



# MARINE MAMMAL STRANDING RESPONSE & DATA TRACKING – ARCHITECTURE

NOVEMBER 10, 2019

## **Group 27**

Brittany Abad

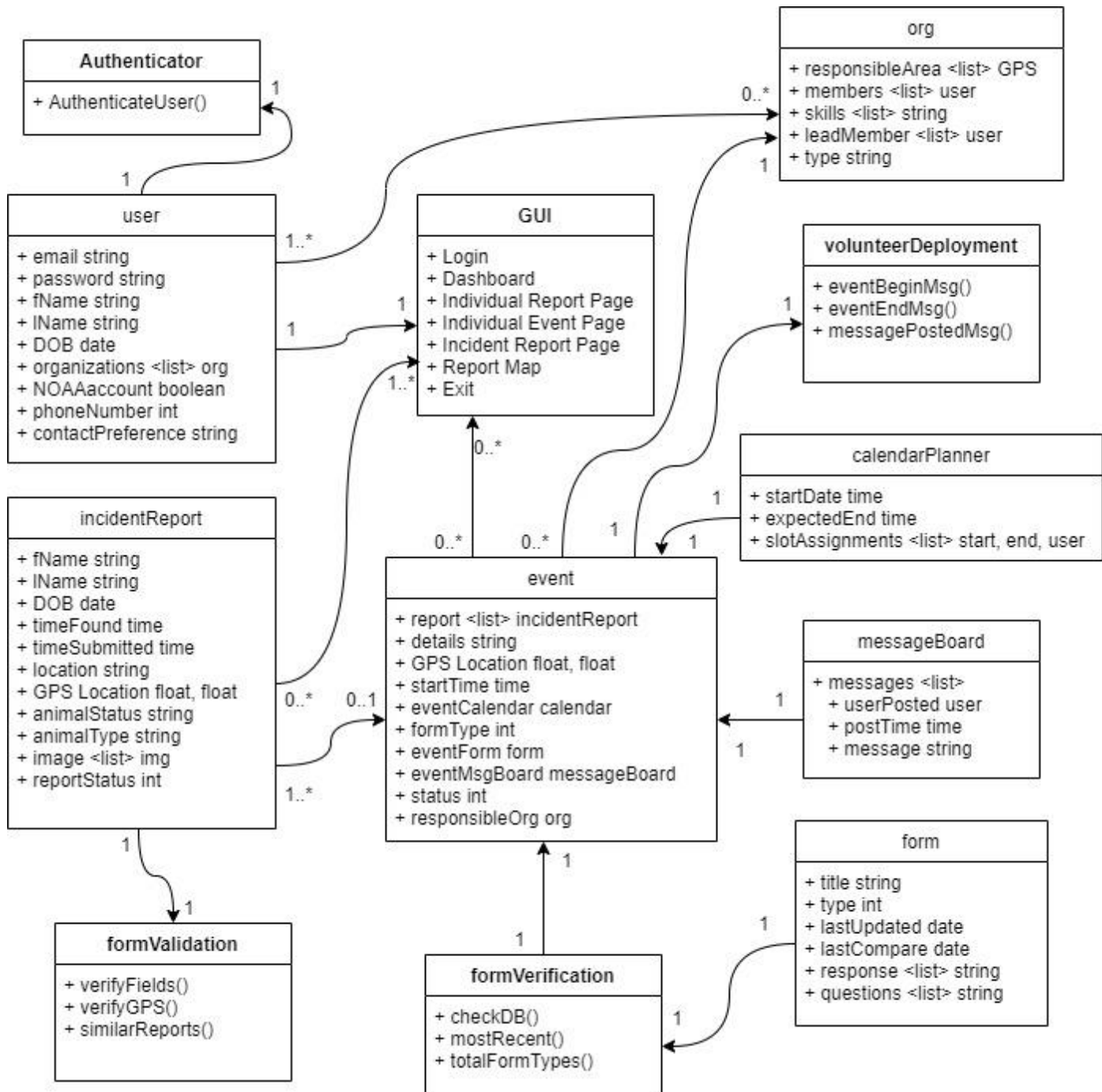
Lauren Boone

Christopher Feth

Manda Phadke

Kunal Patadia

# UML CLASS DIAGRAM





# PACKAGING THE IMPLEMENTATIONS

The packaging implementation described here is to encourage the maximum amount of cohesion and to reduce coupling within development (to increase maintainability). In honor of that, the implementation will be ordered in a way that systems with similar purposes and or use the same data will be implemented alongside each other.

The first round (package) of implementations would be the GUI and the event. The GUI is a central part of the app with all users depending on it so it must be developed before any user or organization functionality is implemented. The event is also important in this same way. The GUI and the event however do serve a similar purpose as they both are essentially the functionality of the app — though the GUI does depend on having the events to show and in this way the GUI and event have a bit of coupling between them. When back end features of an event are updated they may require a GUI update to accommodate those and this decreases maintainability. The volunteer deployment entity should be implemented in this round as well since it directly depends on the event entity. This does create coupling between the event and volunteer deployment entities but by the very nature of the deployment needing event details this must be the case. This does decrease maintainability.

Second, the user and organization entities and functionality will be implemented as they both make use of the same resources, both having many characteristics saved about them within the system. The organization is a form of a user and it would make sense to implement the user and organization alongside each other so that the same standard are followed. The organization only adds onto what the user has and in this way they are cohesive. The organization does depend on the user but updates to the user also update the organization automatically and thus this increases the maintainability of our system.

Third, the message board, calendar planner, incident report and form will be implemented together as they all interface with the GUI and event entities. Being that each of these are their own page within the app with minimal dependencies on other system entities they can easily be implemented alongside each other and have no coupling between them. This dependency obviously lends itself to a maintainable system.



The last round of implementations would be the authentication methods to verify users and forms. Since both user validations (authentication) and form validations make heavy use of databases and external systems (such as GPS validation through an online mapping system) it will be good to develop a standard method of communicating with databases and external systems across the board. Conceivably the teams implementing the two would work with each other to decide on a common method of interfacing with the databases and external servers so that this consistency is implemented. With this consistency, the maintainability of the system moving forward is much easier as when a change to the user validation is made, the same exact change can be made for the form validations without having to understand the in's and out's of each respective system.

## INCREMENTAL AND ITERATIVE DEVELOPMENT

Many of the system's entities inherent purpose supports iterative development such as the calendar planner, message board, incident report and form in the way that they do not depend on each other — though they all interface with the event and GUI. This dependency on the event and GUI does require that incremental development be done in the beginning to implement the GUI and then the event. In the same way that the planner, message board and others depend on the GUI and event being developed, the validation and verification methods rely on the user and organization being developed (which then depend on the GUI and event) — and thus the incremental development must be such that the GUI is implemented, then the event, then the users, then the organizations and then the validation and verification methods. From there iterative development can take place to implement the various other entities listed in the beginning. Once the whole system is developed, it's likely that iterative development would be all that needs to take place to update a wide range of elements within the entities. It's not likely that the GUI or event characteristics will change and thus not require any incremental development within maintaining the systems.

# DESIGN PATTERNS

## Observer:

This design pattern would be useful because our system is primarily driven by the users being notified about changes in the events. The event would be the subject of the system since the GUI would need to be updated every time new data is received about the event. The GUI would then alert the users through features such as our event map/list, calendar planner, message board, etc. Because our system depends on having regularly updated information, the observer design pattern could be helpful in watching for any changes and displaying the most current data.

## Facade:

This design pattern could be useful in our implementation because the users only need to interact with the higher-level interface, which would be the GUI. Since entities like the message board, calendar planner, etc. all interface with the GUI, the users do not have to access parts of the underlying complex system, such as creating the connections to the external databases.

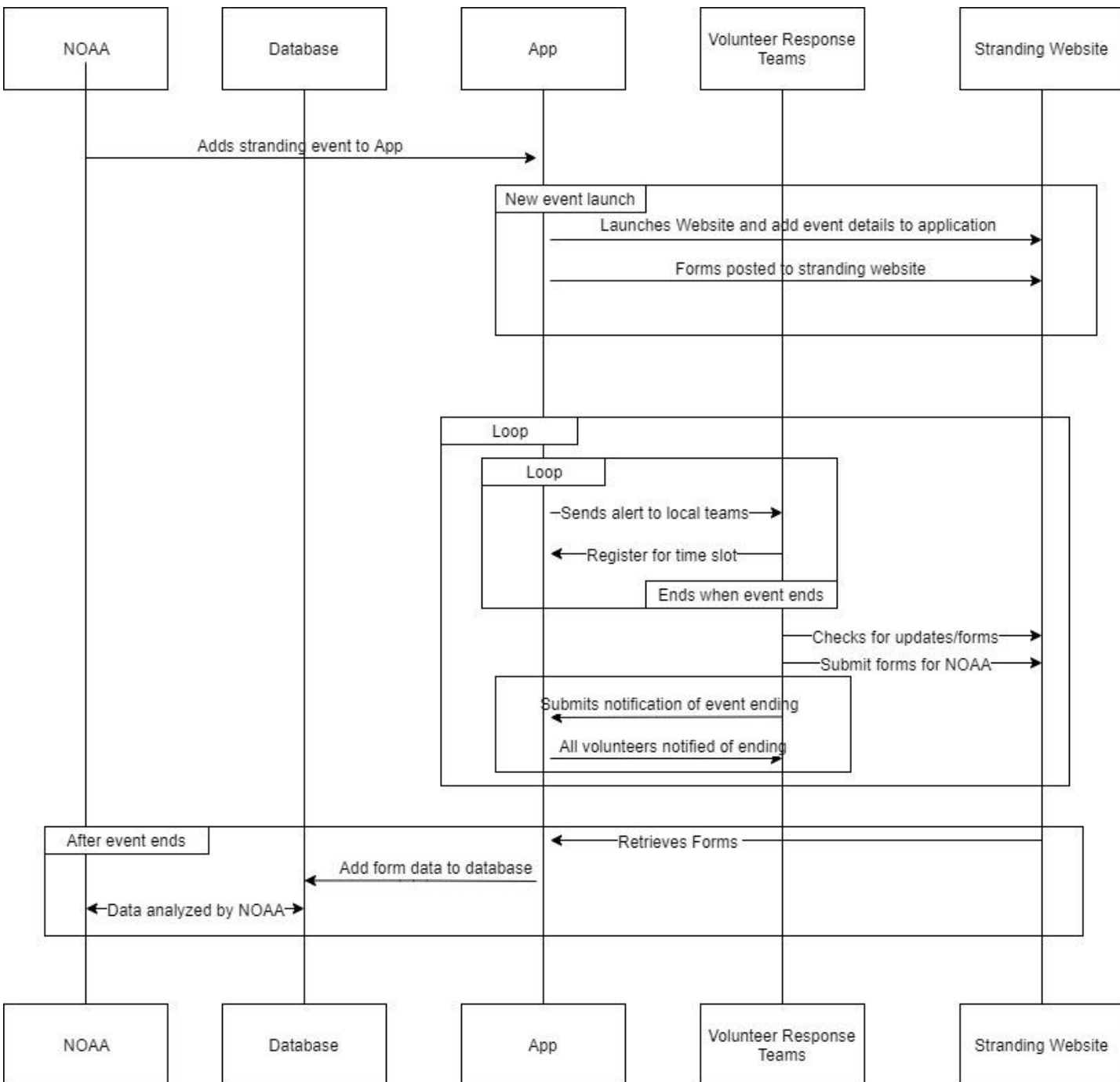
## Template Method:

The template method design pattern might be helpful with implementing the functionalities for the user and organization entities. The user entity could be considered the base class for the organization entity because the organization influences how a user interacts with the GUI, which means these two entities have the potential to have a lot of similar code.

## Decorator:

The decorator design pattern could be helpful in developing the different user roles for the system. Since our current design has 3 different user types (general public, responder, and NOAA), we could develop a basic user class and add/remove different behaviors for each one. This way, we could limit the general public user from accessing and/or altering features that the other users rely on for information.

# USE CASE SEQUENCE DIAGRAM




# INTERFACES

## •Smartphone Location Services

- Input: Uses the system to obtain the data of the user's current location.
- Output: Coordinates of the user to be used and recorded in the incident report forms for accurate mappings.

## •Inter package interfaces

- **GUI**
  - Input: Data and information from the event calendar, authenticator, incident report, message board, and volunteer deployment, majority of data that is represented as an output from all other packages will be displayed on the screen for the user to view.
  - Output: GUI, depending on the page that is in view as well.
- **Incident Report**
  - Input: Names of user, animal location, animal status, animal type, date of birth, time found, time submitted, gps location of user.
  - Output: Sent to government database, stranding database, and eventually to NOAA who pulls the forms for verification.
- **Authenticator**
  - Input: Username, password, incident forms, user location, forms validation, forms verifications.
  - Output: Requests made to servers to obtain the forms, user information such as name and login credentials, and user location.
- **Volunteer Deployment**
  - Input: Information from the event calendar that is stored, informs volunteers with the list of events on the calendar, information about when the events begin and end, posting messages.
  - Output: Calendar of the events on the GUI and volunteers/organizers messages.
- **Event Calendar**
  - Input: Schedules of events that are listed by NOAA to send to volunteers and organizers, the start/end date and time are submitted, list of events as well as event map that utilizes the gps location submitted with the report.

- 
- Output: Calendar on GUI application to view and edit schedules, as well as the date and times of the events.

- **Message Board**

- Input: Messages between users, the user information such as their name and organization they belong to, and the times the messages were sent.
- Output: Messages on the message board discussion, on the GUI.



# EXCEPTIONS

## •External System Exceptions

- If any of the users, whether from the general public, NOAA, volunteer services, tries to obtain or fill out the incident report forms will be met with a 400 error. If a connection isn't made to any of the servers there cannot be an output to show any of the information the user requests from the servers. An example would be if a user from the general public fills out a form and submits it, but has no internet connection, they should be notified that their form was not successfully submitted. If that's the case, the user should also be notified that they should either resubmit the form, or fill out a new one.

## •Login Exceptions

- Any user that has access to create an account will be required to login with the proper credentials. If the user doesn't have a registered account, the system will throw a 401 for an unauthorized request. If the user continues trying to login to their account but cannot, then the system should notify the user to either reset their password or to lock them out depending on if the username/email is valid.

## •Message Board System Exceptions

- The case where messages are not being sent or retrieved by both ends of the users. If the user tries to send a message that wasn't successfully sent, the system should notify the user to retry sending their message. Another instance would be if the user is trying to get into the message board page within the app. This is technically also part of the system exceptions since the page would be inaccessible. In this case the user would be notified to try again later to access the message board and to check their internet connection to make sure they're connected.

## •Events System Exception

- Similar to the message board system exception, the events systems exception would notify the user that the current or updated events are not able to load or download. If the user has some sort of weak connection that is delaying the updated events page, the user can still see the old events calendar. In this scenario, if the user is not able to download any events or gain access to the events calendar, the system should notify them to reconnect to the internet and if applicable, check their own local system settings to see if any permissions are being denied of service for the application.

# SUMMARY

## TEAM MEMBER CONTRIBUTIONS

- All – Team meeting via Google Hangouts on Tuesday, 11/5/19 and continued discussion throughout the week on Slack.
- Brittany Abad – Design Patterns, Assembled and reviewed HW4 final document
- Lauren Boone – Use Case Sequence Diagram
- Christopher Feth – Packaging the Implementations, Incremental/Iterative Development
- Manda Phadke – UML Class Diagram, Portfolio page
- Kunal Patadia – Interfaces, Exceptions