# Proximity

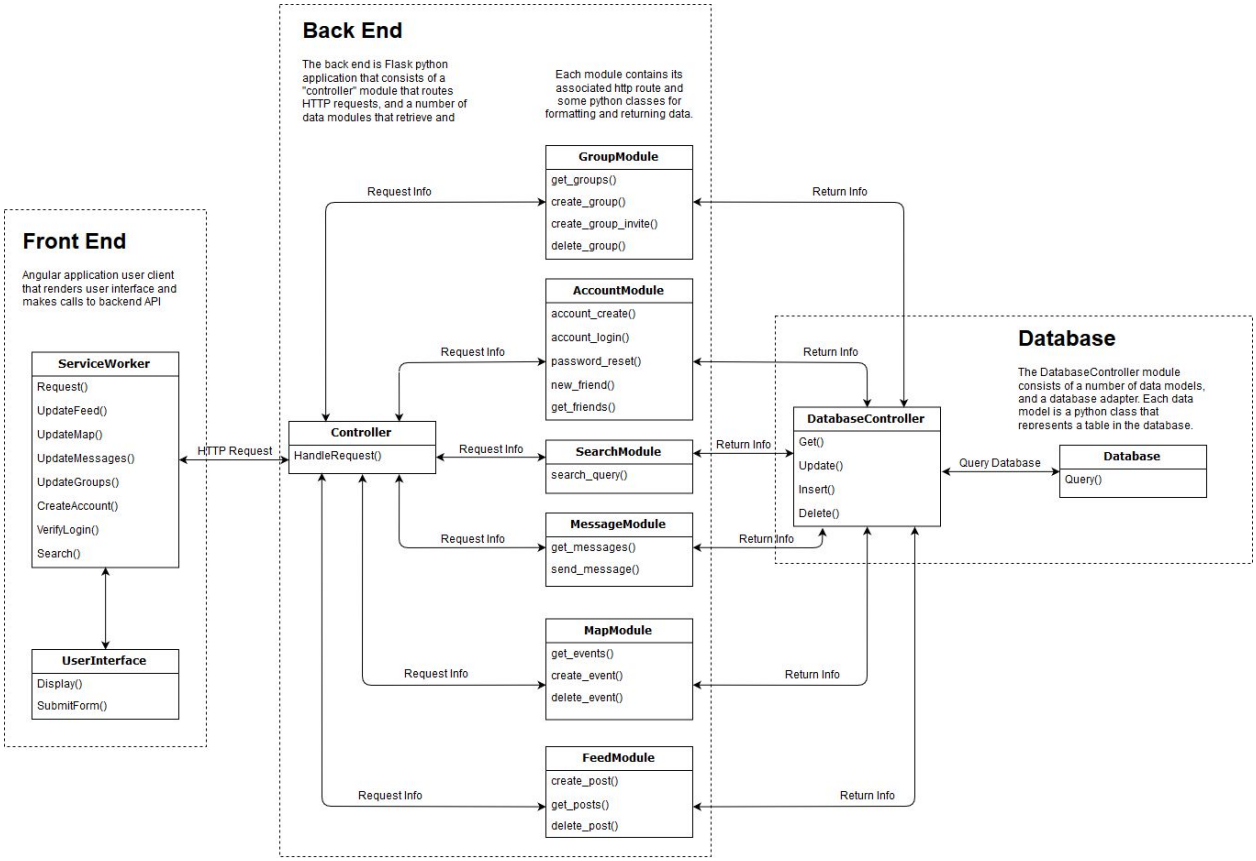
Technical Documentation

Group 16: Ryan Wortmann (Leader), Song Vu Nguyen, Ryan Rottmann, Nathan Kulczak, John Oatey

Group Website: https://github.com/Rdubya54/Software_Engineering_Project

# 6) Domain Analysis

## (a)Domain Model




In the diagram we can see the clear distinction between client (front end) and server (back end) in our application. This application follows the client-server design pattern.

Domain Model Diagram

- Concepts Definition

| Responsibility | Type | Concept |
|---|---|---|
| R1: Accepts and handles request from the front end the application. | D | Controller |
| R2: Fetches new information from the back end | D | ServiceWorker |

| | | |
|---|---|---|
| server and updates the user interface. | | |
| R3: Formats data received from the ServiceWorker and displays it to the user. Accepts input from the user and transmits it to the ServiceWorker. | D | UserInterface |
| R4: Stores and provides access to data | K | Database |
| R6: Sits between database and the rest of the application and provides an easy interface. | D | DatabaseController |
| R7: Fetches data from the posts table using the DatabaseController and formats it to be sent to the user to display the user's home feed.<br>Also uses the DatabaseController to add new post data to the database. | D | FeedModule |
| R8: Fetches a list of events to display on the map using the DatabaseController and formats it to be sent to the user to display the user's event map.<br>Also uses the DatabaseController to add new event data to the database. | D | MapModule |
| R9: Fetches a list of messages to display using the DatabaseController and formats them to be sent to the user to display the user's messages.<br>Also uses the DatabaseController to add new message data to the database. | D | MessagingModule |
| R10: Fetches group information from the DatabaseController and formats it to be sent to the user to display a group's page and new posts.<br>Also uses the DatabaseController to add new group data to the database. | D | GroupModule |
| R11: Creates new account entries in the database and verifies sign ins for already existing accounts. | D | AccountModule |
| R12: Fetches information relating to a search query from the relevant databases using the DatabaseController and formats it to be sent to the user. | D | SearchModule |

- Associate definitions

| **Concept pair** | **Association Description** | **Association Name** |
|---|---|---|

| | | |
|---|---|---|
| UserInterface<->ServiceWorker | The User Interface Displays information to the user that is provided by the ServiceWorker | Display |
| ServiceWorker<->Controller | The ServiceWorker communicates with the Controller request and send data back and forth | Request Communication |
| Controller<->GroupModule | The Controller handles requests from the GroupModule about updating group information | Request handling |
| GroupModule<->DatabaseController | The GroupModule Contacts the the DatabaseController to get or update group data in the Database | Data Interaction |
| Controller<->AccountModule | The Controller handles requests from the AccountModule about creating and verifying user accounts | Request handling |
| AccountModule<->DatabaseController | AccountModule sends create read, or delete requests to the DatabaseController | Data Interaction |
| Controller<->SearchModule | Controller handles requests from the SearchModule | Request handling |
| SearchModule<->DatabaseController | SearchModule sends create read, or delete requests to the DatabaseController | Data Interaction |
| Controller<->MessagingModule | Controller receives requests to send, read, and delete messages from the MessageModule | Request Handling |
| MessagingModule<->DatabaseController | MessageModule sends create read, or delete requests to the DatabaseController | Data Interaction |
| Controller<->MapModule | Controller handle request about map interaction by sending appropriate request through MapModule | Request Handling |
| MapModule<->DatabaseController | MapModule sends create, read, or delete requests to the DatabaseController | Data Interaction |
| Controller<->FeedModule | Controller handle request about feed interaction by sending appropriate request through FeedModule | Request Handling |
| FeedModule<->DatabaseController | FeedModule sends create, read, or delete requests to the DatabaseController | Data Interaction |

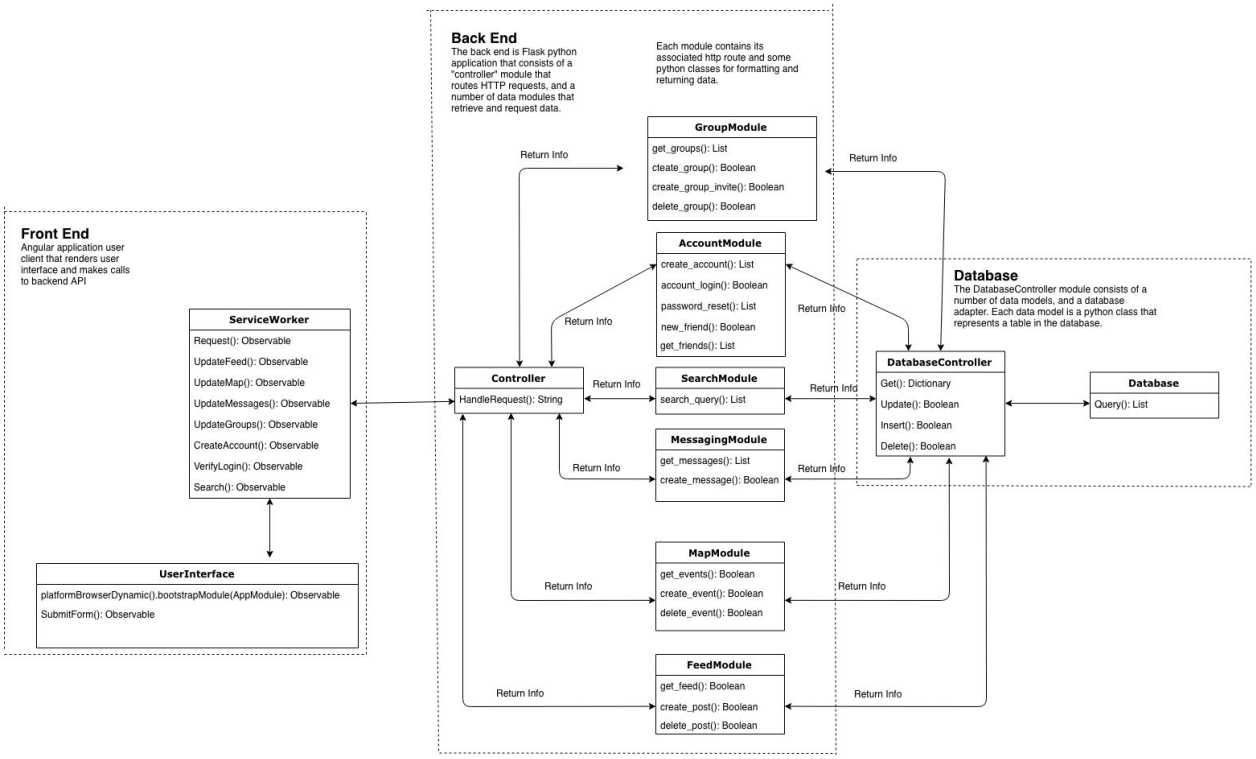
| | | |
|---|---|---|
| DatabaseController<->Database | Database sends Queries to Update the database | Data handling |

- Attribute definitions

| Concept | Attribute | Attribute Description |
|---|---|---|
| Controller | Handle Request | Handle requests from the user by interact with appropriate module to update data in the database and update user interface |
| ServiceWorker | Request | Communicate user request from the interface to the controller |
| UserInterface | Display | Display information data to the user on screen |
| | SubmitForm | Send request base on user interaction on the screen through the ServiceWorker to the Controller for appropriate action |
| Database | Query | Modify data in the database, base on the request of the DatabaseController |
| DatabaseController | Get | Handle request to get and return data from the database |
| | Update | Handle request to update data in the database |
| | Insert | Handle request to insert new data into the database |
| | Delete | Handle request to delete data in the database |
| FeedModule | GetFeed | Get the post data that the user request from the database and format it to human readable format and send it back to interface to be display on screen |
| | CreatePost | Take user's newly created post and added it to the database |
| | DeletePost | Delete the post that the user request from the database |

| | | |
|---|---|---|
| MapModule | GetEvent | Get data about a map event from the database and send it to be display on the interface |
| | CreateEvent | Added data about a map event that is created by user into the database |
| | DeleteEvent | Delete data about a map event from the database |
| MessagingModule | GetMessage | Get data about user request messages from the database |
| | CreateMessage | Added user newly send message to the database |
| | DeleteMessage | Delete message that user requested from the database |
| GroupModule | GetGroups | Get data information about a group from the database |
| | CreateGroup | Add group data that user entered into the database |
| | CreateGroupPost | Add post data from a group to the database |
| | CreateGroupInvite | Add user's send group invitations to the database |
| | DeleteGroup | Remove a group data from the database |
| | DeleteGroupPost | Remove post data from a group from the database |
| AccountModule | CreateAccount | Add a newly user created account information into the database |
| | VerifyLogin | Check user entered login information with user's data in the database and allow user to access that information if it matched |
| SearchModule | Search | Check if a specific data that that the user enter is in the database |

## (a) Class Diagram




## (b) Data Types and Operation Signatures

### UserInterface

platformBrowserDynamic().bootstrapModule(AppModule): Boolean

- Boolean variable corresponds to if UI is displayed

SubmitForm(): Boolean

- Boolean variable corresponds to if the form submitted

### ServiceWorker

Request(): Boolean

- Boolean variable corresponds to if the request was successful or not.

UpdateFeed(): Observable<String[]>

- Observable variable corresponds to a String array that will populate the feed as new data is received. The data is subscribed to

UpdateMessages(): Observable<String[]>

- Observable variable corresponds to a String array that will populate the messages for the user. The data is subscribed to

UpdateGroups(): Observable<String[]>

- Observable variable corresponds to a String array of Groups. The data is subscribed to to keep up to date.

CreateAccount(): Boolean

- Boolean variable corresponds to if the request to create an account was successful or not.

VerifyLogin(): Boolean

- Boolean variable corresponds to if the login was successful or not

Search(): Observable<String[]>

- Observable variable corresponds to a String array of Events and People nearby. The data is subscribed to

## Controller

HandleRequest(): String

- String variable corresponds with the data that is returned from the modules

## GroupModule

get_groups(): List

- List variable corresponds with the list of groups that are returned from the search

create_group_invite(): Boolean

- Boolean variable corresponds to if the group invite was successful or not

delete_group(): Boolean

- Boolean variable corresponds to if the group was able to deleted or not

delete_group_invite(): Boolean

- Boolean variable corresponds to if the group invite was able to deleted or not

## AccountModule

create_account(): List

- List variable corresponds with the account information that is provided by the user

account_login(): Boolean

- Boolean variable corresponds to if the login was successful or not

password_reset(): List

- List variable corresponds to the new password that is sent to the back end of the application

new_friend(): Boolean

- Boolean variable corresponds to the respons to the new friend request response

get_friends(): List

- List variable corresponds to the returned list of friends

## SearchModule

search_ query(): List

- List variable corresponds with the list that is returned as a result of the search

## MessagingModule

get_messages(): List

- List variable corresponds to if the messages were properly retrieved

create_message(): Boolean

- Boolean variable corresponds to if the message was able to deleted or not

## MapModule

get_events(): Boolean

- Boolean variable corresponds to if the events were properly retrieved

create_events(): Boolean

- Boolean variable corresponds to if the event was able to be created or not

delete_event(): Boolean

- Boolean variable corresponds to if the event was able to deleted or not

## FeedModule

get_feed(): Boolean

- Boolean variable corresponds to if the feed was properly retrieved

create_post(): Boolean

- Boolean variable corresponds to if the post was able to be created or not

delete_post(): Boolean

- Boolean variable corresponds to if the post was able to deleted or not

## DatabaseController

Get(): Dictionary

- Dictionary variable corresponds with the data that is returned from the database to the controller

Update(): Boolean

- Boolean variable corresponds to if the database was able to delete or not

Insert(): Boolean

- Boolean variable corresponds to if the database was able to insert or not

Delete(): Boolean

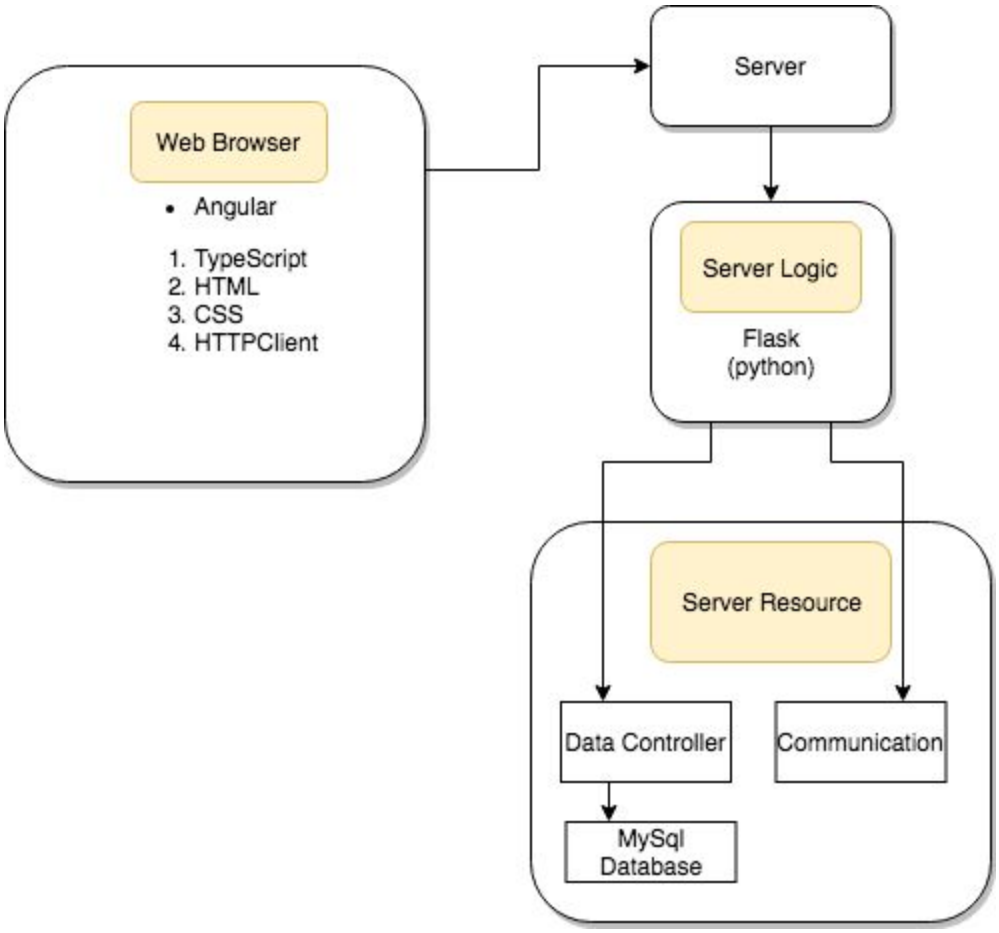
- Boolean variable corresponds to if the database was able to delete or not

## Database

Query(): List

- List variable corresponds with the returned information from the database

## (b) Identifying Subsystems




Subsystems Diagram

## (c) Mapping Subsystems to Hardware

Proximity runs on a client-server architecture. The client exists on the user's computer or mobile device and the server exists on Amazon Web Services.

## (d) Persistent Data Storage

Proximity uitizes MySQL, a relational database system, to store all the information needed about users, posts, events, and groups. Only the model part of the system is allowed to interact with this database. The database lives on Proximity's server.

## (e) Network Protocol

Proximity uses the HTTP as the communication protocol. This was chosen due to HTTP being the standard client-server protocol. It also has the advantage of being stateless and platform independent.

## (f) Global Control Flow

Execution Orderliness

The execution order is event-driven. The application provides a variety of events that the user can activate in an order they desire. For example, a user can create an event then message a friend, or the user can message a friend and then create an event. Aside from from first logging in, there's is no required order for these events.

Time Dependency

Proximity is a real time system in the sense that it records the datetime of every post and every message. In addition events that are posted to the map expire after a certain amount of time.

## (g) Hardware Requirements

- Screen Display
- An internet connection
- A web browser

# 4) Algorithms and Data Structures (If applicable)

## (a) Algorithms

The main algorithm in use is a distance finding algorithm that takes two sets of location data, in (latitude, longitude) format, and returns the distance between the two points in miles. This algorithm is used in database queries to find nearby posts, events, and people.

## (b) Data Structures

Our WebApp is going to pass data back and forth between the database and controller to update the user interface. These data will have the format of associative arrays, which is easy to implement. Other than that, we are not going to use any other complex data structures in our webapp.

## Pre and Post-Conditions for Functions

### UserInterface

platformBrowserDynamic().bootstrapModule(AppModule): Boolean

- Pre-Conditions: The user makes any action to change the view of the application. This includes changing pages, updating the feed or events, sending a message, or any other action that the user can perform that results in a change of data.
- Post-Conditions: The user interface will be updated with the most up to date data from the application.

SubmitForm(): Boolean

- Pre-Conditions: The user is on a page of the site that has a form. The user inputs data into all of the required fields and submits the data to the application.
- Post-Conditions: The form data will be either be submitted or checked against the database to see if data matches.

### ServiceWorker

Request(): Boolean

- Pre-Conditions: The user makes an action that results in a request to the controller of the application. This includes all possible user actions on the application.
- Post-Conditions: The back end of the application will either accept or deny the request resulting in the user interface being updated

UpdateFeed(): Observable<String[]>

- Pre-Conditions: The application's interface is displaying a "Feed page", listing a feed of other user posts. The user will either create, search, or delete a post on the feed.
- Post-Conditions: The application will send the data to the database and update the information on the back end as well as update the feed on user interface.

UpdateMessages(): Observable<String[]>

- Pre-Conditions: The user is on the message page and either creates a new message or deletes an existing message.
- Post-Conditions: The database either adds or deletes a message and the user interface is updated. The message is deleted or created for all recipient users.

UpdateGroups(): Observable<String[]>

- Pre-Conditions: The user is on the groups page and creates or updates a group.
- Post-Conditions: The group information in the database will be updated. The user interface of all of the group members will also be updated.

CreateAccount(): Boolean

- Pre-Conditions: The application's interface is displaying a login screen that will prompt the user to enter a username and password or if they do not already have a username there is a button that will redirect them to a page where they can fill out a form and create an account.

- Post-Conditions: The user information will be encrypted and sent to the database and stored to be checked against the next time the user logs in to the account.

VerifyLogin(): Boolean

- Pre-Conditions: The application's interface is displaying a login screen that will prompt the user to enter a username and password.
- Post-Conditions: The user information will be verified and they will be redirected to their feed page where they can view friends posts.

Search(): Observable<String[]>

- Pre-Conditions:
- Post-Conditions:

## Controller

HandleRequest(): String

- Pre-Conditions: The user will perform any action on the application and this function will be called to communicate between the service worker and the back end of the application.
- Post-Conditions: The back end of the application will be updated or checked against and the user interface as a result will be updated.

## GroupModule

GetGroups(): List

- Pre-Conditions: The user presses the "group page" button in the navigation bar navigating them to the group page.
- Post-Conditions: The applications interface will return a display of all of the groups the user is in.

CreateGroupInvite(): Boolean

- Pre-Conditions: The application's interface is displaying a "groups page", listing all of the groups the user belongs to. On the same page is a group invite button.
- Post-Conditions: An invitation message is sent to the other user that if they accept they will be added to the group.

DeleteGroup(): Boolean

- Pre-Conditions: The application's interface is displaying a "groups page", listing all of the groups the user belongs to. On the same page is a delete group button.
- Post-Conditions: Group data is removed from the database. The group no longer exists in the database and group members can no longer access the group.

DeleteGroupInvite(): Boolean

- Pre-Conditions: The application's interface is displaying a "groups page", listing all of the groups the user belongs to. On the same page is a delete invite within the group button.
- Post-Conditions: The group invite is removed from the database. The group invitation is no longer visible to any members of the group.

## AccountModule

create _account(): List

- Pre-Conditions: The application's interface is displaying a login screen that will prompt the user to click a link if they do not have an account.
- Post-Conditions: The user will have a username and account in the database. The user will now be able to use the app.

account_login(): Boolean

- Pre-Conditions: The application's interface is displaying a login screen that will prompt the user to enter a username and password.
- Post-Conditions: The user information will be verified and they will be redirected to their feed page where they can view friends posts.

password_reset():

- Pre-Conditions: The application's interface is displaying a login screen that will prompt the user to enter a username and password or reset their password.
- Post-Conditions: The user will receive an email allowing them to reset their password.

new_friend():

- Pre-Conditions: The user selects another user to send a friend invitation to
- Post-Conditions: The user will receive a friend invitation from the sending user

get_friends():

- Pre-Conditions: The application's interface will get the users friends when the feed or map page is loaded
- Post-Conditions: The users friends posts will show up on the map or on the feed page

## SearchModule

Search(): List

- Pre-Conditions: The application's interface will be either displaying the feed page or the maps page that will have a text field for the user to input a search for the feed or for the maps.
- Post-Conditions: The users search will be sent to the database controller where then the database controller will check the database to see if any results match. If any results match the user interface will display the results.

## MessagingModule

get_messages(): List

- Pre-Conditions: The user presses the "message page" button in the navigation bar navigating them to the message page.
- Post-Conditions: The applications interface will return a display of all of the "message page" that displays the messages between the user and others.

create_message(): Boolean

- Pre-Conditions: The application's interface is displaying a "message page", listing all of the groups the user belongs to. On the same page is a create message form.

- Post-Conditions: The message data is added to the database. The recipients as well as the sender of the message can now view the message on the “message page”

## MapModule

get_events(): Boolean

- Pre-Conditions: The user presses the “map” button in the navigation bar navigating them to the map page.
- Post-Conditions: The applications interface will return a display of the “map page” that displays a map that contains events locations and information on it.

create_events(): Boolean

- Pre-Conditions: The application’s interface is displaying a “Map page”, listing all of the events on a map to the user. On the same page is a create event button for the user to input event details.
- Post-Conditions: The events data will be entered into the database. The applications interface will return an updated display of the “map page” that displays a map that contains the newly created events locations and information on it as well as all other events.

delete_event(): Boolean

- Pre-Conditions: The application’s interface is displaying a “Map page”, listing all of the events on a map to the user. On the same page is a delete event button for the events that this user created.
- Post-Conditions: The event data is removed from the database. The event no longer exists in the database and no users can see the event anymore.

## FeedModule

get_feed(): Boolean

- Pre-Conditions: The user presses the “feed” button in the navigation bar navigating them to the feed page.
- Post-Conditions: The applications interface will return a display of all of the “feed page” that displays the feed of other users posts as well as your own.

create_post(): Boolean

- Pre-Conditions: The application’s interface is displaying a “Feed page”, listing a feed of other user posts. On the same page is a create post button.
- Post-Conditions: The post data is entered into the database. The applications interface will return an updated display of the “Feed page” listing the new post above all of the previous posts.

delete_post(): Boolean

- Pre-Conditions: The application’s interface is displaying a “Feed page”, listing a feed of other user posts as well as your own. Next to your own post is a delete post button.
- Post-Conditions: The post will be removed from the database. No users will be able to view this post anymore. The applications interface will return a display of

the “feed page” that displays an updated feed that does not include the deleted post.

## DatabaseController

Get(): Dictionary

- Pre-Conditions: The user performs an action on the application that requires the database controller to get or read data from the database.
- Post-Conditions: The database will be read or referenced against from the database controller and will return information necessary to know if the information was inserted correctly or if the information exists in the database.

Update(): Boolean

- Pre-Conditions: The user performs an action on the application that updates the database such as editing an existing post or replying to another users post.
- Post-Conditions: The information will be sent to the database and the database will be updated.

Insert(): Boolean

- Pre-Conditions: The user performs an action on the application that inserts new data into the database such as creating a post or event.
- Post-Conditions: New information will be inserted into the database from the database controller.

Delete(): Boolean

- Pre-Conditions: The user performs an action on the application that deletes information from the database such as deleting a post or event.
- Post-Conditions: The data is deleted from the database and no users will be able to see or access the information anymore.

## Database

Query(): List

- Pre-Conditions: The user will need to perform an action on the web app that results in the database controller sending or requesting data from the database.
- Post-Conditions:  The database will return the results of the Query to the database controller.