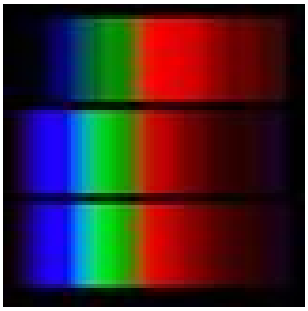

User Manual

2023-04-26

Project Sponsor: Joe Llama, Lowell Observatory

Faculty Mentor: Rudhira Talla, Northern Arizona University

Team: Empyrean- Henry Fye, Nhat Linh Nguyen, Jakob Pirkl, Jacob Penney, Kadan Seward

Overview

This document should act as a guide to the user concerning all matters related to our software product. It should effectively educate the client on how to install, operate, and maintain the product.

Table of Contents

1.0 Introduction

Team Empyrean is pleased that you have chosen the Spectrograph Control System. The Spectrograph Control System is a powerful system for working with a real spectrograph remotely that has been designed to meet your needs. Some of the key features include: request the observation, status of the system updated and log sheet of observations. The purpose of this user manual is to help you, the client, successfully install, administer, and maintain the Spectrograph Control System in your actual business context going forward. Our aim is to bring advantages to your work and make sure that you are able to profit from our product for many years to come!

2.0 Installation

As part of this delivery, we have already deployed this application at Lowell, on a mac connected to a camera and spectrograph. Over time however, you may want to reinstall the project on a new machine. The following directions will take a device from fresh install to a production machine, ready to serve.

1. Download Code:
   We recommend creating a new repository for this code. Then run the following inside that directory:

   ```
   $ git clone https://github.com/Empyrean-Capstone/Empyrean.git
   $ cd Empyrean
   ```
2. Install npm, postgres:
   Ensure that the following are installed:

   ```
   $ brew install npm
   $ brew install postgresql
   ```
3. Make a python virtual environment
   We need a virtual environment so that we can install packages without version management. This is so the installation does not ruin other packages that have been installed. The environment can be created and activated with the following:

   ```
   $ conda create --name {name of environment, i.e. empyrean} python=3.9
   $ conda activate {name chosen earlier}
   ```
4. Install python requirements:
   We need to move into the directory in which the file requirements.txt is, then we can install the requirements with the following:

   ```
   $ pip install -r requirements.txt
   ```
5. Configure python variables:
   In the directory main, there is a .env file. Move into this directory, and open this file, changing the variable for DATA_FILEPATH to the location where FITS file should be saved. The default is the directory created in step one. Remember this file, as we will be coming back to make one more change to this file.
6. Install npm requirements:
   Move back to the root directory of the project and install the node packages needed for react by running:

   ```
   $ cd ../../
   $ npm install
   ```

7. Configure database:
   To get data, we will need to set up postgresql so that our app can access a database. Open postgresql, create a new user with the ability to create databases, and create a database with:

```
$ psql -U postgres
        > CREATE USER empyrean WITH PASSWORD '{chosen password}' CREATEDB;

        > CREATE DATABASE {name of database};

        > \q
```

   Next we need to edit the .env file from before, changing the line SQLALCHEMY_DATABASE_URI=”postgresql://{chosen_username}:{chosen_password}@localhost/{created_database}

8. Migrate database:
   We have an empty database, but we still need to generate a schema for this database. Move to the api directory, then upgrade the database with the following:

```
$ cd /path/to/project/root/Empyrean/api
$ flask db upgrade head
```

   Then, a default database can be migrated from the root directory with:

```
$ psql {created database} < ../documentation/base_database.sql
```

9. Run the project:
   The project can now be run. Two terminals will be needed as one server will serve React, while one will serve the Flask application. These can be run with:

```
# The python project can be run with:
$ python api/wsgi.py

// The React project can be run with:
$ npm start
```

10. Installation of Instruments:
    The instruments are run using the same environment as the main application. The prerequisite is that the camera is able to be used already with the device being developed on, and that the camera works with ZWOasi.
    a. Install the code for the instruments in the root directory of the project:

```
$ git clone https://github.com/Empyrean-Capstone/shelyak_control.git
```

    b. Each of the instruments can be run in separate terminals with:

```
$ python spectrograph.py
$ python camera.py
```

11. Follow the steps from A.3 to install the application in a development environment.
12. Install nginx:
    Install with:

    ```
    $ brew install nginx
    ```

13. Configure nginx:
    This will make sure we can serve our files from React to a user. We first need to build our project with:

    ```
    $ npm run build
    ```

    Make note of the absolute path of the build path (path/to/project/Empyrean/build). Make sure that this location is accessible to any user. Find the location of the configuration files for nginx with:

    ```
    $ nginx -t
    ```

    Open this file in a text editor, and add the configuration found in Appendix B, making sure to change the location of the ‘root’.
    After this, we will start nginx as a service with:

    ```
    $ sudo brew services start nginx
    ```

14. Start Flask server as a service:
    We already were able to run the flask server, but the server will stop whenever the computer loses power. Starting it as a service means that the server will start back up after power is lost.
    We need to add this process to the service list. First, make not of where the python we are running is located, then get to the directory where these services are stored, where we can then create our service with:

    ```
    $ which python
    $ cd /Library/LaunchDaemons/
    $ touch com.empyrean.plist
    ```

    The services for each of the instruments can also be made with:

    ```
    $ touch com.empyrean.camera.plist
    $ touch com.empyrean.spectrograph.plist
    ```

Copy into each of these files the contents of each file found in Appendix C, making sure to change the two strings inside of the ProgramArguments key, as well as the filepath to the error and standard output lines to somewhere convenient.

15. Finally, restart the machine, and make sure everything is working. It may be though, that the camera and spectrograph will need to be reloaded, which can be done with:

```
$ sudo launchctl unload  /System/Library/LaunchDaemons/com.empyrean.plist
$ sudo launchctl load  /System/Library/LaunchDaemons/com.empyrean.plist
```

3.0 Configuration and Daily Operation

After the reboot, the shelyak control should be fully operational, with only a couple of tasks that need to be performed for continued maintenance.

Set up New User: In this section, we will start with one default user, and switch to a custom user with greater protection.

1. Log into the system: Use the default login: (username: root, password: password) to login to the system.
2. On the left bar, click the Manage Users tab.
3. On the user management page, click add new user in the top left of the page.
4. For this user, add a username, password, and name, as well as checking the Is Admin box. This will allow this user to manage new users.
5. Log into the new user, to make sure that this has worked.
6. Once logged into the new user, delete the previous root user.

Make Observations: In this section, we will look into how we can make observations on this application.

1. On the main page of the application, after logging in, the top left box outside of the toolbar is where observations are made.
2. Type in the requested object into the first input box, then click resolve. This first resolution will take longer than subsequent ones because the SIMBAD database needs to be downloaded.
3. Click on the number of exposures and length of exposures desired.
4. Click start, and watch the system get to work. You will see that the statuses of the spectrograph will first change, then the camera will begin to take exposures. You will be able to see each exposure tick up as it progresses, and be able to see each observation in the log sheet below. These will both be updated in real time as they progress.
5. Click stop if necessary. This will delete all of the observations made in the batch.

Changing Instruments: Rarely, it may be necessary to change the spectrograph or camera in use with the system. We cannot guarantee that any spectrograph and camera will be “plug and play” so here are some pointers on what will need to be changed.

1. Check that the spectrograph.py and/or camera.py files will work with the new instrument. If the same make, it should, and some cameras of the same model will also work with the same code. But outside of that, the code will likely need modification.
2. Make sure that new code in these files inherit the Instrument.py class. This class is an Interface, meaning that any subclass of it must implement the methods it describes as abstract. For this situation, any subclass must implement: get_instrument_name, which programmatically get the instrument name of the system, and callbacks, which is where all socket events are found. More information about these socket events can be found in the final-as-built Report.
3. Make any necessary modifications to the observation code. This will be in the actual server, and likely changes will be in the /api/main/observations/views.py file, with some

being in the /api/main/status/views.py file. In particular, attention should be paid to the “post_observation”, “setup_camera”, and “end_exposure” functions in the observations file. In the status file, the function “get_current_camera” may need to be changed to reflect the name of the new camera.

Tips on searching through past observations: This section will have some general tips for looking through past observations. The main logsheet can be found as its own dedicated page on the left bar, and at the bottom of the making observations bar.

- If not all columns are necessary, the first button on the toolbar allows a user to select which columns to look at.
- If a user wishes to sort by, say, what they observed, the second button will open a popup, allowing a user to select a column to search by as well as the value they want to compare against.
- To look for observations made between two dates, heading to the main ‘Explore Logs’ page will allow a user to do this. Notice the additional box in the top left. Selecting two dates will allow a user to look at only logs between these two dates.

4.0 Maintenance

There are several steps that the administrator can take in order to maintain the long-term health of the system. The primary tasks that the development team sees as necessary concern the production database and persistent logging outputs (log files). The chosen database technology can handle very large quantities of data, but, as that data accumulates, it becomes necessary to monitor for outdated information and make sure that the troves of valuable data collected are stored elsewhere for easy retrieval in the case of unforeseen catastrophic failure which results in data loss. Tasks related to protecting one's data and maintaining the health of the database include

1. Managing the relationships between observation entries in the production database and the files in the host file system.

    The product does not monitor the host filesystem to maintain accuracy in the database. For example, version 0.1 (the current version) adds log names and resulting file names to all observation entries in the table of observations made with our system. If one were to move, rename, delete, or otherwise modify one of the FITS files that has resulted from or is connected to an entry in the “observation” table of the database, the database would no longer serve as a meaningful reference to that item. In order to maintain the accuracy and legitimacy of the database's tracking efforts, one should do their best not to modify output names and locations. If one needs to move that data elsewhere, they should strictly copy the file, leaving the original intact. If a file has been lost, the system administrator should remove the entry from the database which tracks it.

2. Making backups of all tables that contain sensitive data, or data that could not be replaced.

    Among the tables in version 0.1’s database, the “observation” table, and “user” table, assuming one has many users, are those which are most irreplaceable. As mentioned prior, the “observation” table contains (and references, in the file system) data which cannot simply be replicated, as is the case with other tables in the database. Regularly making backups and storing them in a separate place than the database itself is a wise decision. How regular this operation should be performed is dependent on how frequently the system is used (and, therefore, how much output is produced) and how important the observation data in the database is perceived to be by the administrator or users. Again, observation outputs are stored in FITS files in the host file system. The observation data held in the database is essentially metadata that references the true outputs, which should contextualize how often backups are made.

3. Regularly revoking login credentials that are outdated or no longer used.

    If, for some reason, usage of the application has consistent user turnover, the system administrator should delete those outdated or abandoned credentials from the “user” table in the database. This can be done very conveniently via the user management page in version 0.1 of the application.

Besides managing the database, the application’s technologies can produce logs if configured to do so. As with proper operating system management logs should be investigated if the disk space of the host device becomes limited or grows short quickly, or if the operations that append to logs becomes slow (indicating that the log length has become cumbersome). Logs can be produced by

1. Nginx, the HTTP proxy server that the application uses. This log can be found at “/var/log/nginx/access.log”

2. Each of the services configured for the application, including
    a. the camera
    b. the spectrograph
    c. the application front-end
    d. the application back-end

    The logs for each of these items will depend on the inputs given in each service’s launchctl configuration file. For more information, please see Appendix C of the team’s “As-Built” Report.

A closing task of the development team will be to transfer ownership of the product over to the client. This entails transferring the digital ownership of the repository that contains the source code and its version history and establishing any digital permissions for developers who will contribute to the product in the future, given that the repository is open-source. This last point may include establishing permissions in the repository settings on the version control host (GitHub, per the product’s requirements specification).

5.0 Troubleshooting

Throughout the production and implementation of this project, issues would arise constantly and there were certain “gotchas” that would appear time and time again. This section aims to help anyone who is struggling with getting software to run correctly on their machine as these problems often troubled even the developers.

1. **Node Dependency Issues**
   If the frontend fails to compile it could be issues with the node dependencies used alongside React. If these dependencies are not installed and updated properly they could cause issues. To begin, delete the “node_modules” folder. Then, type in the command “npm install”. This will install all dependencies declared in the package.json file. If there are errors that occur after these steps check if the error contains missing node modules. If so, run the command “npm install x” replacing x with the module declared in the error. If the error does not contain missing modules then the problem is being caused by something else.

2. **Flask Dependency Issues**
   This step is very similar to the node dependencies step. If backend calls are not working properly, make sure to check the console to see if any errors occur. If there are errors surrounding flask dependencies, move into the api directory and type the command “pip install -r requirements”. If installing all requirements doesn’t work, then use “pip install x” replacing x with the missing dependency. If the error does not contain missing modules then the problem is being caused by something else.

3. **Tool/Language Related Issues**
   Often, the team would come across issues when moving the project between machines. Many of these issues were caused by different versions of python, react, flask, or postgresql. Ensure that all tools being used are at the same version of the tools used for the project.

4. **Browser-Related Issues**
   Occasionally, the team would run into issues concerning the login system. Often, these issues were caused by a browser, in our case Firefox, blocking certain functionalities. If the web application is not behaving as expected, try using a different browser.

5. **Instrument Issues**
   If the web application loads but the status table is completely empty, that means there is some issue with the connection to the instruments. Ensure the instruments are correctly connected to the system and functioning correctly. If it does not seem to be a hardware issue, check to be sure that the instruments are all still running as services.

6. **API Issues**
   Sometimes, if the API malfunctions for an unforeseen reason it may shut down the server and calls will no longer work until it is restarted. Ensure that the Flask backend is running if there are API issues and check the console for more information on what caused the server to malfunction.

6.0 Conclusion

All of us at Empyrean have enjoyed working on this amazing experience in our final year at NAU. We all agree that this project has been enjoyable, fulfilling, and interesting. Thank you for all your support throughout this project, both in the form of virtual communication as well as the numerous in-person meetings. We hope the solution we have developed for you satisfies all of your needs, and is easy to maintain and iterate upon. It is our sincere hope that some truly amazing observations can be made while using our software. While we are all moving on to professional careers, we would be happy to answer any questions in the coming months to help you get the product deployed and operating optimally at Lowell.

With best wishes from your Empyrean developers,

Henry Fye (shf26@nau.edu)
Nhat Linh Nguyen (lnn45@nau.edu)
Jakob Pirkl (jap743@nau.edu)
Jacob Penney (jmp458@nau.edu)
Kadan Seward (kos34@nau.edu)