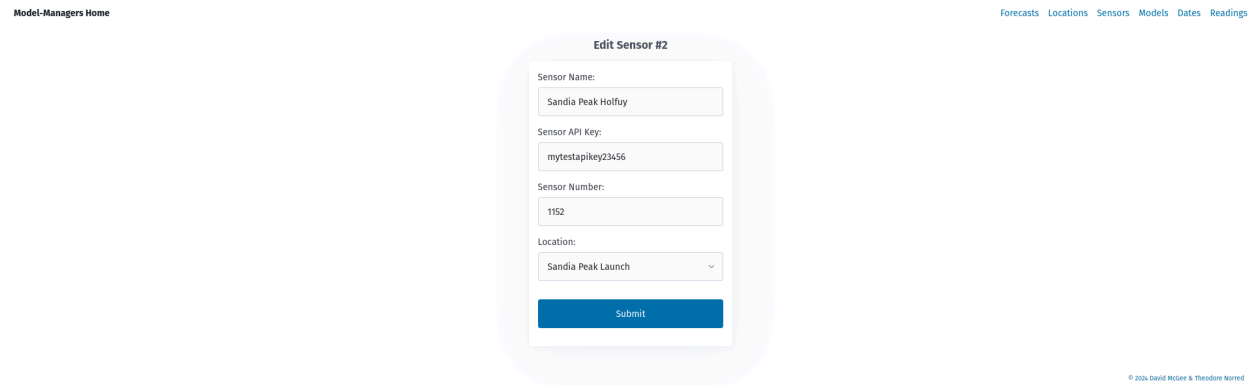
DELETE Dates



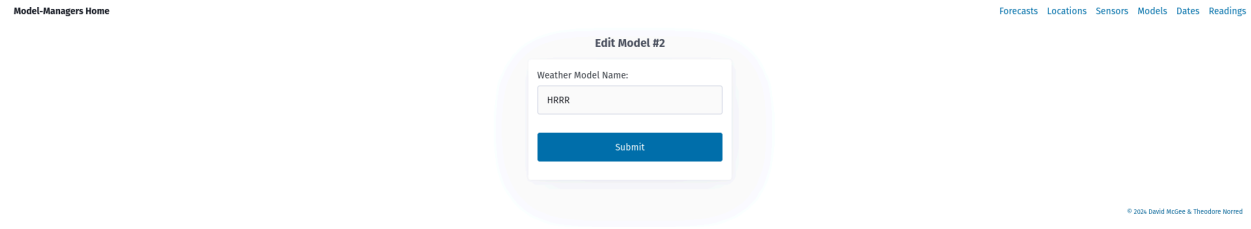
UPDATE Dates



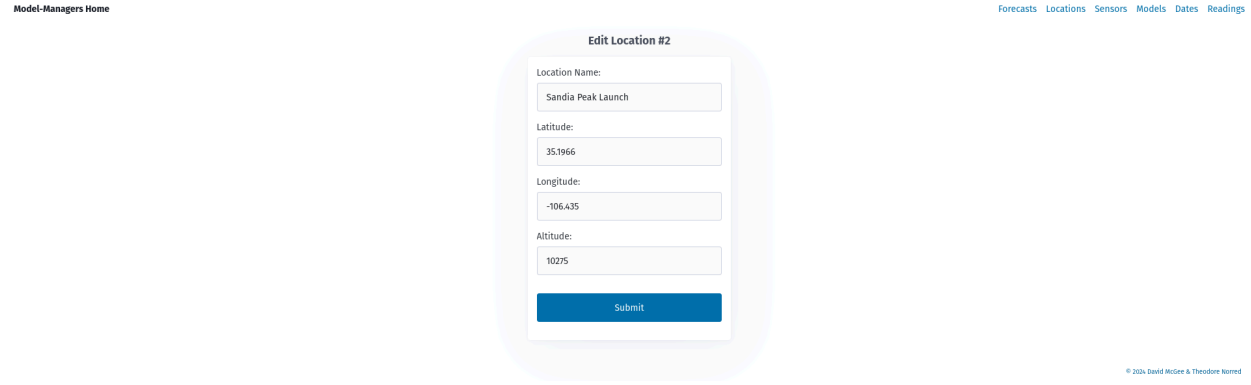
UPDATE Sensors (Dropdown with Location)



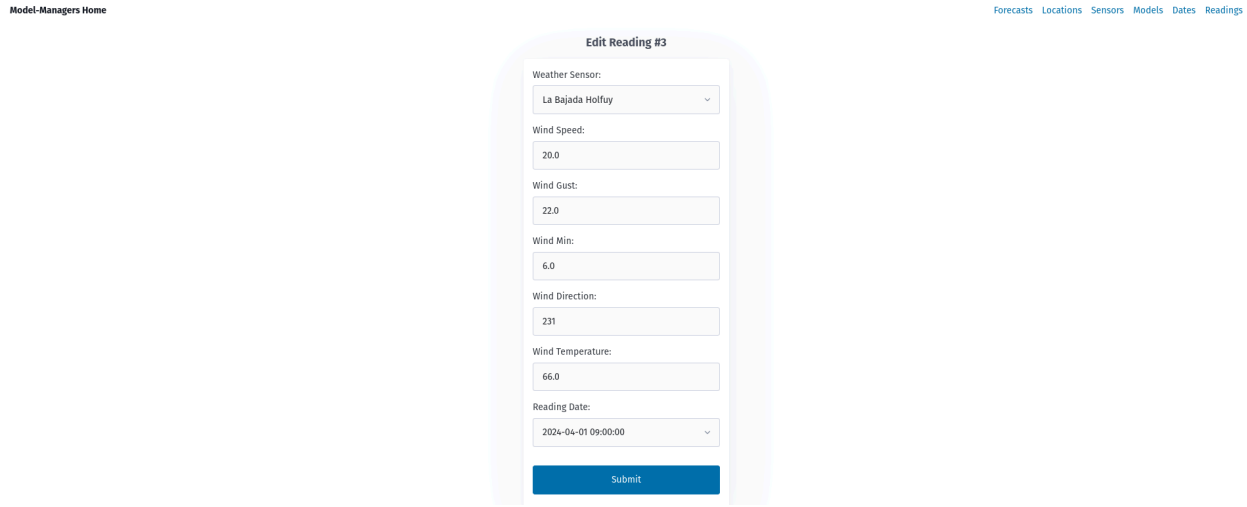
UPDATE Models (Limited to only actual weather model names)



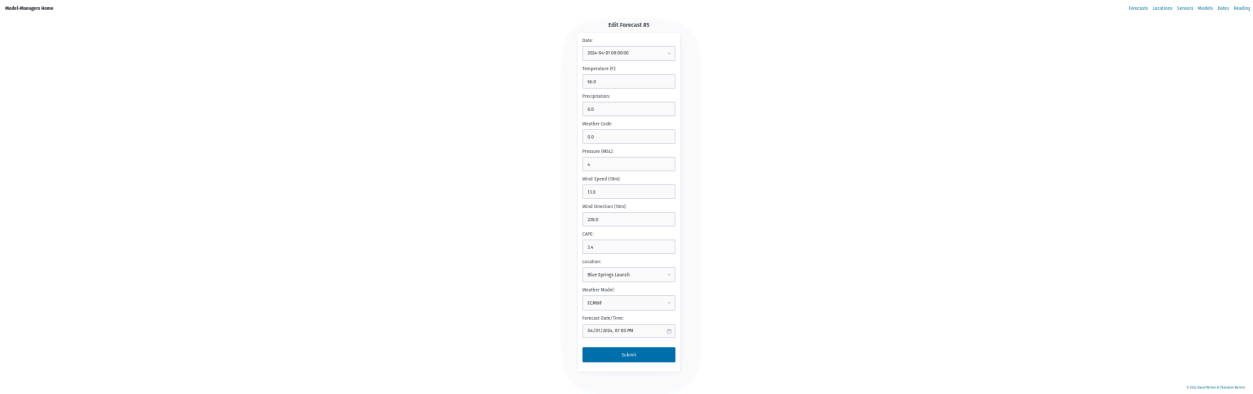
UPDATE Locations



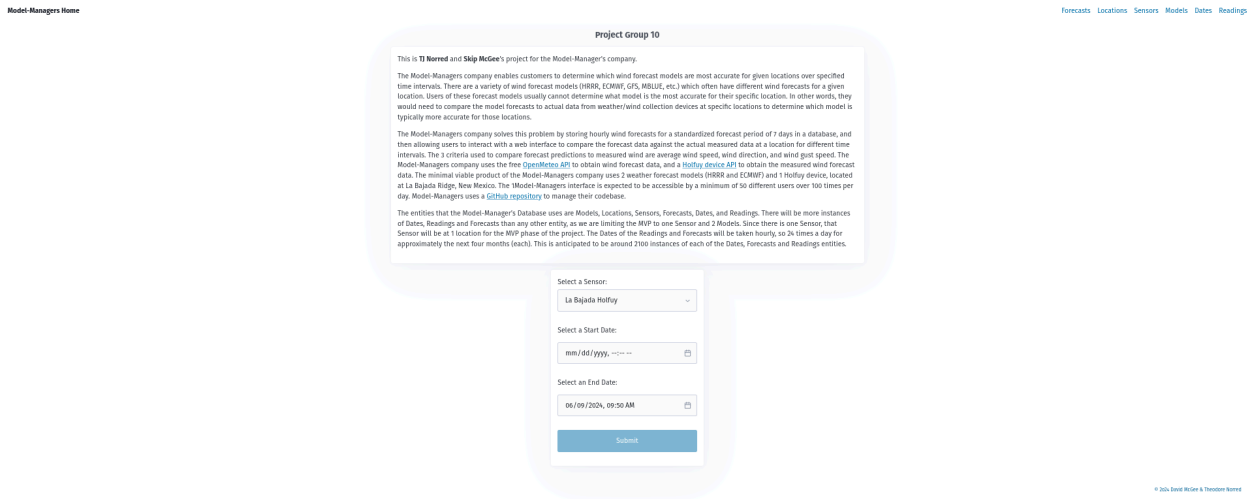
UPDATE Readings (M:N, Dropdown for Sensor and Dates)



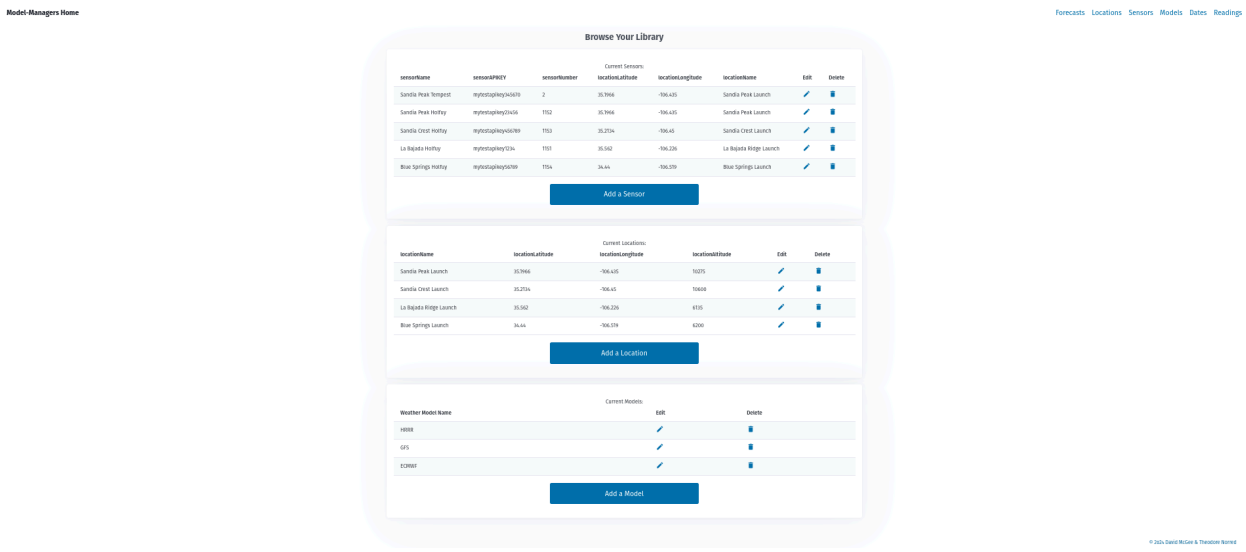
UPDATE Forecasts (M:N, Nullable Model, dropdown Location and Dates)



EXTRA / Additional Pages and Functionality:
HOME with data selection inputs (populated by READ functionality)



LIBRARY (allows READ and CREATE for Sensors, Locations, Models in one easy spot)



RESULTS (comparison of Readings and Forecasts in graphical and table form based on a specified datetime range)