COSC 310 M3 Submission

Testing

Current testing approaches:

- We are implementing mandatory unit tests of major functionality of all code written, using python unittest.
- We are using fake data hardcoded into the tests to maintain reproducibility, such as test JSON files, and test user information.
- We will test cross-functionality, ensuring that all of the individual pieces of code will perform harmoniously with each other. For example, while the class for adding to the database will have its own unit tests, we will also have a test that combines the database with the login form to test that it is properly interacting with the addUser method.
- Regularly opening the entire application and manually running through the implemented features. (at least on every scrum meeting, and individually as well).
- Perform regular visual testing of site layout (we load the application and test different screen sizes, ensuring that all of the components are behaving and scaling properly).

Future testing approaches:

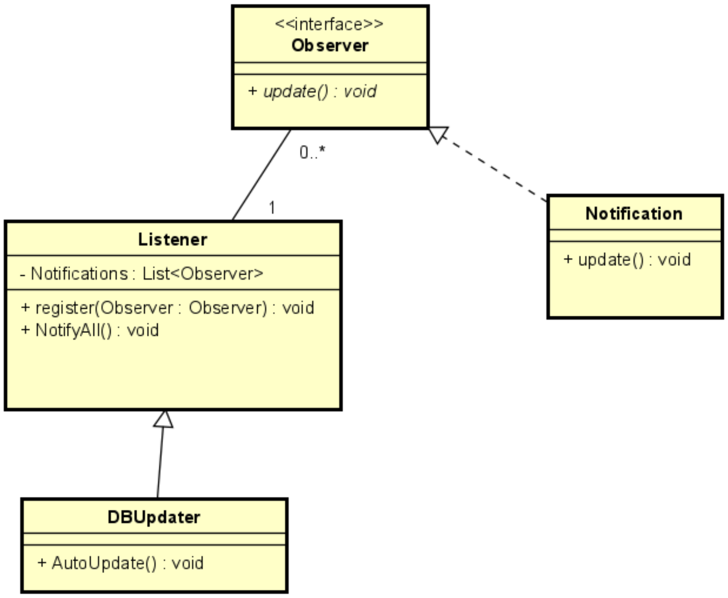
- Will begin to use the Selenium (https://www.selenium.dev/) test framework to test the front-end features.
- Will use pytest (https://flask.palletsprojects.com/en/3.0.x/testing/) to test our flask application, ensuring endpoints are properly configured, form data is passed correctly, JSON data handled correctly, and routing is behaving properly.
- We will set up and start to use github's CI (https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration) to continually run the tests, ensuring that when new code is added via pull requests, that all of the tests still pass.
- We will explore the usage of PyAutoGUI (https://pyautogui.readthedocs.io/en/latest/) to help test components of our webpage that require emulation of user behavior (such as clicking, scroll, hovering, key-presses, etc).

Design Patterns:

Observer Pattern for notifications:

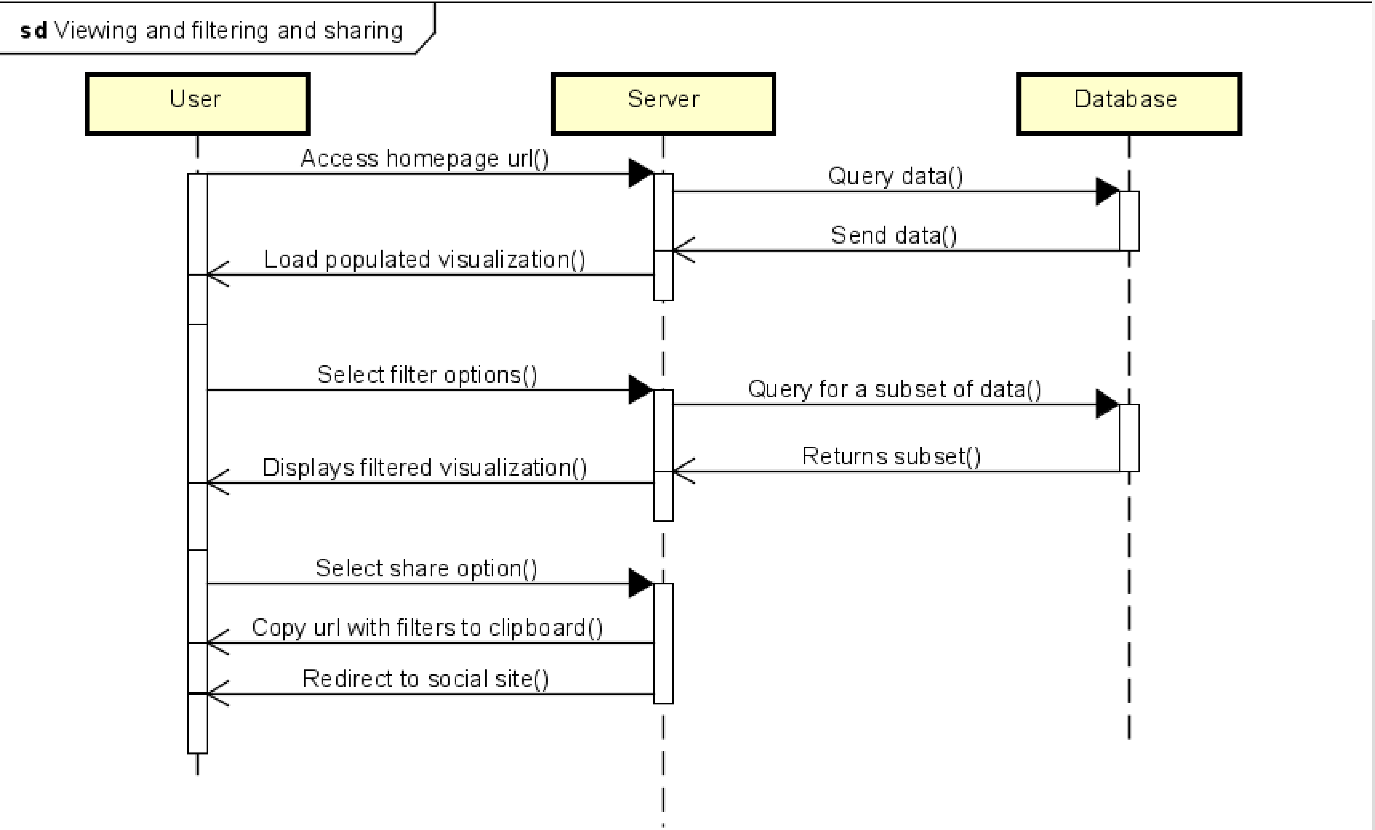
By implementing the behavioral observer design pattern, it will allow us to have a central control to send notifications. Each time the data from the API gets refreshed, we will call a notify method for a class that will handle notifying all notification registrants. The class will be able to classify the new data and determine if there are any registreants that need to be notified. Then, notifications will be sent via email or text (whatever the users preference is).

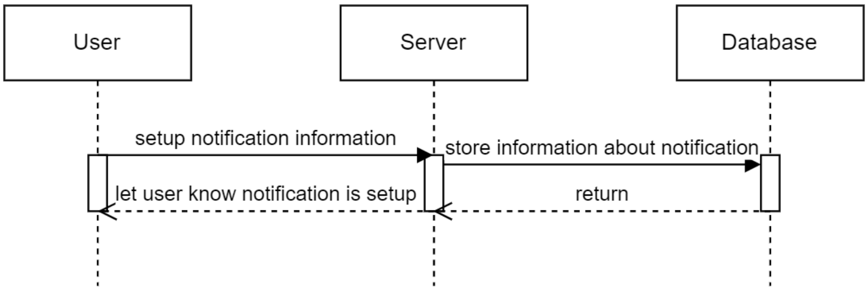
In the following class diagram, DBUpdater auto updates on an interval, storing the new data in the database. When it does this the listener calls notifyall, and each notification object then pulls data from the database and sends notifications. The internal logic in each notification object will set which tuples it pulls from the database to send.

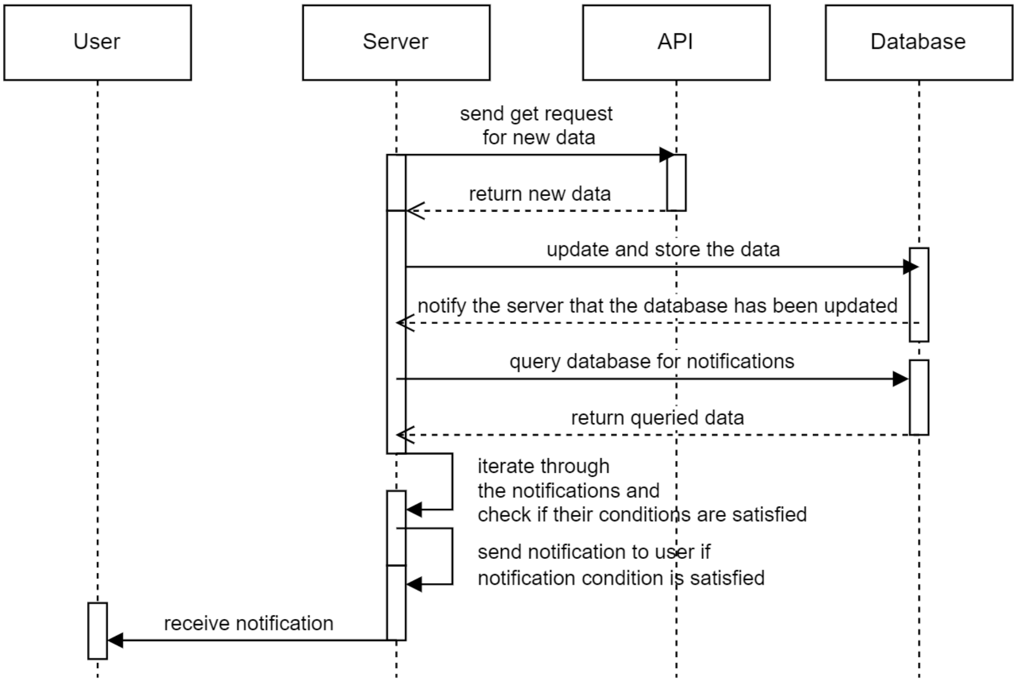


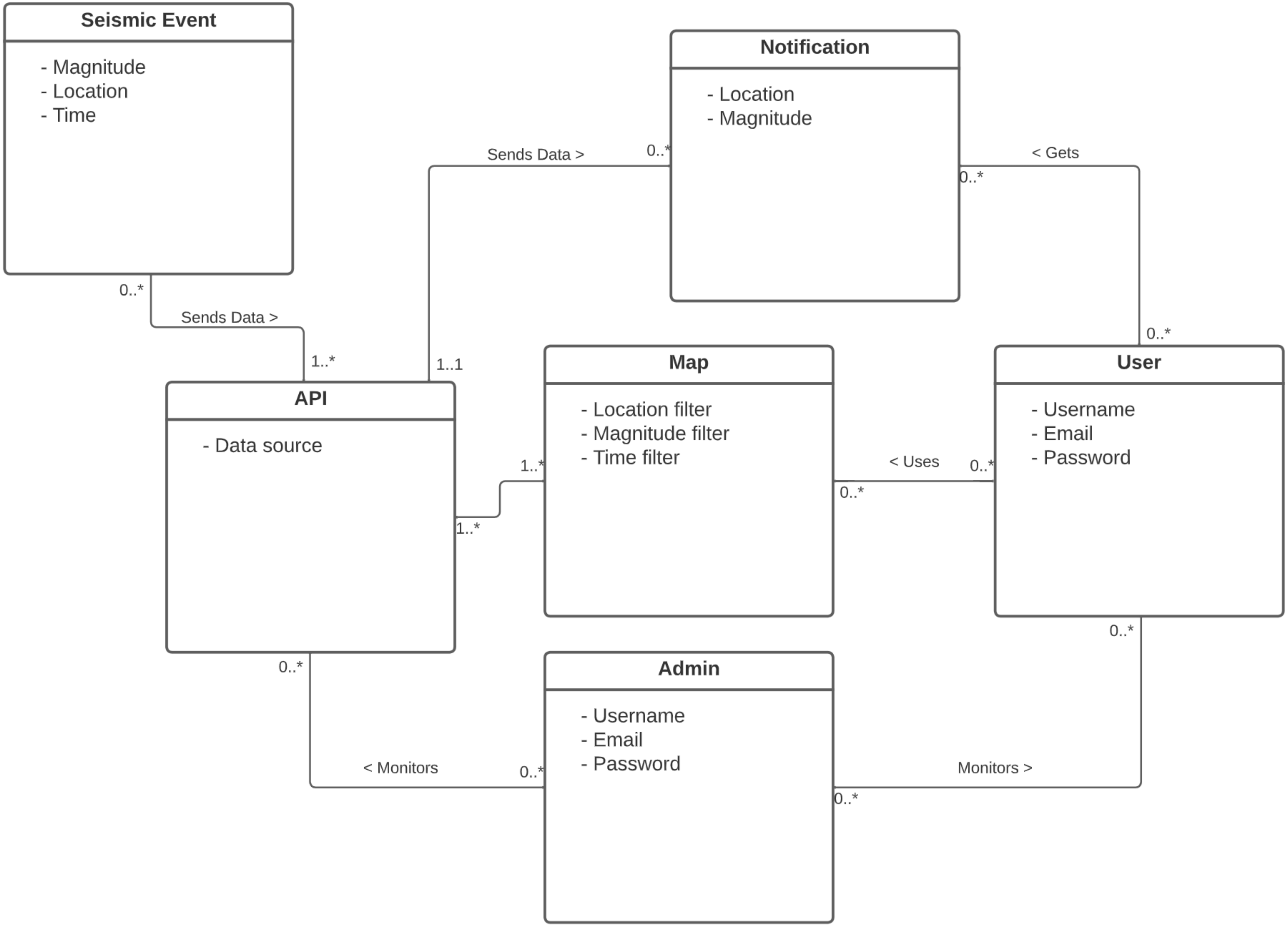
Facade Design Pattern for application control:

We will be using the structural facade design pattern. Our flask application will act as the interface and will facilitate communication between our subsystems. The subsystems include the database, the classes responsible for API data querying, database data insertion, database querying, notification system, prediction system, user authentication, and administration. The flask application will control all of these subsystems, providing a seamless and simple user experience to our intricate application.


sd Viewing and filtering and sharing
User
Server
Database
Access homepage url()
Query data()
Send data()
Load populated visualization()
Select filter options()
Query for a subset of data()
Returns subset()
Displays filtered visualization()
Select share option()
Copy url with filters to clipboard()
Redirect to social site()


User
Server
Database
setup notification information
store information about notification
let user know notification is setup
return


User
Server
API
Database
send get request
for new data
return new data
update and store the data
notify the server that the database has been updated
query database for notifications
return queried data
iterate through
the notifications and
check if their conditions are satisfied
send notification to user if
notification condition is satisfied
receive notification


Seismic Event
- Magnitude
- Location
- Time
Notification
- Location
- Magnitude
Sends Data >
0..*
< Gets
0..*
0..*
Sends Data >
1..*
1..1
0..*
API
- Data source
Map
- Location filter
- Magnitude filter
- Time filter
User
- Username
- Email
- Password
1..*
1..*
< Uses
0..*
0..*
0..*
0..*
Admin
- Username
- Email
- Password
< Monitors
0..*
Monitors >
0..*