MapONE

User Manual

May 5, 2022

Sponsors:
Planetary Geologic Mapping Program, USGS Astrogeology Science Center
Dr. Sarah Black, Research Physical Scientist
Marc Hunter, IT Specialist

Faculty Mentor:
Melissa D. Rose

Team Members:
Samantha Milligan
Michael Nelson
Ricardo McCrary
Jacob Stuck

Overview: The purpose of the User Manual document is to provide a guide to the product's main features including the system's installation, configuration, and maintenance.

Table of Contents

1. Introduction 3

2. Access, Installation, & Configuration 3

2.1 GitHub Repository 3

2.2 Heroku Account 3

2.3 DockerHub Account 4

2.4 Frontend 4

2.5 Backend 6

3. Daily Operation 7

3.1 GUI 8

3.2 Backend APIs 15

3.3 Web Scraper 23

4. Testing & Maintenance 26

4.1 GUI 26

4.2 Backend APIs 27

4.3 Web Scraper 28

5. Conclusion 29

1. Introduction

MapONE is a web application designed to collect planetary map publications for the planetary science community. This manual serves as a user-friendly guide to quickstart the application. This document details how to install and administer MapONE during the configuration, deployment, and maintenance phases. Features include an interactive user interface, multiple extensive Application Programming Interfaces (APIs), and an automated web scraper designed to extract publication source data. All modules have been configured and will be discussed in length throughout this manual.

2. Access, Installation, & Configuration

MapONE consists of three modules: a Graphical User Interface (GUI), backend APIs, and a web scraper. These modules are divided into two domains, the application's frontend (GUI) and backend (APIs and web scraper). The domains run on two separate remote servers where the frontend completes user requests through API calls to the backend. For the purposes of this manual, each domain can be accessed and should be installed separately.

2.1 GitHub Repository

All files and systems discussed in this document can be accessed at https://github.com/samantha-milligan/MapONE. This is a GitHub repository that stores MapONE's code material. At the time of the product delivery, this repository should be private and ownership should be transferred to the client. The repository contains a "mapone_frontend" directory and a "mapone_backend_docker.zip" compressed file. The "mapone_interface" contains all frontend material discussed in this manual. The compressed file is a container of MapONE's backend system.

2.2 Heroku Account

MapONE's remote servers run on Heroku, a hosting platform. At the time of product delivery, the Heroku account's ownership should be transferred to the client. To access Heroku's web application, visit https://www.heroku.com/. There, the client can log in to their account and view

deployment information on both servers. Under applications, the frontend ("mapone-interface") and backend ("mapone-api") servers are listed. Under each server, the client can view additional resources, server activity, and settings. At the time of product delivery, the frontend server should run at https://mapone-interface.herokuapp.com/ and the backend server at https://mapone-api.herokuapp.com/.

2.3 DockerHub Account

Given the complexity of both the APIs and web scraper, the backend has been containerized using Docker. Docker is a platform designed to package code material to easily install dependencies and deploy applications. As previously mentioned, the GitHub repository contains a compressed file "mapone_backend_docker.zip," (see section 2.1). This file contains the backend's Docker container. To access all Docker images and containers used in MapONE's development, visit https://hub.docker.com/ to sign in to an account. At the time of product delivery, the DockerHub account's ownership should be transferred to the client.

2.4 Frontend

MapONE's frontend GUI was created using Flutter, a user interface framework software written in Dart. Although the interface has been deployed on Heroku (see section 2.2), the initial deployment required previous software installation. The following steps are necessary to run and deploy MapONE's interface:

1. **Install** Flutter at https://docs.flutter.dev/get-started/install depending on the operating system. Here, users can download all system requirements to run a Flutter application. Follow all installation instructions including downloading the latest release of Flutter SDK and updating the package source path. Additionally, macOS and Linux users can use Homebrew, a software package management system, to install Flutter at https://formulae.brew.sh/cask/flutter.
2. **Download** the frontend directory from GitHub, "mapone_frontend," onto the client's local machine.
3. **Change** into the frontend directory, "mapone_frontend," using a command-line tool.

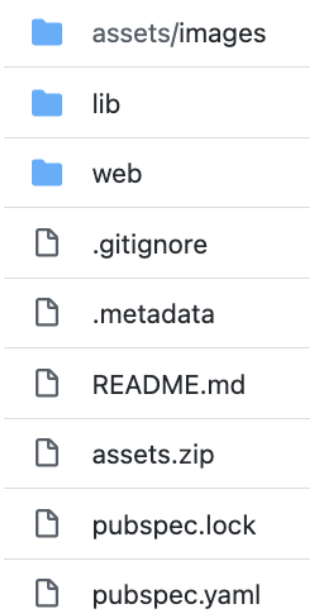


Figure 1. Frontend directory view.

4. **Run** *Flutter run* to run MapONE's interface on localhost. MapONE's interface should now be displayed on localhost on a pop-up browser (this will be automatically opened by the command).
5. **Make** changes to the source material as needed and repeat step 4 to see results.
6. **Install** Heroku at https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli depending on the operating system.
7. **Log** in to the existing Heroku account by running *heroku login* using a command-line tool.
8. **Run** *heroku git:clone -a mapone-interface* to clone the frontend server's source code to the local machine.
9. **Change** into the "mapone-interface" directory where it was cloned in step 8.
10. **Replace** the files in "mapone-interface" with the new changes in "mapone_frontend."
11. **Run** *git add .* and *git commit -m "write-message"* to update changes.
12. **Run** *git push heroku master* to push changes to https://mapone-interface.herokuapp.com/. The frontend should now be updated and deployed remotely.

2.5 Backend

The process to install and deploy the backend server is similar to the frontend but requires additional steps given the Docker container. MapONE's backend system is created using Django, a web framework software written in Python. However, the backend has been containerized using Docker to allow for easier configuration and set-up. The following steps are necessary to run and deploy MapONE's backend server:

1. **Install** Docker at https://docs.docker.com/get-docker/ depending on the operating system. Here, users can download all system requirements to run a Docker container.
2. **Install** Heroku (see step 6 in section 2.4).
3. **Download** the backend compressed file from GitHub, "mapone_backend_docker.zip," onto the client's local machine.
4. **Unzip** the file.
5. **Change** into the backend directory, "mapone_backend_docker/mapone-api/" using a command-line tool.

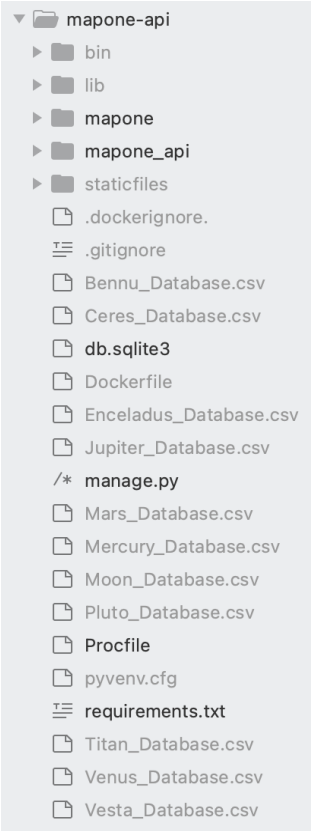


Figure 2. Backend directory view.

6. **Add** the password for EMAIL_HOST_PASSWORD in "mapone_backend_docker/mapone/settings.py". This will allow users to receive emails from gs-g-wr_astro_map_search@usgs.gov.
7. **Make** changes to the source material as needed.
8. **Run** *pip install -r requirements.txt* to install all necessary packages. For this step, it is recommended the client use a virtual environment. For more information, visit https://virtualenv.pypa.io/en/stable/.
9. **Run** *python3 manage.py runserver* to run the backend server locally at http://localhost:8000/.
10. **Log** in to the existing DockerHub account by running *docker login* using a command-line tool.
11. **Run** *docker container stop $(docker container ls -aq)* to stop all Docker containers as this is an existing container.
12. **Run** *docker container rm $(docker container ls -aq)* to delete all containers.
13. **Run** *docker rmi $(docker images -q)* to remove all Docker images.
14. **Run** *docker images -a* and *docker ps -a* to ensure container and image lists are empty.
15. **Run** *docker build -t mapone-api .* to create a new Docker image.
16. **Log** in to the existing Heroku account by running *heroku login* using a command-line tool.
17. **Run** *heroku container:login* to log in to the container.
18. **Run** *heroku container:push web* to update changes.
19. **Run** *heroku container:release web* to push changes to https://mapone-api.herokuapp.com/. The backend should now be updated and deployed remotely.
20. **Enter** https://mapone-api.herokuapp.com/timer/ into any browser to start the system's internal timer (see section 3.2 for more information).

3. Daily Operation

Once MapONE is fully installed, configured, and the client has access to all code material and accounts, the system is fully operational. The following sections discuss the product's daily operations on the GUI, backend APIs, and web scraper levels.

3.1 GUI

MapONE's interface can be accessed at https://mapone-interface.herokuapp.com/. Here, users will be able to create or log into a user account, view and filter publication source data, and create automated searches. For an additional tutorial, please view MapONE's demonstration at https://youtu.be/O9Ndj1D5o_o.

MapONE's Taskbar

The interface's taskbar is located at the top right corner of each MapONE page. In *Figure 3*, from left to right, the icons are listed in the following order: User Profile, Filter, Main Page, Login Page, and Search.

Figure 3. MapONE's taskbar.

Login Page

MapONE's website directs users first to the login page. Here, users can log in to an existing account, create a new account, continue as a guest, or reset their password if forgotten. Guest users will have access to all publications and can search, view, and filter source data by keyword, map body, or publication year. Additionally, existing and new users will have the same privileges as guest users in addition to creating automated saved searches and receiving notifications on new map additions (see section 3.2 for more information).

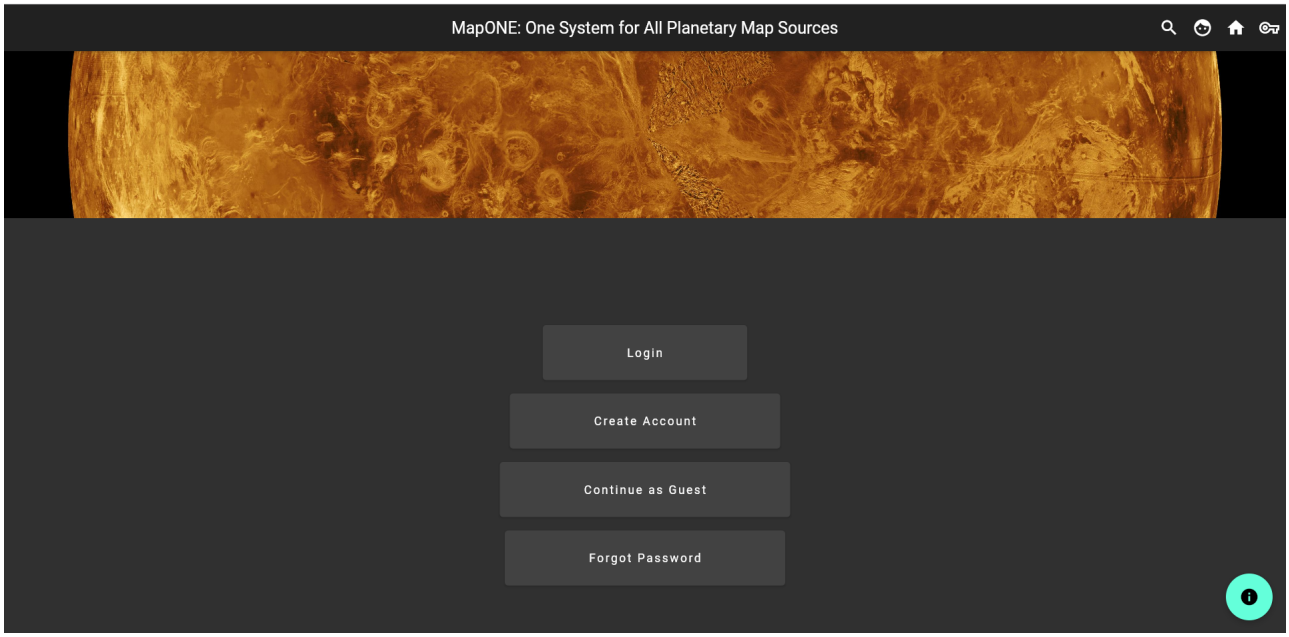


Figure 4. MapONE's login page.

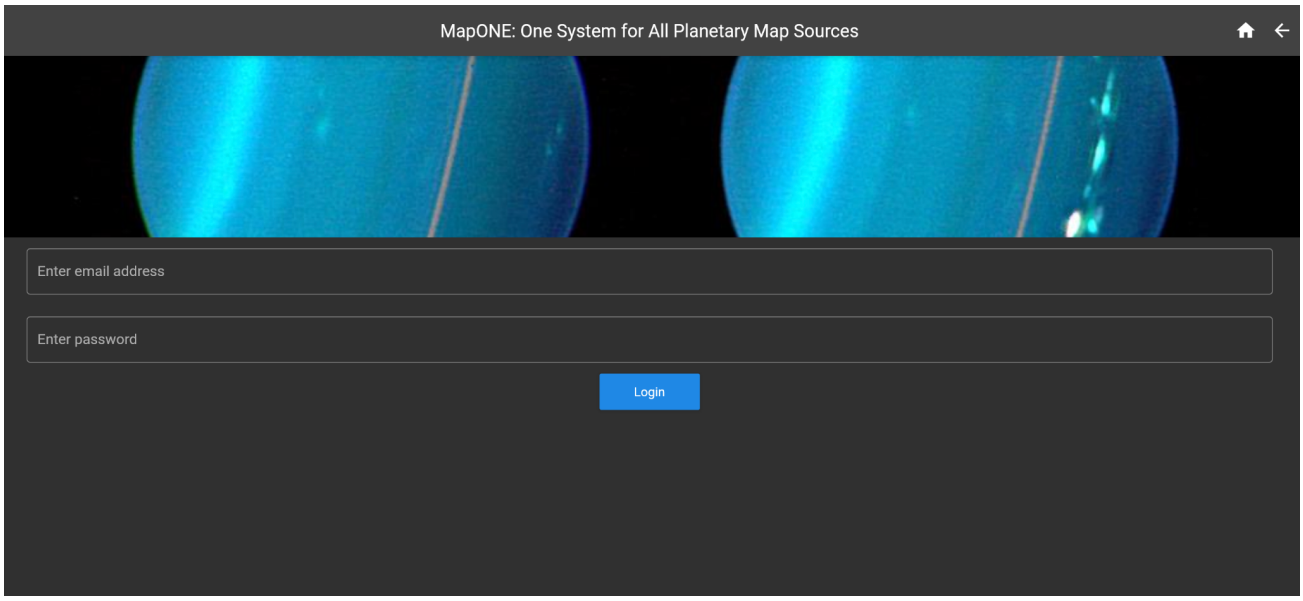


Figure 5. Log in as an existing user.

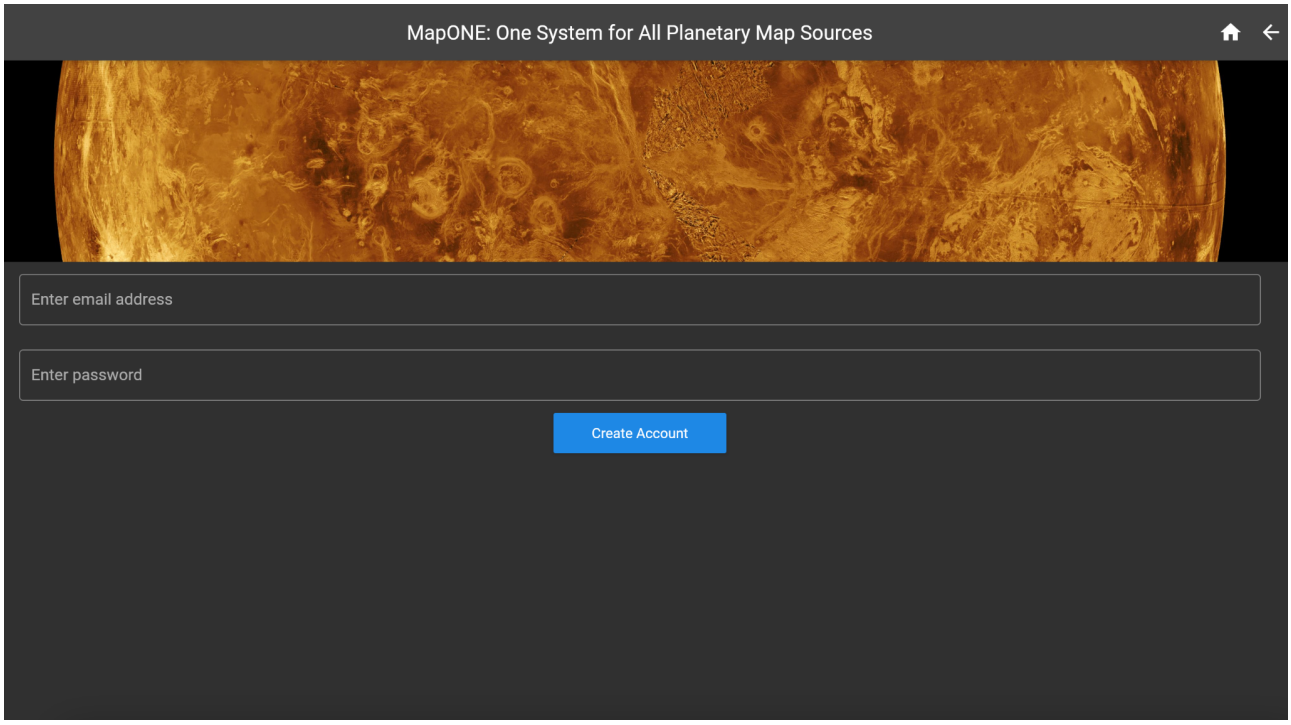


Figure 6. Create a new user account.

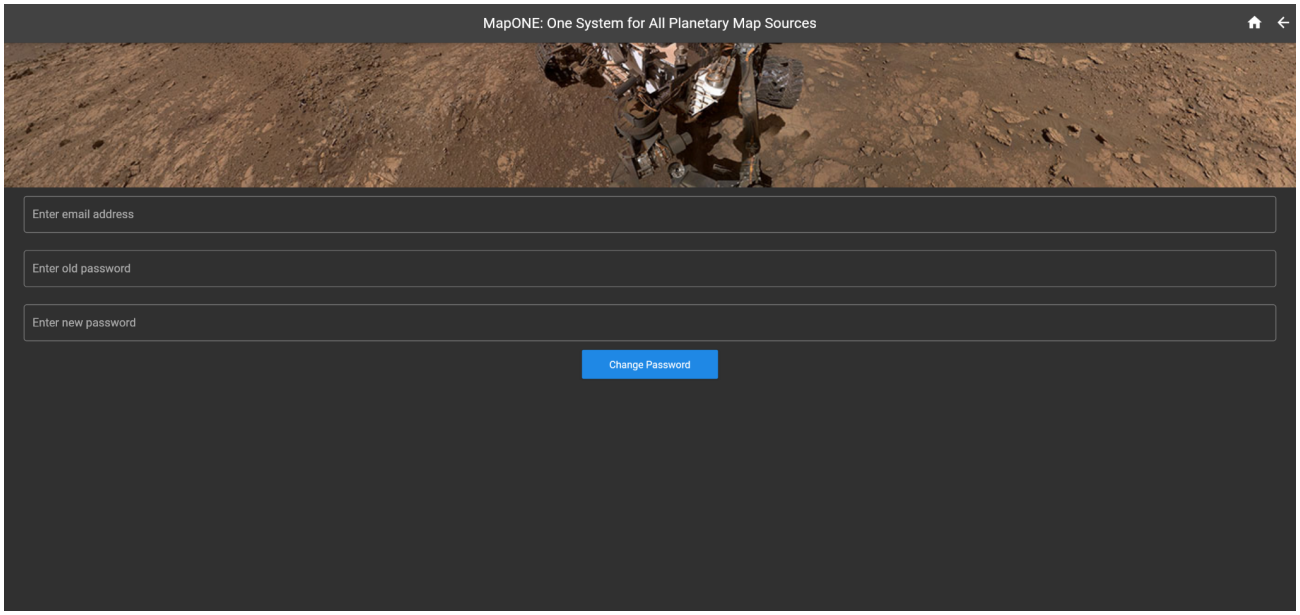


Figure 7. Change password.

Main Page

Once logged in or continued as a guest, the user will be directed to MapONE's main page. This page displays publication source data including source name, source link, map body, article title, author(s), and publication date for all entries. At the time of product delivery, users can scroll through over four hundred publications pulled from various science journals with the following map bodies: Bennu, Ceres, Enceladus, Jupiter (including moons), Mars, Mercury, Earth's Moon, Pluto, Titan, Venus, and Vesta.

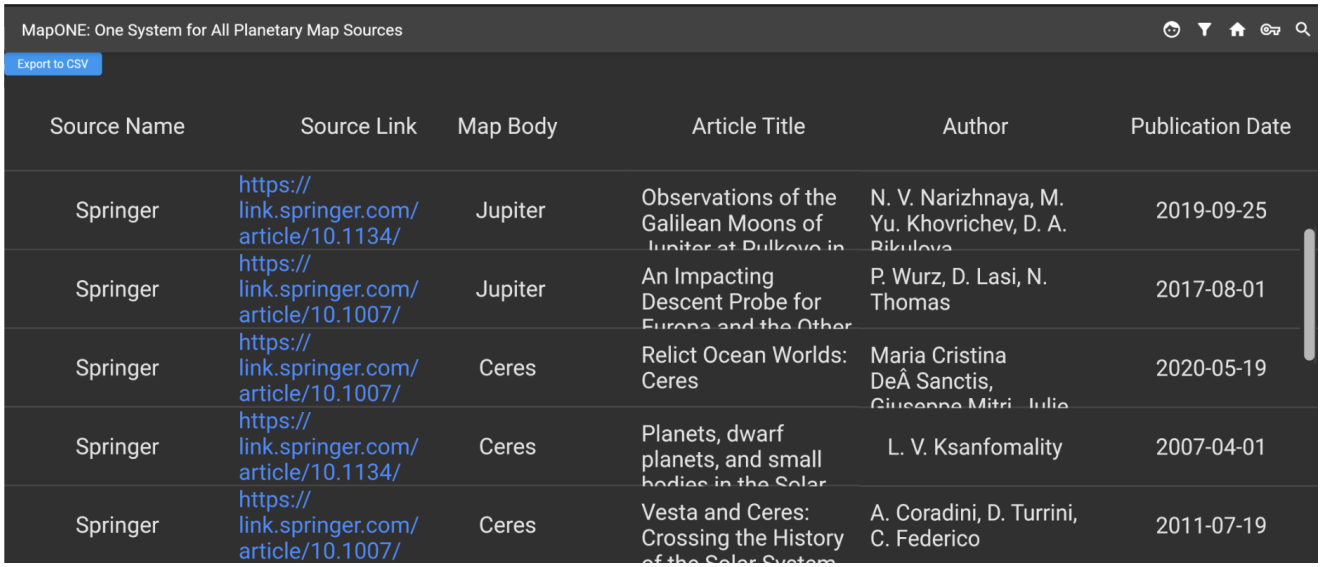


Figure 8. MapONE's main page.

Users can also search publications by a specific keyword or map body or filter by publication date. The application will pull up publications that only meet the search and filter criteria.

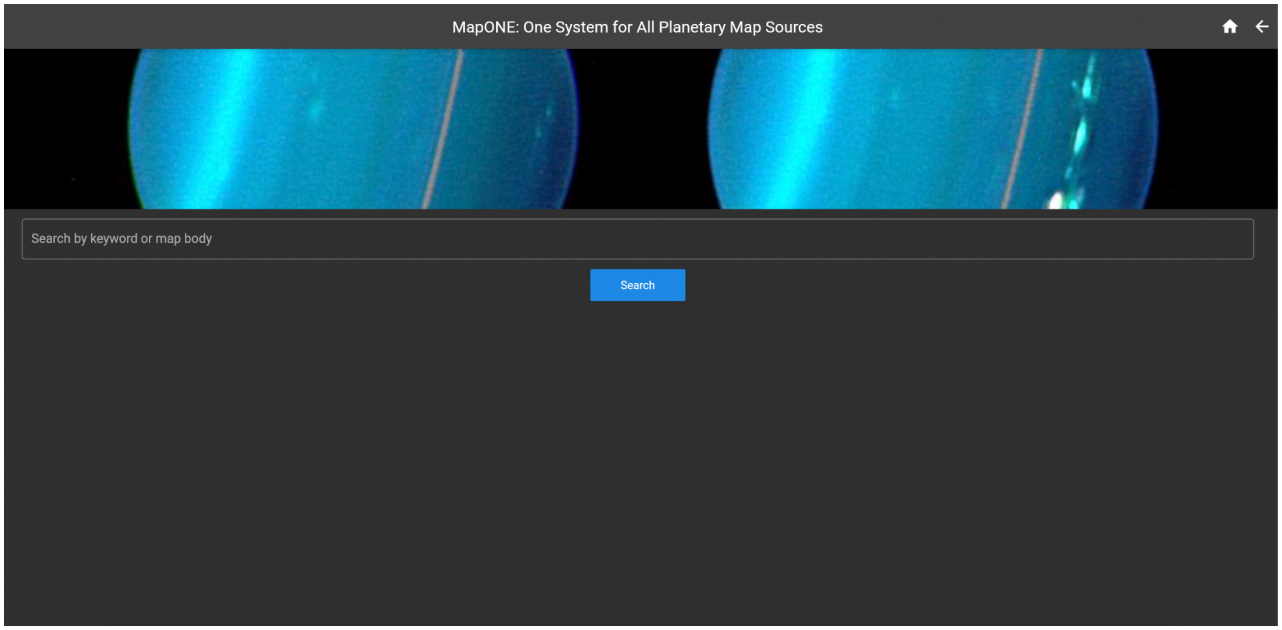


Figure 9. Search by keyword or map body.

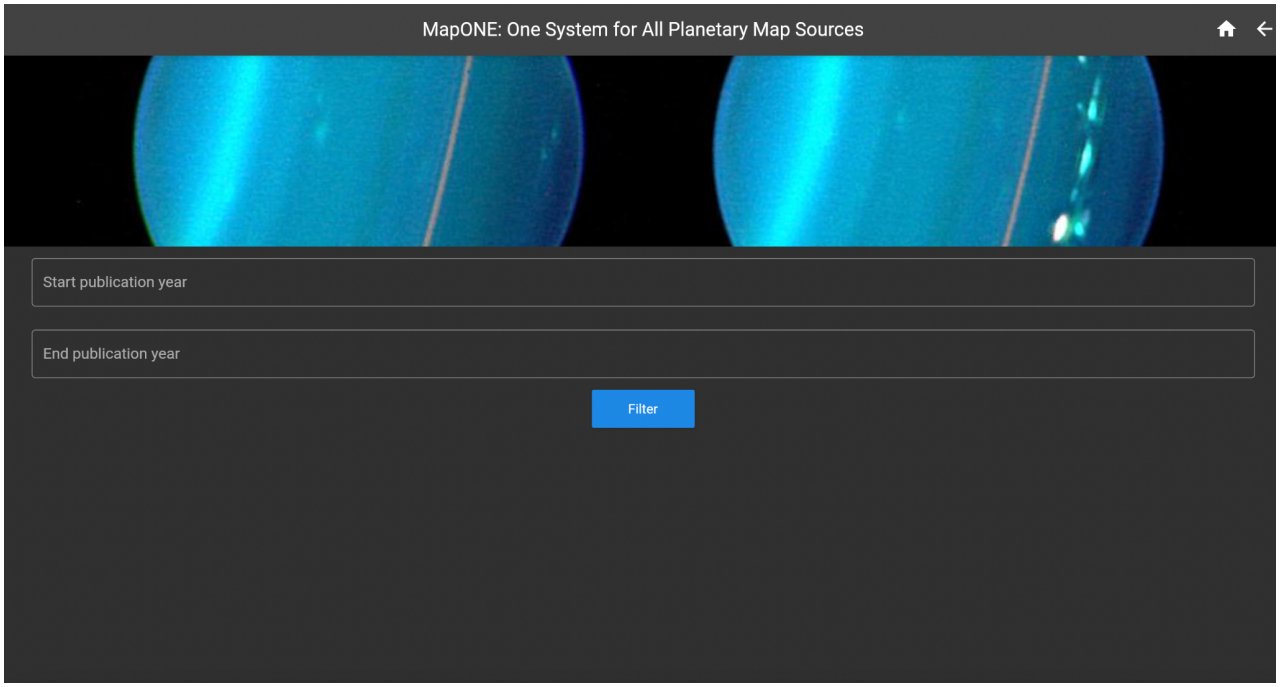


Figure 10. Filter by publication year.

Lastly, users can submit feedback on the main page. Because MapONE is intended for the planetary science community, integrating community validation and input is a crucial step in the product's development. As a direct measure of the product's performance, users can submit

feedback on each publication. Users can validate if a publication contains/discusses a planetary map. Users can also confirm if the source data displayed is correct. Lastly, users can offer feedback in a submission form to provide further clarification.

To access this feature, at the end of each publication entry is a “Submit Feedback” button. Upon clicking this button, users will be redirected to the submission page where they can verify if the publication “Contains a Planetary Map” and confirm if the “Correct Source Data” is displayed. As shown in *Figure 11*, these two fields (dropdown menus) are *required*. Users can also provide further feedback in the “Comment” section as an *optional* field.

Use Case:

1. The user lands on MapONE’s starting login page.
2. The user selects to continue as a guest (without an account).
3. The user is then directed to the main page. The user can view planetary map source data for a variety of publications.
4. The user selects “Submit Feedback” for a specific publication.
5. The user selects values for “Contains a Planetary Map” and “Correct Source Data.”
6. The user selects “Submit” and is directed to the confirmation page with an option to return to the main page.

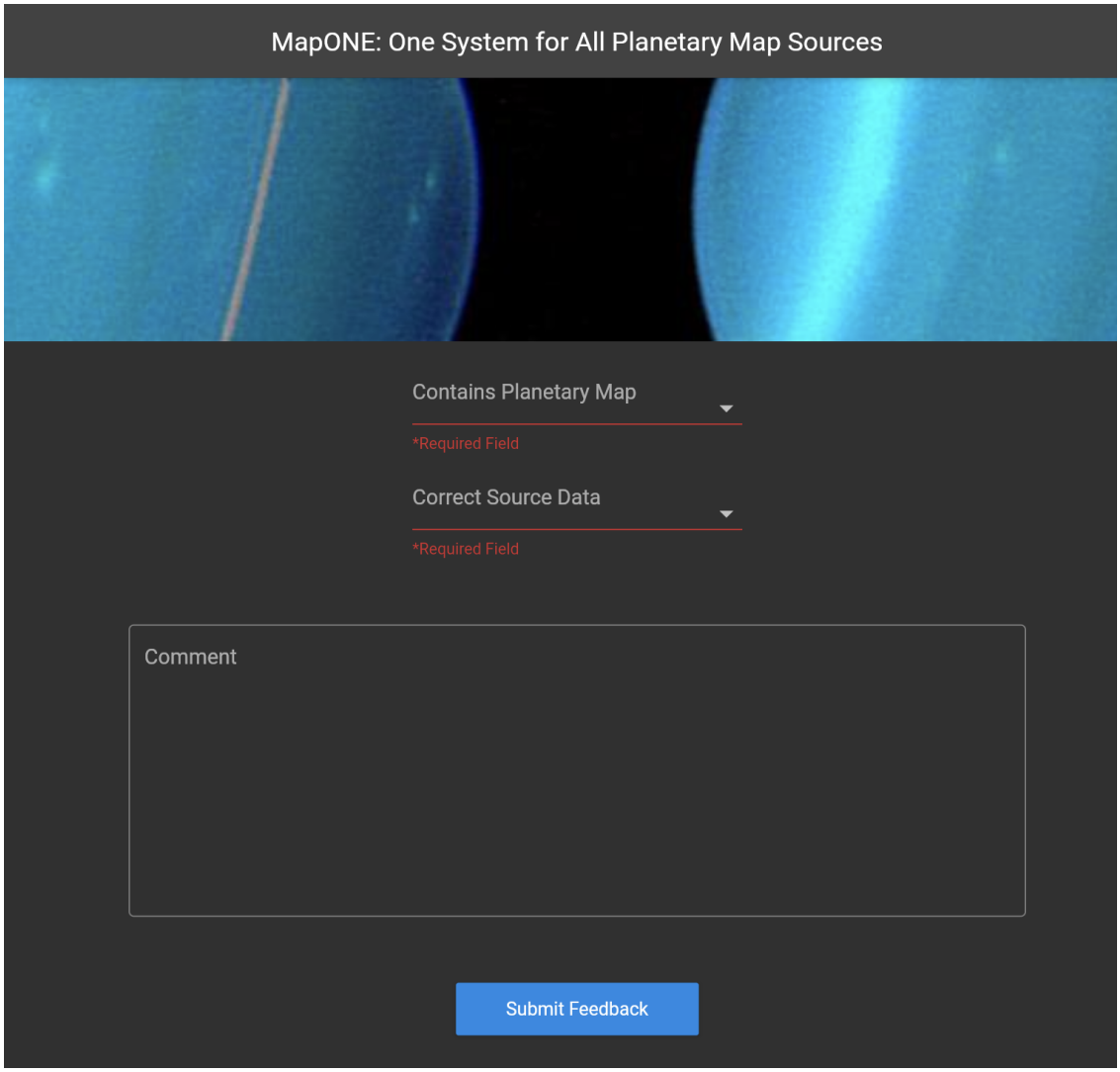


Figure 11. MapONE's feedback submission page.

User Profile

As previously mentioned, new and existing users can access the user profile (see "Taskbar" for more details). Here, users can enter a keyword and frequency to create a new automated save search. For future work, users should be able to view all searches and results under the user profile.

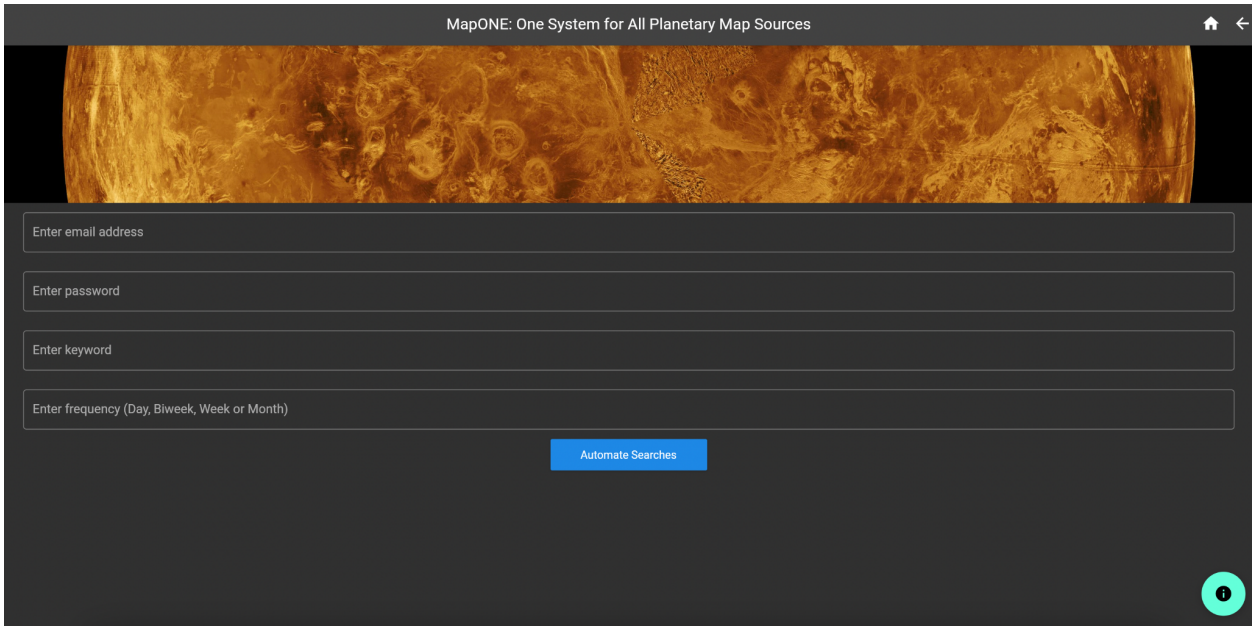


Figure 12. Create an automated search under the user profile.

3.2 Backend APIs

MapONE's frontend server uses API calls to request information stored in and operated by the backend. However, the backend server can be operated separately. The backend is divided into three main APIs - user, entry, and archive - which are accessed by the interface. The system also uses two additional APIs - web scraper and timer - to operate the automated web scraper (stored in the backend).

User API

The user API is responsible for accessing all user account information. This allows the system to directly communicate with the interface at the login and user profile levels. The following actions can be requested of the backend at https://mapone-api.herokuapp.com/user/:

1. **CREATE_USER**: Creates a new user account. The necessary parameters are *action*, *email_address*, and *password*. Action must be set to 0. The *email_address* must be a valid address. The *password* must be eight characters long and contains at least one number and one special character.

2. **LOGIN**: Logins and verifies existing users. The necessary parameters are *action*, *email_address*, and *password*. Action must be set to 1. The *email_address* and *password* must match the credentials of an existing user.
3. **CHANGE_PASSWORD**: Updates the password of an existing user account. The necessary parameters are *action*, *email_address*, *password*, and *new_password*. Action must be set to 2. The *email_address* and *password* must match the credentials of an existing user. The *new_password* must be eight characters long and contains at least one number and one special character.
4. **DELETE_USER**: Deletes an existing user account. The necessary parameters are *action*, *email_address*, and *password*. Action must be set to 3. The *email_address* and *password* must match the credentials of an existing user.

Entry API

The entry API is responsible for accessing all publication and source data information. This allows the system to directly communicate with the interface at the search engine level. The following actions can be requested of the backend at https://mapone-api.herokuapp.com/entry/:

1. **MAIN_PAGE**: Displays all publications and source data in the database. The necessary parameter is *action*. Action must be set to 0.

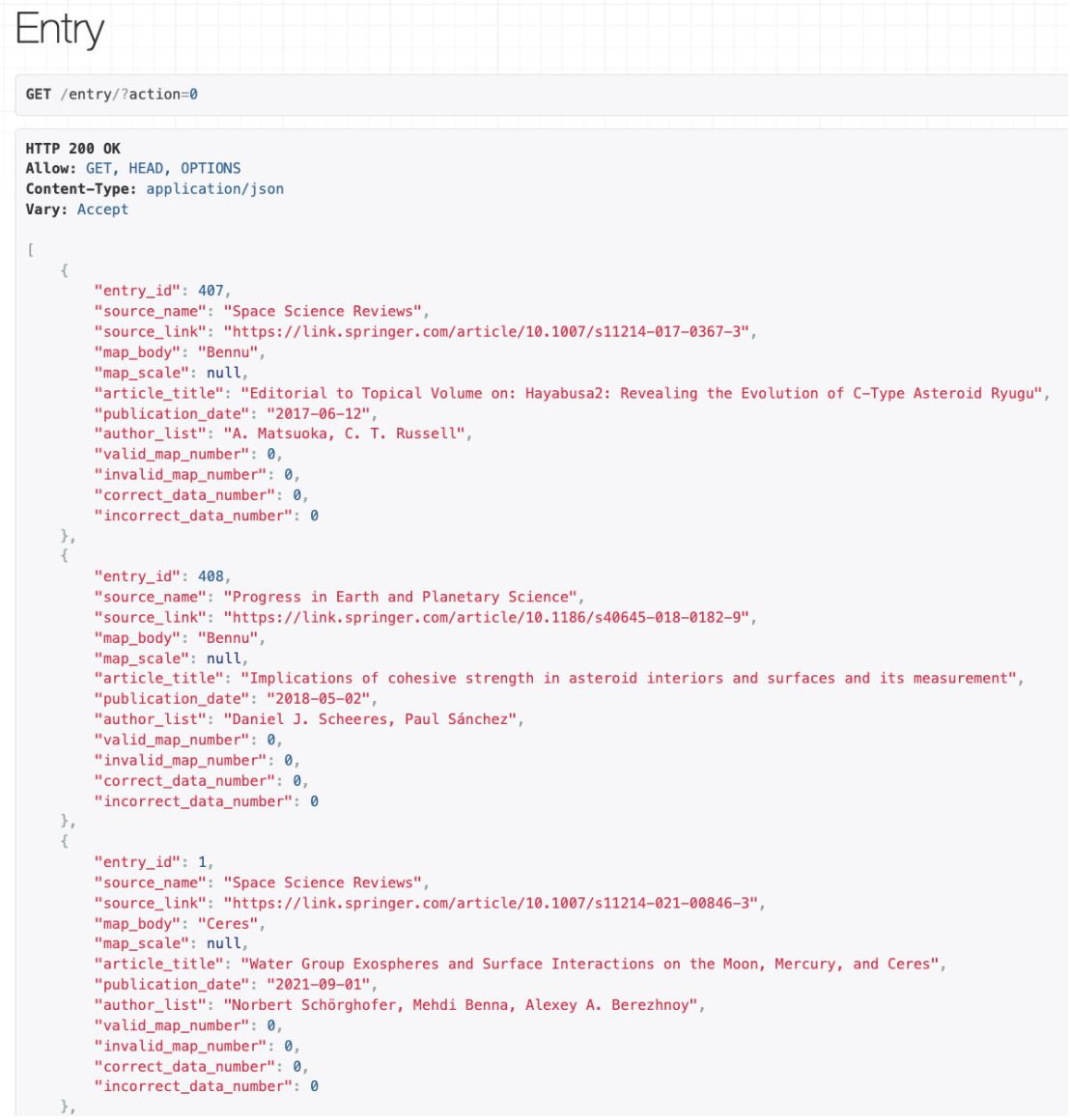


Figure 13. The entry API output after running the action MAIN_PAGE at https://mapone-api.herokuapp.com/entry/?action=0.

2. **SEARCH_KEYWORD**: Searches all publications and source data for a specific keyword (case insensitive). The necessary parameters are *action* and *keyword*. Action must be set to 1.
3. **FILTER_YEAR**: Finds all publications published within a certain year range. The necessary parameters are *action*, *first_year*, and *second_year*. Action must be set to 2. As a note, if the filter is for a one-year range, enter the same year for both *first_year* and *second_year*.

4. **LEAVE_FEEDBACK**: Stores user feedback into the backend database. The necessary parameters are *action*, *entry_id*, *validate_map*, and *validate_data*. Action must be set to 3. The *entry_id* is a publication's ID in the entry database (see "Admin View" for more details). The parameters *validate_map* and *validate_data* must be set to either "Yes" or "No" when submitted.

Archive API

The archive API is responsible for accessing all automated search information. This allows the system to directly communicate with the interface at the archive and user profile levels. The following actions can be requested of the backend at https://mapone-api.herokuapp.com/archive/:

1. **CREATE_ARCHIVE**: Creates a new automated search for an existing user. The necessary parameters are *action*, *email_address*, *password*, *keyword*, and *frequency*. Action must be set to 0. The *email_address* and *password* must match the credentials of an existing user. The *frequency* parameter must be one of the following: "Day," "Week," "Biweek," or "Month." The *keyword* must be unique to the user's saved searches.
2. **DISPLAY_USER_ARCHIVES**: Displays all saved automated searches under an existing user account. The necessary parameters are *action*, *email_address*, and *password*. Action must be set to 1. The *email_address* and *password* must match the credentials of an existing user.
3. **DELETE_ARCHIVE**: Deletes a user's automated search. The necessary parameters are *action*, *email_address*, *password*, and *keyword*. Action must be set to 2. The *email_address* and *password* must match the credentials of an existing user.
4. **UPDATE_FREQUENCY**: Updates the frequency of an existing automated search. The necessary parameters are *action*, *email_address*, *password*, *keyword*, and *new_frequency*. Action must be set to 3. The *email_address* and *password* must match the credentials of an existing user. The *new_frequency* parameter must be one of the following: "Day," "Week," "Biweek," or "Month."

Web Scraper API

The web scraper API can be accessed at https://mapone-api.herokuapp.com/web_scraper/. The web scraper ("mapone_backend_docker/mapone-api/mapone_api/web_scraper/scraperV2.py")

calls this API in realtime to add new scraped publications to the database. The API uses one action to verify and store publications. The necessary parameters are *api_key, source_name, source_link, article_title, publication_date, author_list, map_body,* and *map_scale.* The *api_key* parameter must be a valid key in the database to add a publication (see "Admin View" for more information). The *publication_date* must be in this format: YYYY-MM-DD. As a note, all publications listed in CSV files (see *Figure 2*) were previously loaded into the database during the initial configuration.

Timer API

MapONE's timer API starts the system's automated search and web scraper internal timers. The API calls the "run_timer" command listed in the "mapone_backend_docker/mapone-api/mapone_api/management/commands/run_timer.py" file (see section 2.5 step 20 for initial configuration). Entering https://mapone-api.herokuapp.com/timer/ into any browser starts the daily automated search scheduler that will check all users' automated searches and notify users (via email) if new publications (matching their saved keywords and based on set frequency) are added. Additionally, this action starts the web scraper's schedule to collect and store new planetary map publications in the database each week (see section 3.3 for more information).

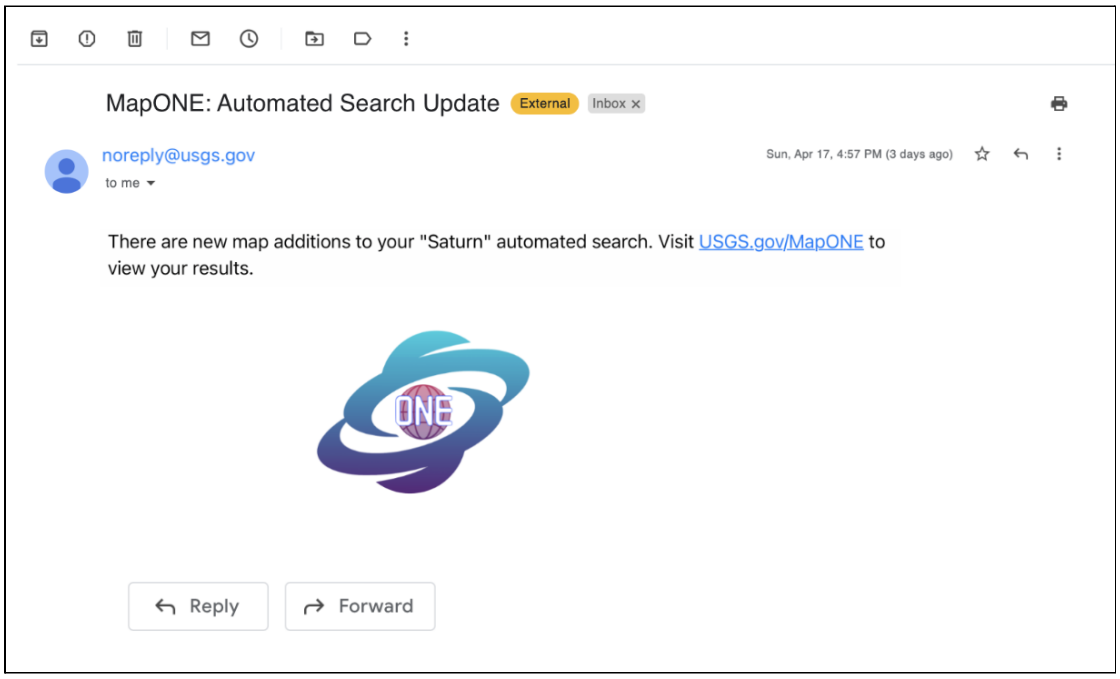


Figure 14. Users receive email notifications when new map additions are added.

Error Messages

When making API requests to the backend, the system will generate HyperText Transfer Protocol (HTTP) responses for each action. The following are all of the system's main response messages for HTTP 200-299 status codes:

1. **SUCCESS**: The operation was successful. Displays "operation success" as the message.

*Figure 15. Shows success message for creating a new user (*https://mapone-api.herokuapp.com/user/?action=0&email_address=sam@gmail.com&password=password1?).

2. **INVALID_EMAIL**: The operation was unsuccessful due to an invalid email address (either not listed in the email server or incorrect format).
3. **INVALID_PASSWORD**: The operation was unsuccessful due to an invalid password (either an incorrect password for a given user or the incorrect format).
4. **EMAIL_IN_USE**: The operation was unsuccessful since the email address is already in use for an existing user.
5. **KEYWORD_USED**: The operation was unsuccessful since the keyword is used for an existing automated search under a user account.
6. **INCORRECT_FREQUENCY**: The operation was unsuccessful since the keyword was not set to one of the following: "Day," "Week," "Biweek," or "Month."

For all other responses (missing parameters or generally incorrect requests), the server will display an HTTP 400 Bad Request.

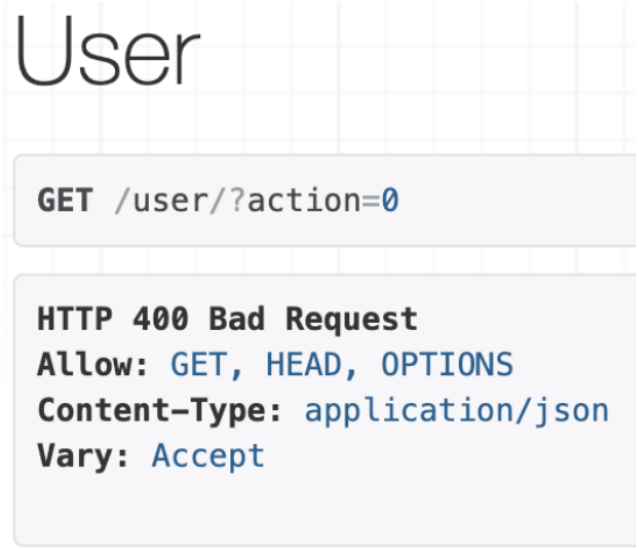


Figure 16. Shows HTTP 400 error for missing parameters when creating a new user (https://mapone-api.herokuapp.com/user/?action=0).

All error messages and declared actions are listed at "mapone_backend_docker/mapone-api/mapone_api/constants.py."

Database Admin View

The client can view MapONE's database at https://mapone-api.herokuapp.com/admin/. At the time of product delivery, the admin user's username and password are set to "123" on the system. Once logged in, the client can change or create new admin users at https://mapone-api.herokuapp.com/admin/auth/user/. The admin can access all saved data for the Archive, Entry, Feedback, Key, and User database tables. These tables store all data related to the backend APIs.

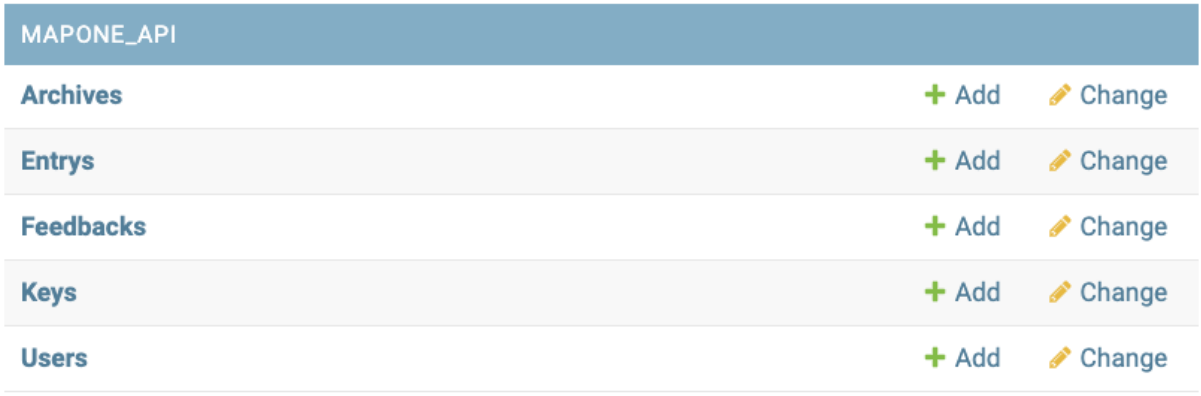
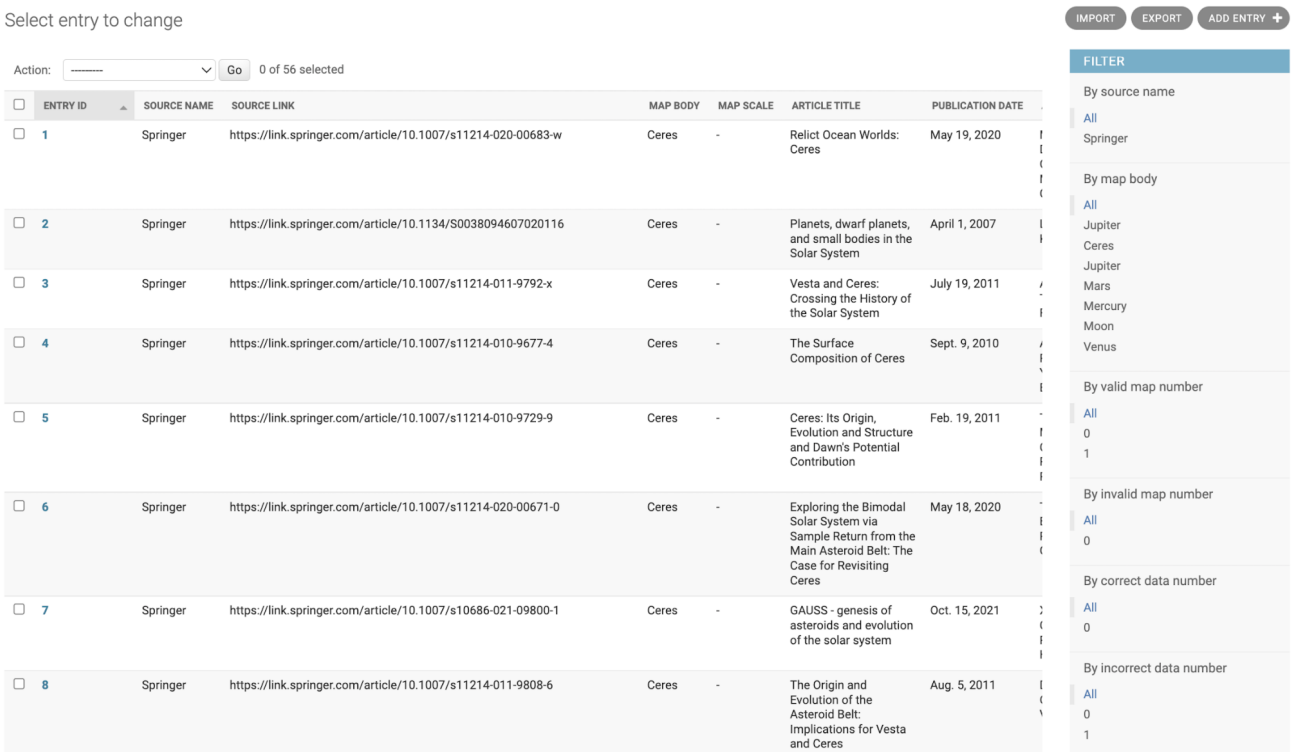


Figure 17. Admin view of all database tables.

Specifically, Feedback stores write-in submissions, and Entry stores publication source data (entry_id, source_name, source_link, map_body, map_scale, article_title, author, and publication_date) and the following additional fields:

- **valid_map_number**: Entry field for how many users have confirmed publication contains a valid planetary map.
- **invalid_map_number**: Entry field for how many users have confirmed publication does not contain a valid planetary map
- **correct_data_number**: Entry field for how many users have confirmed publication source data is correct.
- **incorrect_data_number**: Entry field for how many users have confirmed publication source data is incorrect

Admin users can use these metrics to see what users think of each publication and to check if the web scraper is extracting valid and useful information for the planetary science community. As shown in *Figure 18*, users can import and export this data and filter the database table based on source name, map body, and feedback metrics. Users can use this for further external analysis.



ENTRY ID	SOURCE NAME	SOURCE LINK	MAP BODY	MAP SCALE	ARTICLE TITLE	PUBLICATION DATE
1	Springer	https://link.springer.com/article/10.1007/s11214-020-00683-w	Ceres	-	Relict Ocean Worlds: Ceres	May 19, 2020
2	Springer	https://link.springer.com/article/10.1134/S0038094607020116	Ceres	-	Planets, dwarf planets, and small bodies in the Solar System	April 1, 2007
3	Springer	https://link.springer.com/article/10.1007/s11214-011-9792-x	Ceres	-	Vesta and Ceres: Crossing the History of the Solar System	July 19, 2011
4	Springer	https://link.springer.com/article/10.1007/s11214-010-9677-4	Ceres	-	The Surface Composition of Ceres	Sept. 9, 2010
5	Springer	https://link.springer.com/article/10.1007/s11214-010-9729-9	Ceres	-	Ceres: Its Origin, Evolution and Structure and Dawn's Potential Contribution	Feb. 19, 2011
6	Springer	https://link.springer.com/article/10.1007/s11214-020-00671-0	Ceres	-	Exploring the Bimodal Solar System via Sample Return from the Main Asteroid Belt: The Case for Revisiting Ceres	May 18, 2020
7	Springer	https://link.springer.com/article/10.1007/s10686-021-09800-1	Ceres	-	GAUSS - genesis of asteroids and evolution of the solar system	Oct. 15, 2021
8	Springer	https://link.springer.com/article/10.1007/s11214-011-9808-6	Ceres	-	The Origin and Evolution of the Asteroid Belt: Implications for Vesta and Ceres	Aug. 5, 2011

Figure 18. Entry database table. Shows ability to view, filter, and export source data.

3.3 Web Scraper

As noted in section 3.1, the web scraper runs automatically based on an internal timer. This means there is no necessary daily operation of the tool. All new publications are scraped and added to the database on a weekly basis. However, for future use or modifications, all functions related to the web scraper can be found in *scraperV2.py* and *util_functions.py* in the "mapone_backend_docker/mapone-api/mapone_api/web_scraper/" directory. The main script, *scraperV2.py*, hosts multiple functions for locating and extracting metadata from each planetary map publication. The *util_functions.py* serves as a helper file for the main script.

Main Script

At the end of the *scraperV2.py*, there is a *main()* function that is used to run and configure the entire web scraper. Here, the user can modify the list of keywords they wish to search for. Currently, the following map bodies are used as keywords: Ceres, Vesta, Earth's Moon, Mars, Jupiter, Pluto, Mercury, Venus, Titan, Enceladus, Bennu, and Charon. As a note, more specific searches require that each keyword is concatenated with a "+". Each keyword in the list is

concatenated with a standard search tag which acts as another layer of specification when searching for articles related to planetology.

```
# Here, we call the main function that runs the scraper with specified keywords
def main():
    #keyword_list = ["Ceres", "Vesta", "Moon", "Mars", "Jupiter",
    #"Jupiter+Io", "Jupiter+Europa", "Jupiter+Ganymede", "Pluto",
    #"Mercury", "Venus", "Titan", "Enceladus", "Charon", "Bennu"]

    # Abstracts containing these keywords will be scraped from Springer
    keyword_list = ["Ceres", "Vesta", "Moon", "Mars", "Jupiter", "Pluto",
    "Mercury", "Venus", "Titan", "Enceladus", "Bennu", "Charon"]

    for keyword in keyword_list:
        main.body = keyword
        '''MODIFY KEYWORD SEARCH HERE'''
        keywords_to_search = keyword + "+planetary+science"

        # References a separate LOG folder that holds abstract names and status of each one
        abstracts_log_name, status_logger_name = pre_processing(keywords_to_search)

        # Calling the scraper_main() to start the scraping process
        scraper_main(keywords_to_search, abstracts_log_name, status_logger_name)
```

Figure 19. Main function within scraperV2.py where keyword_list and keywords_to_search are modified.

If new keywords are added to the list in *main()*, they must also be added to the keyword list in *abstract_database_writer()*. This list acts as a dictionary for the database to determine what planetary body is currently being searched for so it can be written into the database.

```python
def abstract_database_writer(abstract_page_url, title, author, abstract, abstracts_log_name, abstract_date, status_logger_name):
    # Makes text/csv files to contain the abstracts for future reference
    # It holds: 1) Title, 2) Author(s), 3) Date(s), 4) Scale(s), 5) URL(s)
    abstract_database_writer_start_status_key = "Writing " + title + " by " + author + " to disc"
    status_logger(status_logger_name, abstract_database_writer_start_status_key)

    keyword_list = ["Ceres", "Vesta", "Moon", "Mars", "Jupiter", "Pluto",
    "Mercury", "Venus", "Titan", "Enceladus", "Bennu", "Charon"]

    for keyword in keyword_list:
        if keyword == main.body:
            body = keyword

    # call web scraper API
    url = 'https://mapone-api.herokuapp.com/web-scraper/'
    params = {
        'api_key': 1,
        'source_name': abstract,
        'source_link': abstract_page_url,
        'article_title': title,
        'publication_date': abstract_date,
        'author_list': author,
        'map_body': body,
        'map_scale': None
    }
    requests.get(url=url, params=params)

    abstract = abstract.replace('\'', '\"')
    title = title.replace('\'', '\"').replace('\xe2\x80\x99', '\'')

    abstracts_csv_log = open(abstracts_log_name + '.csv', 'a')
    abstracts_txt_log = open(abstracts_log_name + '.txt', 'a')
    abstracts_csv_log.write(abstract)
    abstracts_csv_log.write("," + abstract_page_url)
    abstracts_csv_log.write("," + title)
    abstracts_csv_log.write("," + abstract_date)
    abstracts_csv_log.write(",\"" + author + "\"")
    abstracts_csv_log.write("," + body)
    abstracts_csv_log.write(",")
    abstracts_csv_log.write('\n')
    abstracts_txt_log.close()
    abstracts_csv_log.close()

    abstract_database_writer_stop_status_key = "Written " + title + " to disc"
    status_logger(status_logger_name, abstract_database_writer_stop_status_key)
```

Figure 20. Function to write and format all scraped metadata to a database CSV file. This function also calls on the web scraper API to verify and store new publications (see section 3.2 for more details.)

Utility Functions

Utilities to log the output of the web scraper, as well as tools to help reduce the workload on scraper-specific functions, can be found in *util_functions.py*. Each time the web scraper is executed, a new session folder is created with the help of *pre_processing()* and displayed using *status_logger()*. Not only does this containerize the files, but it allows the user to see what is happening during execution. The final function, *end_process()*, is used to notify the user of session completion.

For general use, this script does not need to be changed. However, users can modify the session folder names within *pre_processing()* if desired. The variable *logs_folder_name* specifies the name of the LOGS folder containing each session folder. *Abstracts_log_name* specifies the name of the database file within each session folder, while *status_logger_name* specifies the name of the status file for each session. By default, these are assigned to the time at which they were created.

```
# Declaration of the LOG folder and the abstract, abstract_id & status_logger files
logs_folder_name = "LOGS" + "/" + "LOG" + "_" + run_start_date + "_" + run_start_hour + "_" + run_start_minute + "_" + folder_attachement
abstracts_log_name = logs_folder_name + "/" + "Abstract_Database" + "_" + run_start_date + "_" + run_start_hour + "_" + run_start_minute
status_logger_name = logs_folder_name + "/" + "Status_Logger" + "_" + run_start_date + "_" + run_start_hour + "_" + run_start_minute
```

Figure 21. Declaration of filenames generated by the web scraper can be modified if desired.

4. Maintenance & Testing

At this point in the manual, the client should know how to install, configure, and operate all MapONE features. This section outlines the testing and potential maintenance necessary to properly run and operate MapONE in the future.

4.1 GUI

Unspecified errors are common in Flutter application development, specifically with compiler errors. To ensure the development environment is properly running, **change** into the "mapone_frontend" directory and **run** *Flutter doctor* in the command line. As shown in *Figure 22*, in this example, the Android toolchain was not properly set up. Thus, the debug command recognized and displayed this error. This tool allows users to quickly identify any issues with a Flutter application. For all additional issues or questions, please visit Flutter at https://flutter.dev/.

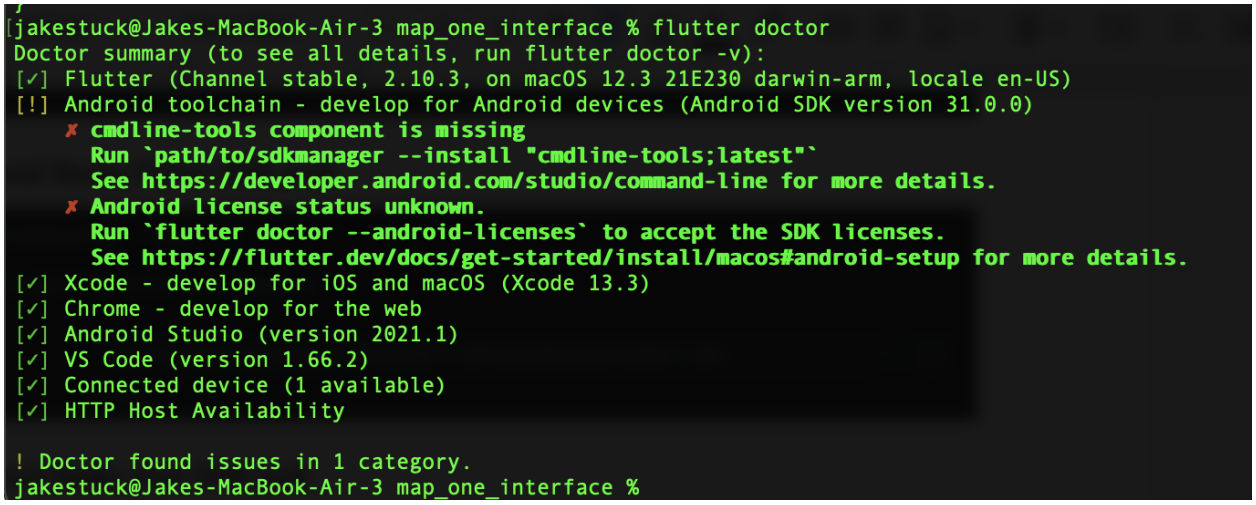


Figure 22. Run Flutter doctor to ensure correct Flutter functionality.

4.2 Backend APIs

All backend testing is outlined in "mapone_backend_docker/mapone-api/tests.py." Here, the client can view the following Django test cases:

1. **WebScraperTestCase**: Tests the web scraper main script to ensure publications are being scraped and stored correctly in the database.
2. **APITestCase**: Tests all actions listed in section 3.2 for the user, entry, and archive APIs to ensure APIs are correctly completing calls.
3. **UserTestCase**: Tests all class functions in the *user.py* file (in the same directory as *tests.py*).
4. **EntryTestCase**: Tests all class functions in the *entry.py* file (in the same directory as *tests.py*).
5. **ArchiveTestCase**: Tests all class functions in the *archive.py* file (in the same directory as *tests.py*).

For more information on how Django testing works, please visit https://docs.djangoproject.com/en/4.0/topics/testing/overview/.

To run all test cases, the client can do the following steps:

1. **Change** into the backend directory, “mapone_backend_docker/mapone-api/” using a command-line tool.
2. **Run** *pip install -r requirements.txt* to install all necessary packages. For this step, it is recommended the client use a virtual environment. For more information, visit https://virtualenv.pypa.io/en/stable/.
3. **Enter** a valid email address for test variables in *tests.py.* Otherwise, all tests that require an *email_address* or *test_email* will fail.
4. **Run** *python3 manage.py test mapone_api* to run all tests.
5. **Run** *python3 manage.py test mapone_api.tests.WebScraperTestCase* to run the individual Web Scraper test case.
6. **Repeat** step 5 for all other TestCases by replacing the *WebScraperTestCase* with the desired test case name.

For all other inquiries or overall maintenance, please refer to Django at https://www.djangoproject.com/ for any updates or dependencies.

4.3 Web Scraper

For the sake of modularity, the web scraper is capable of running independently from the rest of the system. As mentioned in section 4.2 step 5, running *python3 manage.py test mapone_api.tests.WebScraperTestCase* will run the main function of the web scraper file, *scraperV2.py.* This allows the client to adjust the web scraper while still running the backend tests in the same directory/code material. Before running the web scraper, be sure to comment out the web scraper API requests shown in *Figure 20*. Otherwise, all scraped publications will be added to the database (not recommended during testing). As a note, it is recommended that the default version of the web scraper be saved before making any changes or adjustments.

When running the main function, the web scraper should generate a LOG folder containing a session folder with one database file and a status logger file. This process may take a few minutes for keywords with multiple page URLs, but the status logger will display a session completion message on the command line.

To test individual keywords, the main script can be modified to only search for specific map bodies or keywords. This may be useful for locating publications on map bodies not already declared in the keyword list.

```
# Here, we call the main function that runs the scraper with specified keywords
def main():
    '''MODIFY KEYWORD SEARCH HERE'''
    keywords_to_search = "Mars+planetary+science"

    # References a separate LOG folder that holds abstract names and status of each one
    abstracts_log_name, status_logger_name = pre_processing(keywords_to_search)

    # Calling the scraper_main() to start the scraping process
    scraper_main(keywords_to_search, abstracts_log_name, status_logger_name)
```

Figure 23. Modified main function used to test individual keywords.

Depending on the keywords provided, and their level of specificity, the web scraper may return "IndexError: list index out of range". Searches with strict parameters may not return any resulting publications, or the publications found do not contain planetary maps. If this error occurs, it is suggested to:

- Broaden the search query for that specific keyword.
- Check the results on Springer (https://link.springer.com/search) directly to ensure results (if any) are found for that specific keyword.
- Append the keyword search to the end of *keyword_list* in the *main()* function.

The last suggestion, although not ideal, allows the web scraper to extract results for all other keywords in the list before encountering the error. This may be done if the user wishes to save keywords for future searches when more results become available. The error-inducing keyword, so long as it is placed at the end of the list, will not affect the rest of the system.

5. Conclusion

This user manual discussed the access, installation, configuration, operation, and maintenance of MapONE. The client should now have access to all source material and accounts as well as know

the internal and external operations of the system. For additional information, please refer to the code material or discussed installed software: GitHub (https://github.com/), Heroku (https://www.heroku.com/), Docker (https://www.docker.com/), Flutter (https://flutter.dev/), and Django (https://www.djangoproject.com/). For any additional questions or concerns, please contact MapONE's team leader, Samantha Milligan, at smm885@nau.edu.