TEAM SKYWARD

Frost Monitor Project

Software Design Document

Version 1.1

2/14/2017

Team Name:

Skyward

Team Members:

Gage Cottrell
Justin Kincaid
Chris French
Alexander Sears

Sponsors:

Dr. Michael Mommert and Dr. David Trilling

Mentor:

Dr. Otte

Table of Contents

Introduction 2

Implementation Overview 4

Architectural Overview 5

Module and Interface Description 10

Implementation Plan 15

Conclusion 18

References 19

Introduction

Document Purpose

The purpose of this document is to outline the technical aspects of the FRoST Monitor System and the technologies used to develop and implement the application. The goal of this document is to give the reader a better understanding of how the application is being developed and implemented through examples of requirements, constraints, and system architecture.

FRoST Monitor Project

We are Team Skyward and we are building an online dashboard for the FRoST Monitor. The FRoST monitor is a telescope dome that is a few miles south of the Northern Arizona campus. The FRoST monitor dashboard will display vital telescope and weather information quickly and easily. Weather information gathered from the FRoST monitor and displayed on the dashboard will include data such as current temperature, dew point temperature, wind, and the latest image from the all-sky-cam. All of this data is supplementary to help our clients use the FRoST monitor telescope to photograph and study Near Earth Objects.

Near Earth Objects (NEOs)

Everyday about one hundred tons of particles originating from outside of Earth's atmosphere makes contact with Earth's surface. While the vast majority of the particles contacting Earth are very small, instances of larger objects impacting the surface have been recorded. Roughly every 10,000 years, a rocky or iron asteroid larger than 100 meters impacts the earth and causes localized destruction. On an average of every few 100,000 years, asteroids larger than 1 kilometer impact the earth and cause global disasters that can severely impact life on Earth. Due to the risk of having Earth impacted by objects that have the potential to cause massive destruction, it is vital that near earth objects be discovered, tracked, and analyzed.

Researching Near Earth Objects (NEOs) is an ongoing joint effort between many countries and governments. Using telescopes, it is possible to discover and track NEOs as they move throughout the solar system. Making publically available the information gained from NEO discoveries, allows for a collaborative environment in which NEOs can be discovered, tracked, and analyzed very quickly. To date, there have been more than 15,000 different NEOs discovered. As telescope technology advances, so do the amount of NEOs discovered. Figure 1 below depicts the discovery rate of near-earth-asteroids from 1980 to present date.

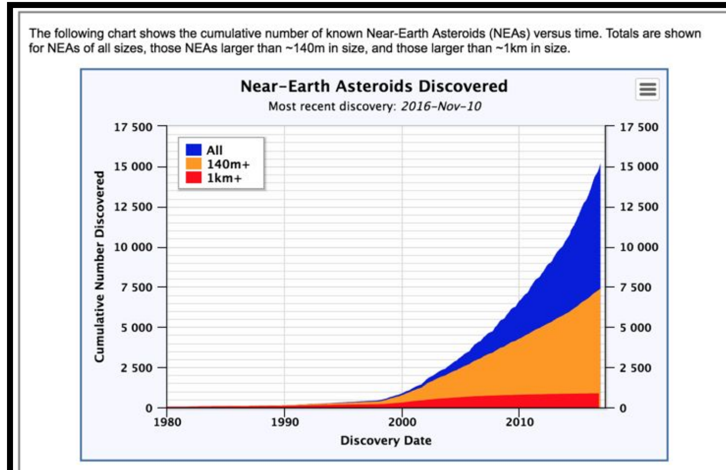

Figure 1: Near-Earth Asteroids Discovered

Nuthatch Server

"Nuthatch" is a server built and maintained by our clients. Our team will not be responsible in any way for the development or maintenance of the Nuthatch server. This server will gather and host weather and telescope information. Our team will then take this information and use it to populate all of the required data to display on our web application.

Project Requirements

The first requirement is a web application that will host display all of the weather and status information that is gathered from the telescope. The second is including a password protected user account for functionality and information that will be hosted behind a layer of security. All other requirements involve communication between the FRoST monitor web application and our clients server Nuthatch in order to get all of the necessary data to display on the web application.

Brief project requirements list includes:

- Web Application
  - Access to our clients Nuthatch server
  - Database to store mirror of data hosted on Nuthatch server
  - Ability to update data to web application in real time
  - Maintain a list of issued commands to the telescope
- Single Password Protected User
  - Dome shutdown button

Implementation Overview

The FRoST Monitor system will be built to give researchers and users of FRoST the ease of being able to quickly and efficiently check on the live status of the telescope and surrounding weather conditions. The FRoST Monitor System will be built using the Django web development framework - and will be using the Django REST Framework as a means to receive current data. The entire FRoST Monitor System will be created within a virtual environment using Anaconda, to ensure that our application can easily be modified without worrying about other outside factors.

Using Anaconda, we will install a virtual environment on our clients server so that we may make any changes we need in order for Django to run the way we want it, just so these changes aren't made for the whole machine. We aren't the only ones using it since it belongs to the whole NAU physics department, so it's best for us to use a virtual environment in order to keep our footprint on the system to a minimum. The machine will have to be configured to direct certain traffic to this environment in order to provide access to the FRoST Monitor system.

The Django REST Framework will be used to communicate with the client's own data processing machine. That machine will have a script that sends data to the web server which, since the data will be available locally, will make it quick and easy to display data on the web page.

The Django web development framework utilizes a Model View Controller (MVC) architecture that is essentially a 3-tier architecture. This document will later discuss in detail this architecture and its strengths and weaknesses.

The webpage that is produced with the Django framework will update the information being displayed automatically every 60 seconds using asynchronous JavaScript (AJAX) calls. Implementing AJAX into our Django application allows the application to automatically update the data being displayed, which is extremely convenient for users who need to view the latest data available.

The following section of this document will discuss in detail the architecture behind the FRoST Monitor System.

Architectural Overview

Application overview

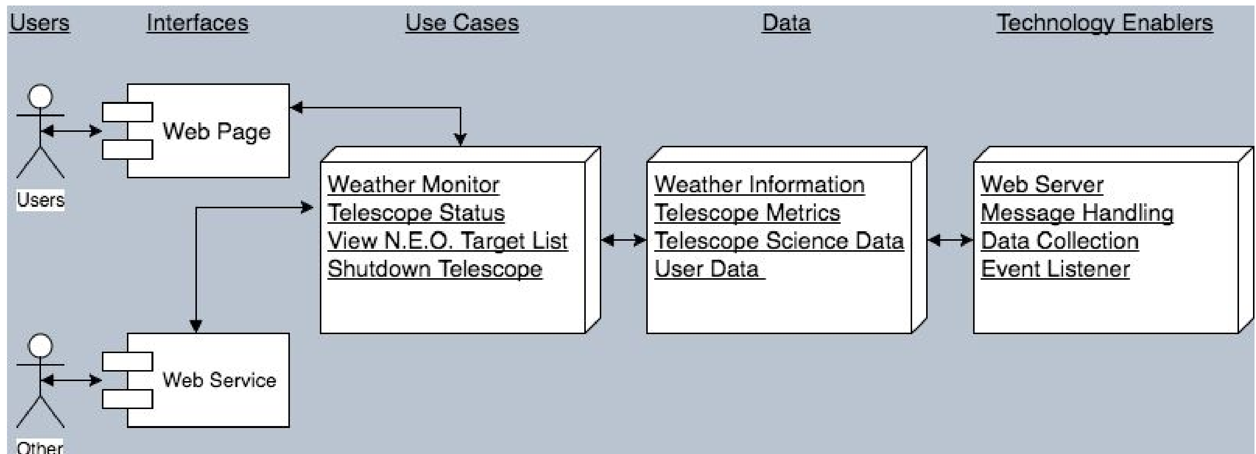

Figure 1: Application Overview and Diagram

User and delivery

The view in Figure 1 allows for different audiences to access the frost monitor through different channels. These two channels are:

- Researches and the public users will access application over the internet using a web browser. These are the main users of the FRoST Monitor System.
- Other users such as the web service requester will access the application through a web service implemented in a ReSTful service. Essentially these "other users" are applications/systems that access our application via the REST service to get/post data such as Nuthatch.

Core application uses

Figure 1 highlights the use-cases for the Frost Monitor application. These uses are the expectations of information to be presented to the user.

- Monitor Weather
    - The user will be able to monitor current weather conditions around and surrounding the FRoST telescope.
- Retrieve Telescope Status

    - The user will be able to quickly observe the current telescope status. An example is where the telescope is currently pointing in the sky.
- View N.E.O list and information
    - The user will be able to retrieve information on the current and upcoming Near Earth Objects that are to be observed by the telescope.
- Emergency Shutdown
    - An admin user will be able to login and send a signal to the telescope. This signal will trigger a mechanical response to the telescope to shut the dome.

Data

The core information and data constructs required to realize the core application uses are highlighted in Figure 1. The follow data is fundamental to the core use cases.

- Weather
    - The current weather data will be the current weather conditions of surrounding the FRoST telescope. This system will manage the propagation of weather data from web services and local hardware.
- Telescope Metrics
    - Telescope Metrics represent the physical status of the FRoST telescope. Examples are the current telescope position, and dome status.
- Telescope Observation Data
    - The telescope observational data represents the past and current observed data from the FRoST telescope. Examples of the data will include a list of N.E.O that are queued to be observed and pictures and telemetry data of past observations.
- User
    - The user data will allow for permission based action between the web application as well as the web service. There will be a user to allow for web service data push's. There will also be a user for access to the emergency shutdown.

Technology Enablers

Figure 1 highlights the key set of components to support implementation of the Frost Monitor application.

- Web Server
    - The application will reside on a web server. The web server will be required for providing access to the FRoST monitor application interface and the REST api.
- Message Handling

    - Will manage request over http from the event listeners. The handler will provide the implementation of client request for information and web service request.
- Data Collection
    - The data collection will be implemented on a gateway machine that will propagate weather and telescope data from the telescope and other external sources. This module will also be responsible for sending data to the web database through the REST api.
- Event Listeners
    - This module is the main function of the REST api. It will handle incoming events and request made over http from clients and web service users. It will then hand off the messages to the message handler.

The FRoST Monitor application is built predominantly upon the Django framework. The Django framework uses a 3-tier architecture very similar to the popular Model, View, Controller (MVC) architecture. Figure 2 can be seen below and is a diagram of the implemented framework in respect to the web application. In Figure 2 below you can see that a Django project can have multiple applications within itself, and that the FRoST Monitor System is using two applications in tandem.

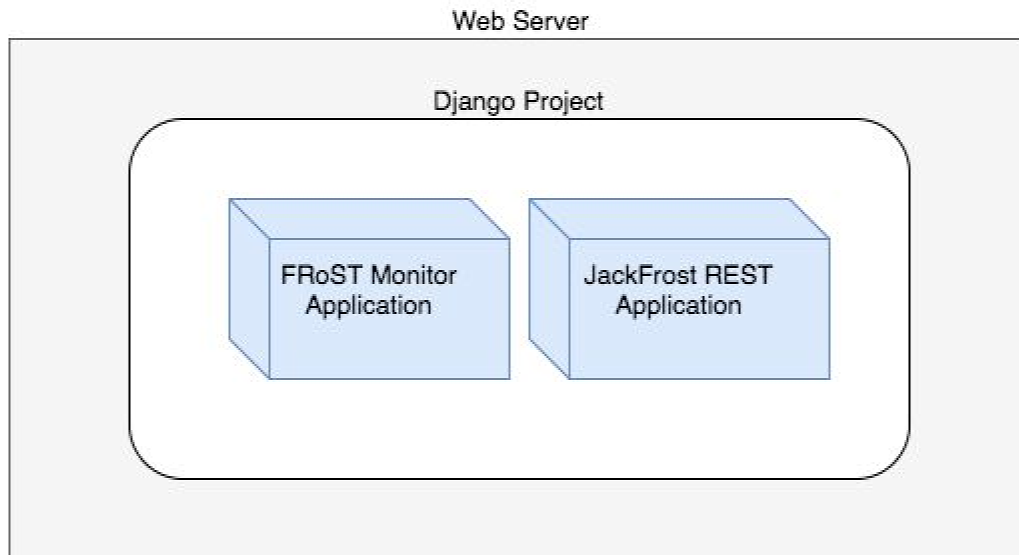

Figure 2: Web Application Layers Diagram

In Figure 2 above you see that there are two applications within the single Django Project. These two applications work together to accomplish the goals of the FRoST Monitor System. The FRoST Monitor Application is the application that is built upon the standard Django framework and is modeled after an MVC architecture. The JackFrost REST Application is the

application that is built upon the Django REST Framework, and follows a REST architecture. Keeping the standard Django application separate from the REST application allows us to keep a separation of concerns within our Django project, and will allow for easy modification in the future. Both of our applications live within the same Django project, and that Django project is hosted on a web server so that it is available online.

Figure 3 below is a diagram of the FRoST Monitor and Rest application layers. In the diagram below you can see that there are three main layers: Models, Views, and Controllers. Inside the Models you can see that there are the actual data objects that our application uses to store and display, as well as the REST Serializers that are able to serialize our data into JSON and make it easily consumable by the REST api. The Controller layer handles the GET/POST requests made by users and our application to send and receive data. The View layer is the piece of the application that allows our webpage to be created and displayed with data from the models.

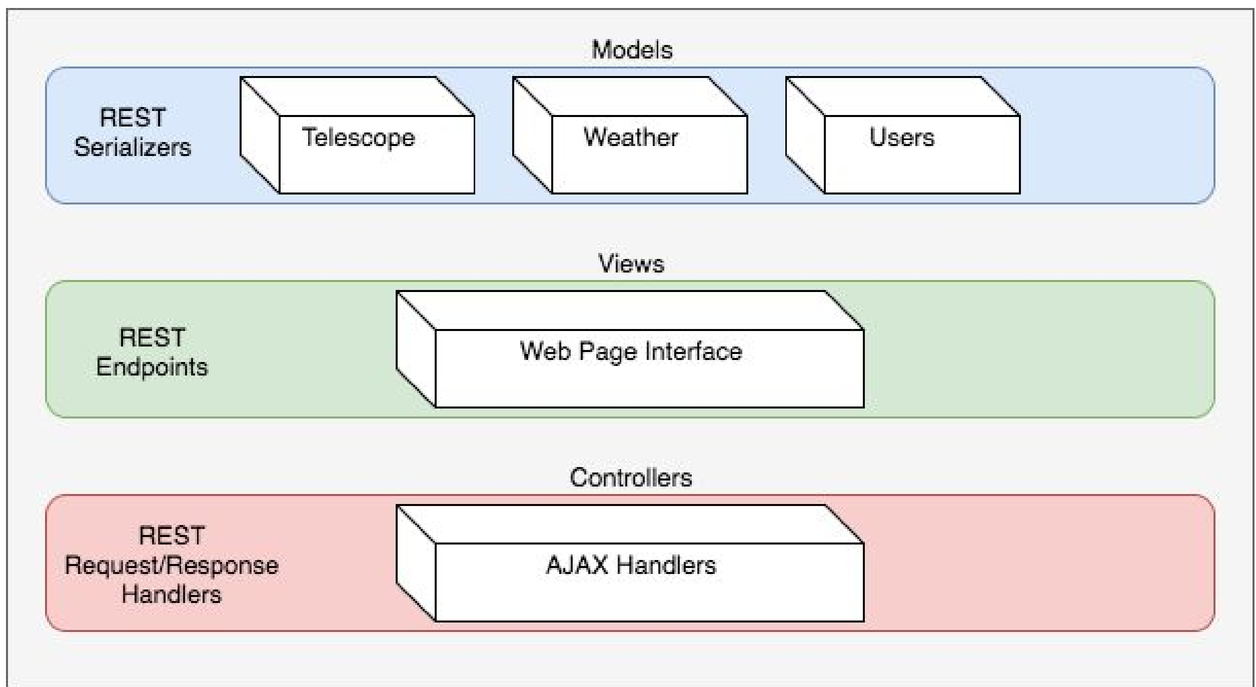

Figure 3: Layered View of FRoST Monitor Architecture

The FRoST Monitor application will use a fusion of two architectures Model, View, Controller architecture, and RESTful architecture. Inside our Django project you may recall that we have created two separate applications. The MVC will be used in the implementation of the web application interface and the RESTful architecture will be used for the jack frost application, which will handle http request to api endpoints including ajax request from the

frost monitor web application and external request from data collecting gateway machines. An overview of how the RESTful architecture works with our standard Django MVC architecture can be seen below in Figure 4.

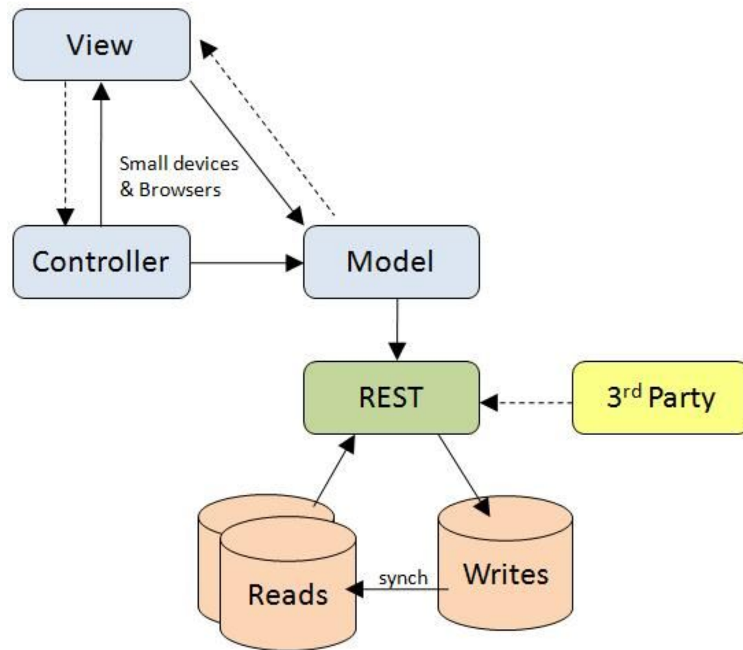

Figure 4: REST & MVC Architecture Diagram

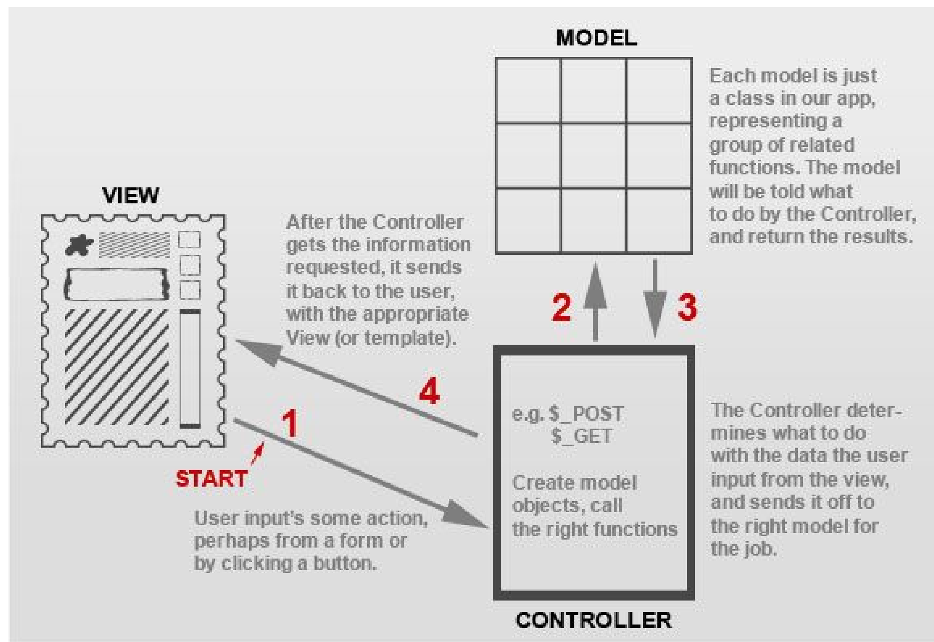

Figure 5: MVC Diagram

Module and Interface Description

Django

MVC framework vs Django MTV framework.

Our project will be using django at its core. This allows us to set up the website using a commonly used MVC framework. The MVC framework consists of 3 main components; Model, View, and Controller. The model portion of the framework consists of all of the classes that we will need for the project. The View is basically what will appear on the webpage. The controller is what links the Model and the View together.

In Django the underlying MVC architecture is actually slightly different from the classic MVC approach. Models are still Models in Django, but a View is actually called a Template, and a Controller is actually called a View. This means that a Django Template is actually what you see on the webpage, and a View links the classes in a Model together with a Template.

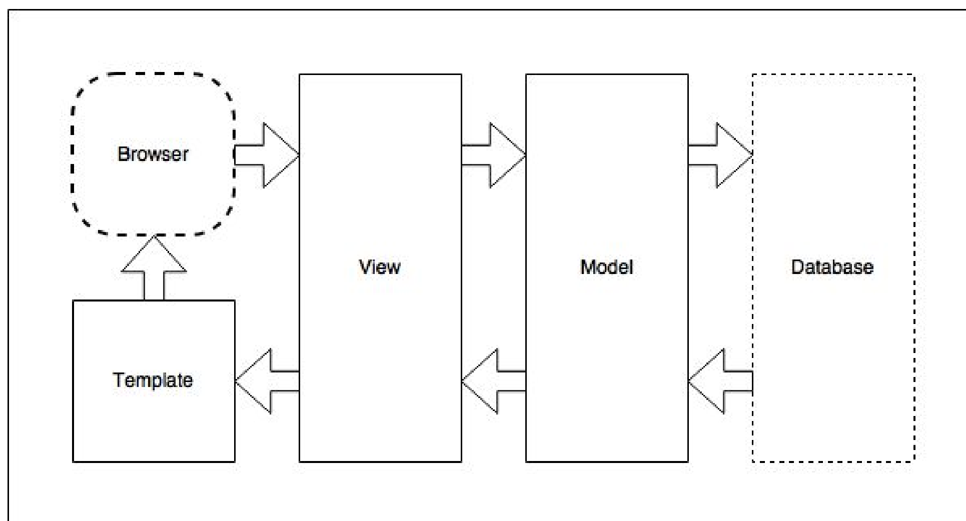

Figure 6.1: Django MTV Diagram

Flow of data in a Django MTV framework.

Figure 6 shown above showcases a brief overview of the flow of data in a basic Django application, beginning with a request from a browser and resulting in a web page produced back to the browser.

When a request is made to view a webpage provided by a Django application, it is first referenced in a list of url patterns located in a file "url.py". The url patterns in this file will link directly to the View portion of the MTV framework by accessing a file called "views.py"

The file "view.py" basically holds all of the functionality for the Django application (which explains why we reference these as "controllers"), and uses the classes defined in your Model to manipulate the data before sending the data to a template.

The Model keeps all of the models in a file labelled "models.py". Once a class is defined in this file, any objects created from each class will automatically be added to an SQLite database that is maintained inside the Django app. The requested data from the database will then be returned back to the View, and then returned to template.

Templates are used to dictate how the processed data will look on a webpage after it has been requested. A template consists of all of the basic utilities that can be included in any html document. Each page in a Django project will require its own template.

Urls.py

The url patterns in the "urls.py" file include "index", "admin", "login_admin", and "ajax". Each url pattern sends a request to the View which calls a function by the same name in the "views.py" file. For example, the url pattern for "index" uses a line of code called "views.index", and will call the function "index", from the file "views.py". This means that "views.py" will only consist of the 4 functions listed above. This "urls.py" file handles the url routing for the django application.

Models.py

In Django, database tables are created via python classes. These classes are individually referenced as a "model" and all together we call our database entries the "Models". In the FRoST Monitor System we currently have three models which are the Telescope Data, Weather Data, and the User Data.

Telescope Status Class

The fields for the Telescope Data are "pointing_position", "dome_position", "local_time", "date_added", and "last_issued_command". This model is displayed within our template so that our users can view the telescope data at anytime.

The "last_issued_command" field will be useful in the case of any disruptions in communication between the application and the dome, as well as any unforeseen failures, because it will be possible to see what the telescope was doing during a point of failure. The "date_added" field gets stored along with all of the other information in the class when a new instance of "TelescopeStatusInformation" is created in the database. This allows the View to request the most recent instance of the class in the database, or request any amount of instances, and order them by date, which would be handy for building a chart that tracks status over time. For example, it would be possible to view the last issued command given to the

telescope, or even view a list of the last 30 commands on a specified date range, by using both "last_issued_command", and "date_added" fields.

The "dome_position" and "pointing_position" fields are important entries for the class. These fields allow us to trace the facing direction of both the telescope and the dome, as well as compare the orientation of each with each other. A mismatch in alignment between the telescope and the dome would result in black images taken by the telescope, because it is actually photographing the inside of the dome rather than the night sky.

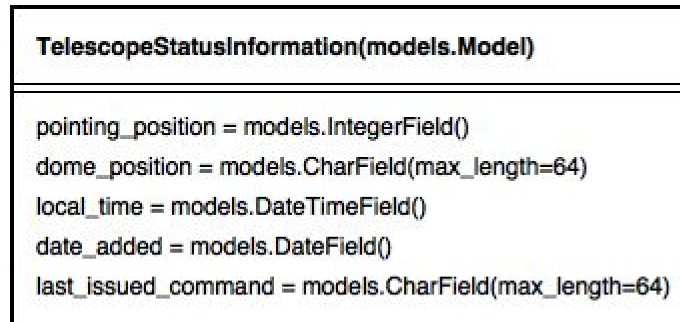

Figure 6.2: Telescope Status Class UML

Weather Status Class

The Weather Data model is made up of "current_temp", "humidity", "dewpoint", "wind_speed", "wind_direction", and "local_time". These are the attributes of our weather model that allows users to view important weather information within our application.

As mentioned above, the time and date fields will allow whether information to be requested and displayed across a wide range of time. This will allow us to display the weather data in different ways. For example, a graph could be constructed that would show the temperature over the course of the previous 3 days, or all of the temperature readings could be gathered in the range of an entire day to produce a high and low temperature for that day.

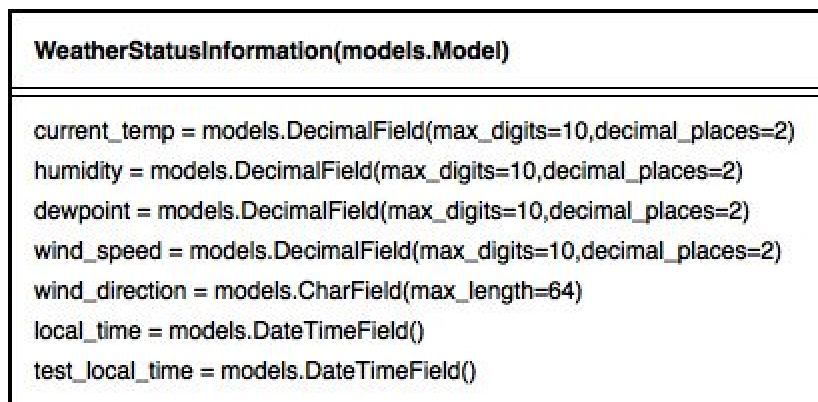

Figure 6.3: Weather Status Class UML

User Class

The User model is actually a model that is built in to Django and does not require setup or explicit definition. The default fields for a user in Django are "username", "email", "admin_status", "password". These fields are all we need for our user models so we intend on keeping them the way they currently are.

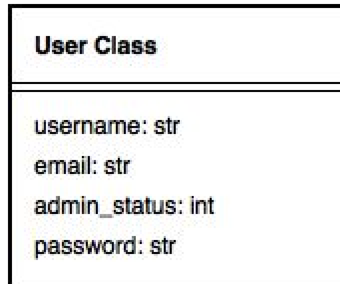

Figure 6.4: User Class UML

Views.py

"Views.py" will consist of 4 functions; "index", "admin", "login_admin", and "ajax". These functions return data to specific web pages within our web application. Each of the functions in the View are referenced by the "urls.py" file, and are called after being requested by the corresponding URL address in the Django application.

The View is not responsible for how the data is displayed, but rather what data will be displayed. The View is the section in which all of the functionality for the web application will reside. It is responsible for requesting queries from the database, and then manipulating and organizing the data before passing it off to a template for displaying to a bowser. For example, the request for weather information in a date range to be displayed in graph for would start in the "views.py" file.

Index

The index function currently generates the most recent weather data to display on the index page. This will eventually be expanded to include extra data such as the latest all-sky-cam image, last given telescope command, and any other pieces of data that would be relevant to have on the front page.

Login_Admin

This function simply calls "login.html" template which requires no additional data from the database. This login web page allows users to input a username and password.

Admin

This function checks to see it the correct login information is given. If the login information does not match the information saved in the database, then the user is referred back to the previous login screen. If the information provided is correct, then the user is referred to the "admin.html" page, which will contain the emergency shutdown button for the telescope.

Ajax

The purpose of the "ajax" function is to gather the most recent weather information. This information will be updated periodically and reflected on the webpage without the need for a page refresh by referencing the "ajax" output in a template.

Templates

The templates used in the Django application are typical of any webpage in that they can be written in HTML, and use tools to augment the page such as CSS and JavaScript. The fields in the Model classes are called and manipulated by the View which results in a query of instances from the Model classes. These instances can then be utilized in a template by referencing the returned results from the View. This makes the content displayed in the webpage to be dynamic.

In a Django application, the html content you view is not always static, which is great for modern dynamic web applications. Templates can be fed dynamic queries that are requested by the View before the pages are actually displayed to the user. For example, every time a user visits the "index" template within our project, Django's templating system will go and fetch the latest database information to be loaded into our template. This is a very powerful tool, because this allows our index page to always display the latest data from the database, without the need for a POST request or a page refresh.

We intend on continually leveraging the template system to make our application as useful as possible.

Implementation Plan

The following section outlines the steps and milestones that need to be completed so that the FRoST Monitor System can be implemented on time. Figure 7.1 below depicts a rough project timeline and includes 10 major implementation milestones.

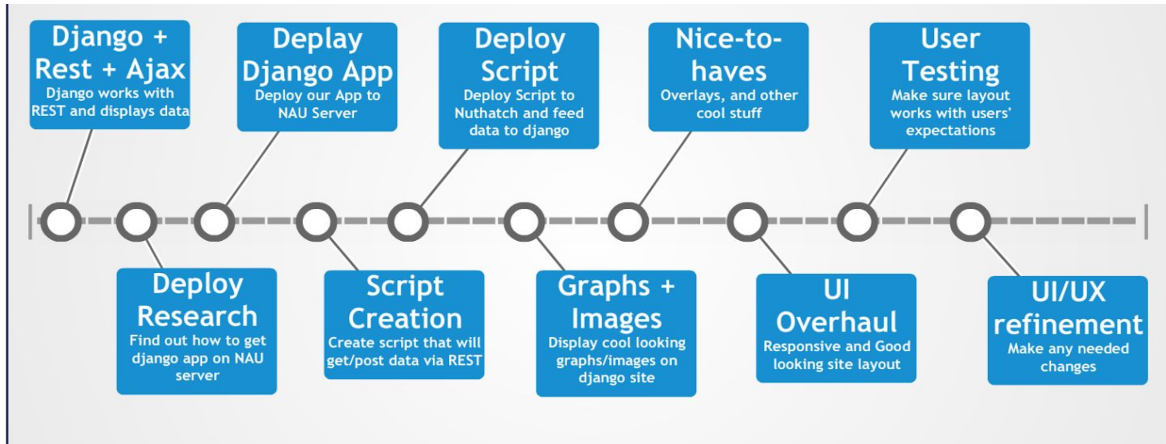

Figure 7.1: FRoST Monitor System Timeline

Figure 7.2 below depicts a gantt chart representation of the project's current milestones and completion dates.

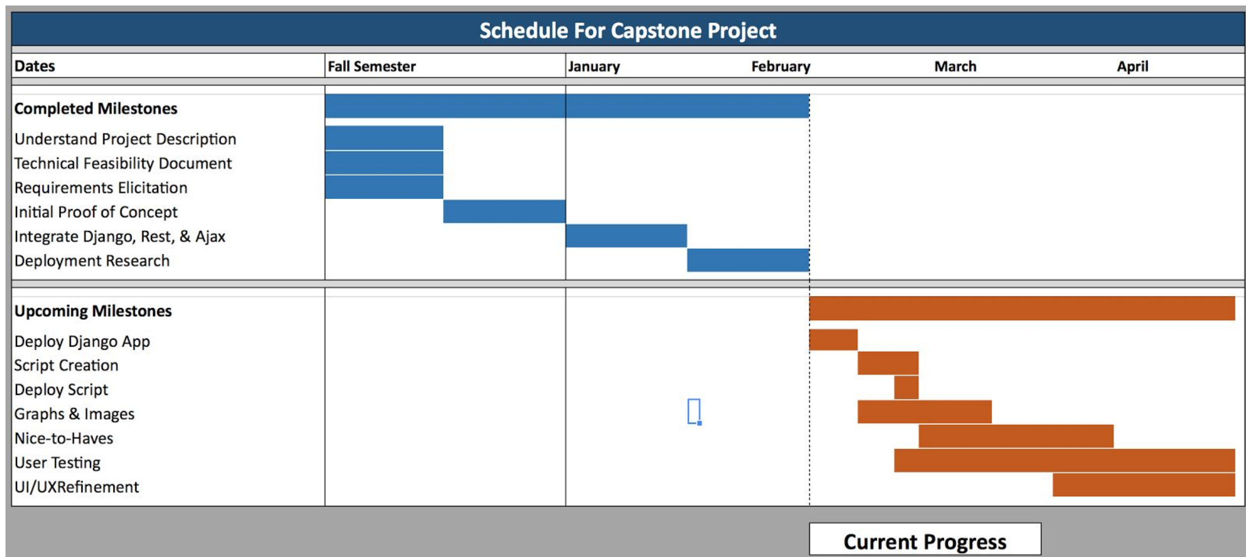

Figure 7.2: FRoST Monitor System Gantt Chart

1. Django with Ajax

An important part of the FRoST Monitor System is being able to display the latest weather and telescope data within a webpage. In order to update our webpage dynamically with current data, we will need to get our django application working together with asynchronous javascript (Ajax) calls that fetch updated data. Without the use of Ajax, our webpage would need to be refreshed in order to display the latest data - which takes valuable time. As of the time of writing this document, this milestone has been completed.

2. Deployment Research

Successfully implementing our application requires us to be able to actually deploy a live version of our webpage to the client's server. Due to our team's lack of experience in deploying django projects to live server environments, this milestone consists of our team coming together and researching the details of django deployment.

3. Application Deployment

Once our team figures out a way to deploy a simple django web application to our clients server's, we will then be able to deploy our main developed code for live testing. After this milestone is completed, our project will be publically available and easily testable.

4. Script Development

The FRoST Monitor System will be relying on a script that is able to fetch and post JSON data to our deployed django application via the Django Rest Framework. This script is in the early stages of development now, and should be completed within a few weeks time. Once this script is created, it will be placed on nuthatch and feed our webpage the latest weather and telescope data.

5. Script Deployment

After developing the script that fetches and posts the latest data to our webpage, we will need to deploy the script onto the client's nuthatch machine. Nuthatch is a machine that has access to the latest data for our application, thus is required for the script to post accurate data. Once the script is deployed to nuthatch, the FRoST Monitor System will be using and displaying the latest data.

6. Graphs & Images

Implementing the ability for our application to display graphs and images is an important part of the FRoST Monitor System. Having the ability to view graphs and diagrams makes it very easy for users to view large datasets quickly and concisely. This milestone consists of researching the latest and best graphing techniques inside of a django application. Our team has researched a few solutions for this milestone and have decided that a JavaScript framework will be what we use to display graphs. Django is able to store and display image data within its sql database, therefore we intend on using this built in feature to display images within our webpage.

7. Nice-To-Haves

At this point in our projects implementation and timeline, the major requirements for the project have already been completed. This project milestone is intended to cover the "nice-to-haves" of our project such as a Telescope Status Overlay. The telescope status overlay will be an image of the night sky overlaid with the actual current pointing position of the telescope relative to the sky. The implementation details for this section are not very clear as our team is focused on some of the key requirements of our project at this time.

8. User Interface Overhaul

After completing the nice-to-haves for the FRoST Monitor System we will work as a team to revise the interface and usability of our web application. Focusing on building a clear and concise interface is crucial for the success of our application because our project relies on having the ability for users to quickly view important information. Once we believe our interface has a solid design and is easy to use, we will turn our focus to making the webpage responsive and accessible for all users on all devices.

9. User Testing

Once our interface has been designed and deployed we will need to test the usability of our web application with actual users. We plan on having our sponsor's and others test the interface and features of our web application so that we can better understand any remaining issues. Once we have detected any issues we will try our best to improve the interface and application until it is working as intended.

10. Application Refinement

The final step to implementing the FRoST Monitor System is refining both the interface and application features so that they meet the needs and requirements of our sponsors. We will dedicate a good amount of time to demoing our application to our clients and revising any issues that arise from those demos. Once we have completed a few refinement cycles, our application should be completed and fully functional.

Conclusion

As a team, we feel that this project has great potential to very useful to our client. It will enable them to more effectively use their telescope to scan possible impactors and learn about the NEOs, and thus making a large impact in their productivity.

This project consists of three main parts: the Nuthatch machine, the Django Rest framework, and the FRoST Monitor web page itself. This makes the project as simple as possible which makes it easy to design and implement. In the end, our client will be maintaining it, so having a simple design is best for them and make them happy about having this product.

References

- Figure 4: REST & MVC Diagram, http://criticaltechnology.blogspot.com/2011/08/mobile-devices-and-good-architecture.html

- Figure 5: MVC Diagram, http://www.onextrapixel.com/wp-content/uploads/2012/02/mvc-model.jpg