Wade Johnson (wmj2108)
Charlie Summers (cgs2161)
Nick Clark (ntc2120)
Wenhe Henry Qin (whq2000)
Magikarp
COMS W4156
Magikarp Team Assignment 5

Part 1: User Stories

1. As a traveler, I want to discover crowdsourced travel tips and advice directly on my mobile device so that I can be best informed during my vacations.

   My conditions of satisfaction are:
   - An application on a mobile device that I can carry with me while travelling.
   - Ability to read posts created by other users.
   - If the application cannot download content, provide the ability to view the most recently accessed data offline or inform the user if no data is available.

2. As a traveler, I want to be able to find the best travel locations near me so that I know where to go in order to have the best experience.

   My conditions of satisfaction are:
   - An interactive map that shows my location relative to various crowdsourced destinations.
   - Ability to read the full posts that I select from the map.
   - Ability to get directions to the geotagged location of the post.
   - If I am unable to select a post (for example, due to network connectivity issues) inform me that the post cannot be viewed right now.

3. As a content creator, I want to share geotagged text descriptions and pictures of areas of interest so that I can help other travelers.

   My conditions of satisfaction are:
   - Create a post and add a text description and picture of an area of interest.
   - The text description cannot be blank and cannot be longer than 280 characters.
   - Geotag a post using the current location on the device.
   - If a post cannot be created (ex. due to loss of network connectivity or inability to determine current location) provide a notification and ask me to try again.

4. As a content creator, I want to securely log in with an account so that I can control the content of my posts.

   My conditions of satisfaction are:
   - Log in by linking my existing Google account to create and edit posts.
   - If login fails (for example, because I already have an account linked) then provide a notification.
   - If I attempt to create a post without a linked Google account or try to edit another user's post, prevent the change and respond with an error message saying I cannot do that.

5. As a content creator, I want to auto-save partially complete post drafts so that I can finish them later without losing my work.

   My conditions of satisfaction are:
   - Auto-save the current post I am working on so I can complete and publish it later.
   - If I close and reopen the app while working on a post, the app should return me directly to that post.
   - If an auto-save fails (for example, due to lack of required access permissions) then provide a notification so I know my current input cannot be saved.

Part 2: Test Plans

Android UI Tests: https://github.com/hwqin16/Magikarp/tree/master/app/src/androidTest/
Android Unit Tests: https://github.com/hwqin16/Magikarp/tree/master/app/src/test/
Server Unit Tests: https://github.com/hwqin16/Magikarp/tree/master/server/src/test/

Test Plans for Android Application (Front End)

- Post Editor Content Validation Test Plan. A post consists of three primary pieces of content: an image, a geotagged location and a text description. Primary methods for this test plan: PostFragment#onPostButtonClick() (including #uploadFile() and #uploadPost() by proxy).
    - Testing the user's inclusion of an image in the post is binary; the user has either selected an image or not. Thus, our equivalence classes are also binary, with one class corresponding to the user choosing an image for inclusion in the post and the other when the user has not chosen an image. Our application shall only allow users to create new posts or edit existing posts if the user has chosen an image in the editor.
    - Similar to image uploads, there are two partitions for the geotagged location of a post: the user has set the location or not set the location. It follows that our equivalence classes are equivalent to the partitions themselves. Our application shall only allow users to create posts when the location has been set.
    - Valid text descriptions must have at least 1 character but no more than 280 characters. This makes our equivalence classes strings of lengths $z = 0$, $0 < z <= 280$, and $z > 280$, where z is an integer value.

0	1 - 280	> 280

Equivalence Partitions/Classes for Text Description Length

    - The post editor supports creating new posts from scratch and editing existing posts. The state space of the editor is therefore divided into a binary space of "New" and "Edit". In the "New" space, none of the post requirements have been covered and a user must perform steps to ensure all three pieces of content are covered. In the "Edit" space, all of the requirements for a post have been previously covered but one or more of them may be altered.
    - **Multidimensional Class for New Post Success:** There is an image, there is a geolocation, there is a caption between 1 and 280 characters inclusive, and the editor is in a new post state.

- **Multidimensional Class for Edit Post Success:** There is an image, there is a geolocation, there is a caption between 1 and 280 characters inclusive, and the editor is in an edit post state.
- Boundary Analysis
  - Upload Image
    - Test with no image
    - Test with image
  - Geo-Location
    - Test with location
    - Test with no location
  - Post Caption/Description
    - Test with empty (0 characters, invalid)
    - Test with 1 character (lower boundary of valid)
    - Test with 2 characters (one above lower boundary of valid)
    - Test with 280 characters (upper boundary of valid)
    - Test with 279 characters (one below upper boundary of valid)
    - Test with 140 characters (middle of valid)
    - Test with 281 characters (lower boundary of invalid)
    - Test with 282 characters (one above lower boundary of invalid)
    - Test with 300 characters (above lower boundary of invalid with no upper bound)
  - Post Editor Type
    - Test with New Post type
    - Test with Edit Post type
- Test Plans for Equivalence Classes
  - Test creating a new post successfully
    - Open the Android application
    - Log in to the application
    - Navigate to “My Places”
    - Click the “+” icon in the top right corner
    - Click the image icon and select an image
    - Enter the caption into the ‘Description’ field no more than 280 characters.
    - Click the location icon at the top right corner of the screen
    - Click the post icon at the top right corner of the screen
    - Successful if the application goes back to the previous screen and displays the new post on the map.
    - The test is repeated multiple times, performing the caption, location and image steps in a different order.
  - Test editing a post successfully
    - Open the Android application
    - Log in to the application
    - Navigate to “My Places”
    - Click a marker on the map to open the editor

        - Change the text of the description to be a new string between 1 and 280 characters inclusive
        - Click the post icon at the top right corner of the screen
        - Click on the same marker on the map
        - Successful if the changes are reflected correctly in the view of the post.
        - Repeat the test multiple times, editing an image, the geotagged location and then varying combinations of the three.
- Test entering a description string of length 0
    - Open the Android application
    - Log in to the application
    - Navigate to “My Places”
    - Click the “+” icon in the top right corner
    - Click the image icon and select an image
    - Click the location icon at the top right corner of the screen
    - Click the post button
    - Successful if the application does not proceed with upload and notifies the user that a required content item is missing.
- Test entering a description string of greater than length 280
    - Open the Android application
    - Log in to the application
    - Navigate to “My Places”
    - Click the “+” icon in the top right corner
    - Click the image icon and select an image
    - Enter the caption into the description field and attempt to type more than 280 characters.
        - Failure case if the text field allows new characters to be typed after it reaches 280 characters or if the character counter goes above 280 characters.
    - Click the location icon at the top right corner of the screen
    - Click the post button
    - Successful if the application goes back to the previous screen and displays the new post on the map (noting that the UI does not allow typing more than 280 characters).
- Test not selecting an image
    - Open the Android application
    - Log in to the application
    - Navigate to “My Places”
    - Click the “+” icon in the top right corner
    - Click the location icon at the top right corner of the screen
    - Enter the caption into the description field no more than 280 characters.
    - Click the post button

            - Successful if the application remains on the fragment and notifies the user that a required content item is missing.
        - Test not entering a location.
            - Prerequisites: User has not yet approved app permission to location services.
            - Open the Android application
            - Log in to the application
            - Navigate to “My Places”
            - Click the “+” icon in the top right corner
            - Click the image icon and select an image
            - Enter the caption into the description field no more than 280 characters.
            - Click the post button
            - Successful if the application does not proceed with the upload and notifies the user that a required content item is missing.

- Post Editor Contextual Actions Test Plan. The available actions presented to the user will vary based on the state of the system. Primary methods for this test plan: PostFragment#onCreateOptionsMenu().
    - There are three states the post editor can be in, representing our three equivalence classes: “New”, “Edit”, and “View”.
    - Boundary Analysis
        - Test with New Post State
        - Test with Edit Post State
        - Test with View Post State
    - Test Plans for Equivalence Classes
        - Test entering new post state
            - Prerequisites:
                - User is logged into the device
                - Application is running
                - Application is coming up from a cold start
            - Navigate to “My Places”
            - Click the “+” icon in the top right corner
            - Successful if there are no contents in the fields
        - Test entering the edit post state
            - Prerequisites:
                - User is logged into the device
                - Application is running
            - Navigate to “My Places”
            - Click a marker on the map
            - Successful if the contents of the fields match what is in the database
        - Test entering the view post state
            - Prerequisites:

            - Application is running
        - Navigate to “Map”
        - Click on any marker
        - Wait for a reasonable amount of time for the image and text description to load.
        - Successful if the image and text content match the database entry corresponding to the clicked marker.

- Post Editor Actions Execution Test Plan. There are four distinct action buttons that can be pressed representing four of the five equivalence classes for this test plan: the get location button, the upload content button, the delete button, and the directions button. The fifth equivalence class occurs when an unrecognized button is pressed. Primary methods for this test plan: PostFragment#onOptionsItemSelected().
    - Boundary Analysis
        - Test by pressing the location button
        - Test by pressing the upload button
        - Test by pressing the delete button
        - Test by pressing the directions button
        - Test by manually invoking the action callback with a different ID than the ones above
    - Test Plans for Equivalence Classes
        - Location Button and Upload Button (tested with same test):
            - Prerequisites:
                - On the post editor view populated with an image and a valid caption.
                - User is logged in.
            - Click the location icon
                - Successful if a location permission request dialog opens (if the user has not yet granted location permissions), or a notification stating the location has been updated.
            - Click the post icon
            - Successful if application has location permissions and no notification that a user-rooted error has occurred.
        - Delete Button
            - Prerequisites:
                - User is logged in
                - User is editing a post that they created through the “My Places” navigation
            - Click the ellipsis icon to open the action overflow menu (note that delete action is never displayed directly on the main action bar)
            - Click the “Delete post” option
            - Successful if the post no longer exists on the server or within the application
        - Directions Button:

                - Prerequisites:
                    - User is viewing a post through the “Map” navigation
                - Click the navigation button at the top right of the screen
                - Successful if the device define map application opens to the given location
            - Button with other ID will be tested through unit tests, as it is not invokable through user interaction

- Post Editor User Interface Test Plan. The Post Editor user interface (UI) can exist in three states, represented by three equivalence classes. The origin of data can be the saved instance information from android, it can be retrieved from the arguments of the view, or there can be no population of the view (occurs on a new post with no cached data). Primary methods for this test plan: PostFragment#onViewCreated().
    - Boundary Analysis
        - Data Population
            - Populate from saved instance bundle
            - Populate from arguments
            - No population
        - Editable State
            - View only state
            - Edit Post state
            - New Post state
    - Test Plans for Equivalence Classes
        - New Post No Population
            - Prerequisites:
                - Application was cold started
                - User is logged in
            - Navigate to “My Places”
            - Click “+” button in top right corner of screen
            - Success if no fields are populated
        - New Post with Population from Saved Instance Bundle
            - Prerequisites:
                - Application was cold started
                - User is logged in
                - In Developer Options settings, ensure “Don’t keep activities” is turned on to ensure app is freed from memory when put into the background
            - Navigate to “My Places”
            - Click “+” button in top right corner of screen
            - Enter text into description field
            - Enter image into image field
            - Background the application
            - Bring application to foreground
            - Success if fields are populated with the previously entered content

        - Edit Post with Population from Arguments
            - Prerequisites:
                - Application was cold started
                - User is logged in
            - Navigate to “My Places”
            - Click on a marker on the map
            - Success if fields are populated with contents matching the database
        - View Post with Population from Arguments
            - Navigate to “Map”
            - Click on a marker on the map
            - Success if fields are populated with contents matching the database

- Post Editor Image Display Test Plan. There are four equivalence classes for the loading of an image in post editor. One of the success equivalence classes is that the image link stored by the editor resolves to a reachable remote resource. The other successful equivalence is that the link resolves to an existing local resource. The equivalence classes for errors are that the stored link points to a non-resolvable remote resource or a non-existent local resource. Primary methods for this test plan: PostFragment#loadImage().
    - Boundary Analysis
        - Reachable remote resource
        - Unreachable remote resource
        - Existing local resource
        - Non-existing local resource
    - Test Plans for Equivalence Classes
        - The success cases will be covered by testing of creating new posts, editing posts, and viewing posts. All cases will be covered by unit testing.

- Post Editor Location Resolution Test Plan. There are two equivalence classes for the action to occur on the location button click. The first is that the app retrieves a valid location for the user. The second is that the service failed to get a location. Primary methods for this test plan: PostFragment#onLocationButtonClick().
    - Boundary Analysis
        - User has granted app permission to device location services and service returns a valid location
        - User has not granted app permission to device location services (or explicitly denied permission) and location cannot be accessed
        - Location services access granted but no location available
    - Test Plans for Equivalence Classes
        - Location successfully retrieved
            - Prerequisites:
                - Application was cold started

        - User is logged in
    - Navigate to “My Places”
    - Click “+” button in top right corner of screen
    - Click the location button
    - Enter description into text field
    - Click image icon and select image
    - Click post button in top right corner of screen
    - Successful if no notification of missing field
- Location services permission not granted
    - Prerequisites:
        - Application was cold started
        - User is logged in
        - User has not yet granted app permission to location services
    - Navigate to “My Places”
    - Click “+” button in top right corner of screen
    - Click the location button
        - A dialog appears asking user to grant permission
        - Allow the app permission to access location services
    - Enter description into text field
    - Click image icon and select image
    - Click post button in top right corner of screen
    - Successful if no notification of missing field
- Location services permission granted but location could not be retrieved
    - Prerequisites:
        - Application was cold started
        - User is logged in
        - No GPS signal and network connection is disabled
    - Navigate to “My Places”
    - Click “+” button in top right corner of screen
    - Click the GPS button
    - Enter description into text field
    - Click image icon and select image
    - Click post button in top right corner of screen
    - Successful if notification of missing field

- Post Editor Image Upload Test Plan. When an image is uploaded to the server, the server responds with a remote URL indicating the image’s new location. There are two equivalence classes for the file upload callback. The successful equivalence class is that a URL to a remote resource exists and contains a scheme indicating this (specifically “https” in this case). If this is not true, a result falls into the failure equivalence class. Primary methods for this test plan: PostFragment#onFileUploaded().
    - Boundary Analysis
        - Remote URL exists and contains a remote scheme

        - Remote URL exists and does not contain a remote scheme
        - Remote URL does not exist
    - Test Plans for Equivalence Classes
        - This will not be tested directly through user-oriented test plans. This is a smaller function that will be tested through unit testing.

- Post Editor Download URL Fetch Test Plan. There are two possible states of the download URL fetch task: successful and unsuccessful. The unsuccessful state can be subdivided into having an exception and not having an exception. These are the three equivalence classes for the URL fetch task. Primary methods for this test plan: PostRepository#uploadFile() (#getDownloadUrl() and #onUriReceived() by proxy)
    - There are two possible states for the on URI received task. If the listener for "on URI received" exists versus if it doesn't. We can subdivide the state where it exists into two states: the case where there is a successful URI task result provided and a case where there isn't. Thus, these are the three equivalence partitions.
    - **Multidimensions Class for Successful Upload:** The download URL fetch task is successful and there exists a URI received listener.
    - Boundary Analysis
        - Download URL fetch task
            - URL task successful
            - URL task unsuccessful and has an exception
            - URL task unsuccessful and does not have an exception
        - URI Received Task
            - Listener exists and the URL task was successful
            - Listener exists and the URL task was unsuccessful
            - Listener does not exist and the URL task was successful
            - Listener does not exist and the URL task was unsuccessful
    - Test Plans for Equivalence Classes
        - As part of post creation testing, this will be tested. In addition, unit tests will cover this functionality by testing each of the boundaries.

- Main Activity User Interface Test Plan. The main activity controls the top-level navigation for the entire app. The user can be in two states when viewing the UI, logged in or logged out. Logged in can be subdivided into two categories based on account information: with and without an avatar/profile picture. Primary methods for this test plan: MainActivity#updateSignInUi() (and #setLoggerdInUi(), setLoggedOutUi() by proxy).
    - Boundary Analysis
        - No user logged in
        - User logged in with no avatar
        - User logged in with an avatar
    - Test Plans for Equivalence Classes
        - No user logged in

                - Prerequisites:
                    - User is not logged in
                - Open navigation menu
                - Successful if there is no user in the top banner of the menu
            - User logged in with avatar
                - Prerequisites:
                    - User is logged in
                    - User has avatar on Google account
                    - Application has network connection
                - Open navigation menu
                - Successful if there is a user in the top banner of the menu and the avatar corresponds to that of the user's Google account
            - User logged in without avatar
                - Prerequisites:
                    - User is logged in
                    - User has no avatar on Google account
                - Open navigation menu
                - Successful if there is a user in the top banner of the menu and the avatar is the default avatar

- Map Viewer Filtering Test Plan. Depending on whether a user is logged in to the app, there are options to view all available posts or just posts created by the user. There are two possible states for the user's log-in status: logged in or logged out. These are the two equivalence classes for this state set. Primary methods for this test plan: MapsFragment#onGoogleSignInAccountChanged().
    - There are two possible states for viewing the map: if it's in a user data view state or a generic view state. These are the two equivalence classes for this state set.
    - We also should check if the account being driven to is the same as the currently signed in account. This leads to 3 possible states: the new account is signed out, the new account and old account are signed in and the same account, and the new account and old account are signed in, but different. These are the three equivalence partitions for this state set.
    - **Multidimensional Class for Changing View:** There is currently a logged in user, the target view is a logged in view, and the new user is not the same as the old user.
    - Boundary Analysis
        - Current Logged In Status
            - Logged In
            - Logged Out
        - Current Map View
            - Generic View
            - User Data View
        - Requested Account Status
            - Logged Out

                - Same Account
                - Different Account
        - Test Plans for Equivalence Classes
            - Log out while on “My Places”
                - Prerequisites:
                    - User is logged in
                    - Application is on “My Places”
                - Open navigation menu drawer
                - Click log out
                - Successful if the application navigates to the previous screen
            - Log in while on “Map”
                - Prerequisites:
                    - User is logged out
                    - Application is on “Map”
                - Open navigation menu
                - Click log in
                - Successful if the application navigates to the “Map”
            - Testing “Same Account” and “Different Account” boundaries will be done in unit testing, as they cannot be reached directly through the UI

- Map Viewer Content Update Test Plan. There are two distinct state state sets: the map’s existence and the list of messages containing post content. For a success case, the map exists and the messages received must be non-empty. This is one of the equivalence classes. Another equivalence class is when no map exists. The final equivalence class is when a map exists, but the messages fetched are empty. Primary methods for this test plan: MapsFragment#onMessagesChanged().
    - Boundary Analysis
        - The map exists and there is a non-empty list of messages
        - The map exists and there is an empty list of messages
        - The map does not exist
    - Test Plans for Equivalence Classes
        - Map exists with non-empty set of messages
            - Open map view of application
            - Navigate to region of map with posts
            - Successful if markers appear
        - Map exists with empty set of messages
            - Open map view of application
            - Navigate to region of map without posts
            - Successful if no markers appear
        - The map does not exist
            - Prerequisites:
                - Cold installation of application
                - No network connection
            - Open map view of application

- No map is visible

Test Plans for Server (Backend)

- For Backend testing, we will be testing the API endpoints that we expose for the application. These API endpoints are all handled in our Server class. The other major classes are MessagePosterImpl and MessageFinderImpl which we also define equivalence classes for here.

    - POST /message/{user_id}/new (MessagePosterImpl#postNewMessage())
        - There are two possible states of the endpoint to post a new post, either the post was successful or the post was unsuccessful. The successful state will return a 201 return code with the record id of the post, while an unsuccessful state will return a 401 code. These are the two equivalence classes for the /message/{user_id}/new endpoint.
        - Boundary Analysis
            - Upload a new post to the server
                - Post endpoint returns success (201)
                - Post endpoint returns Fail (401)
        - Test Plans for Equivalence Classes
            - Test the new post endpoint successful
                - Start a server
                - Send a fully populated new post POST call to the server
                - Verify that the server returns a 201 code with a Record ID
            - Test the new post endpoint is unsuccessful
                - Start a server
                - Send a partially populated new POST call
                - Verify that the server returns a 401 code

    - POST /message/{user_id}/update/{record_id} (MessagePosterImpl#updateMessage())
        - There are three possible states of the endpoint to update a post, either the post was successfully updated, the post was unsuccessfully updated, the user is not authorized. The successful state will return a 201 return code, while an unsuccessful update will return a 401 code. If the post is trying to be updated by a user that does not own the post, the server will respond with an unsuccessful unauthorized error. These are the three equivalence classes for the /message/{user_id}/update/{record_id}.
        - Boundary Analysis
            - Update an existing post to the server
                - Update Post endpoint returns success (201)
                - Update Post endpoint returns Fail (401)
                - Update Post endpoint returns unauthorized (401 w/ error message “Unauthorized”)

        - Test Plans for Equivalence Classes
            - Test the update post endpoint successful
                - Start a server
                - Create a new post using the new post endpoint
                - Send a fully populated update POST call to that endpoint to the server
                - Verify that the server returns a 201 code
            - Test the update post endpoint is unsuccessful
                - Start a server
                - Create a new post using the new post endpoint
                - Send a partially populated update POST call
                - Verify that the server returns a 401 code
            - Test the update post endpoint unauthorized
                - Start a server
                - Populate the database with two record ids for two different users
                - Send a fully populated update post call to a recordID that is different from the owner of the record id
                - Verify that the new post returns a 401 code with the error “You do not own this post”

    - POST /message/{user_id}/delete/{record_id} (MessagePosterImpl#deleteMessage())
        - There are three possible states of the endpoint to delete a post, either the post was successfully deleted, the post was unsuccessfully deleted, the user is not authorized. The successful state will return a 201 return code, while an unsuccessful state will return a 401 code. If the post is trying to be deleted by a user that does not own the post, the server will respond with an unsuccessful unauthorized error. These are the three equivalence classes for the /message/{user_id}/delete/{record_id}.
        - Boundary Analysis
            - Delete an existing post from the server
                - Delete Post endpoint returns success (201)
                - Delete Post endpoint returns Fail (401)
                - Delete Post endpoint returns unauthorized (401 w/ error message “Unauthorized”)
        - Test Plans for Equivalence Classes
            - Test the delete post endpoint successful
                - Start a server
                - Create a new post using the new post endpoint
                - Send a fully populated delete POST call to that endpoint to the server
                - Verify that the server returns a 201 code
            - Test the delete post endpoint is unsuccessful

            - Start a server
            - Create a new post using the new post endpoint
            - Send a partially populated delete POST call
            - Verify that the server returns a 401 code
        - Test the delete post endpoint unauthorized
            - Start a server
            - Populate the database with two record ids for two different users
            - Send a fully populated delete post call to a recordID that is different from the owner of the record id
            - Verify that the new post returns a 401 code with the error “You do not own this post”

- POST /messages
    - This method is a POST instead of a GET because the request library we’re using on the android side doesn’t allow setting a request body unless it’s a POST.
    - There are six equivalence classes for the messages endpoint: 1 valid one (returning a list of posts) and 5 invalid ones (returning various error codes). Since we’re passed a bounding box containing two latitudes and two longitudes, we validate that each is within the expected range and non-null which lead to 4 of the invalid cases (one for each latitude and longitude passed). The final invalid equivalence class is when the number of max records is either not passed or is a negative integer.
    - Boundary Analysis
        - Get posts in a specified bounding box up to a maximum number of posts.
            - Get Posts endpoint returns a list of posts and the number of posts returned
            - Get Posts endpoint returns an error message “Invalid latitude_bottom” if the passed latitude_bottom is null (not passed), below -90, or above 90
            - Get Posts endpoint returns an error message “Invalid latitude_top” if the passed latitude_top is null (not passed), below -90, or above 90
            - Get Posts endpoint returns an error message “Invalid longitude_left” if the passed longitude_left is null (not passed), below -180, or above 180
            - Get Posts endpoint returns an error message “Invalid longitude_right” if the passed longitude_right is null (not passed), below -180, or above 180
            - Get Posts endpoint returns an error message “Invalid max_records” if the passed max_records is null (not passed), or below 0

- Test Plans for Equivalence Classes
    - Test the valid response from the messages endpoint
        - Start a server
        - Pass a valid request with proper latitude_bottom, latitude_top, longitude_left, and longitude_right. Set max_records to 0 - a valid value and it also ensures that the tests don't conflict with production data (we were having flaky test issues when trying to check real messages, so we instead test this when checking MessageFinderImpl#findByBoundingBox())
        - Confirm that the response message list is empty as expected
    - Test the valid response right at the boundaries
        - Start a server
        - Pass a valid request with latitude_bottom = -90, latitude_top = 90, longitude_left = -180, longitude_right = 180, and max_records = 0
        - Confirm that the response message list is empty as expected
    - The below are all tested by getByBoundingBoxInvalidInputsTest. We use a single test method using a Table Driven Test because the logic is all the same and there are MANY cases. All the following tests have the following logic:
        - Pass a request with the single specified flaw
        - Assert that the result is the expected message
    - Test the invalid request where latitude_bottom is null and the error is "Invalid latitude_bottom"
    - Test the invalid request where latitude_bottom is -91 and the error is "Invalid latitude_bottom"
    - Test the invalid request where latitude_bottom is 91 and the error is "Invalid latitude_bottom"
    - Test the invalid request where latitude_top is null and the error is "Invalid latitude_top"
    - Test the invalid request where latitude_top is -91 and the error is "Invalid latitude_top"
    - Test the invalid request where latitude_top is 91 and the error is "Invalid latitude_top"
    - Test the invalid request where longitude_left is null and the error is "Invalid longitude_left"
    - Test the invalid request where longitude_left is -181 and the error is "Invalid longitude_left"
    - Test the invalid request where longitude_left is 181 and the error is "Invalid longitude_left"

- Test the invalid request where longitude_right is null and the error is “Invalid longitude_right”
- Test the invalid request where longitude_right is -181 and the error is “Invalid longitude_right”
- Test the invalid request where longitude_right is 181 and the error is “Invalid longitude_right”
- Test the invalid request where max_records is null and the error is “Invalid max_records”
- Test the invalid request where max_records is -1 and the error is “Invalid max_records”
- Test the invalid request where max_records is -10 and the error is “Invalid max_records”

- MessageFinderImpl#findByBoundingBox()
  - The full range of equivalence classes include the methods listed below - we intentionally broke out the rest of the methods in order to ease comprehensibility and testability. So, in testing this method, we primarily focus on the limit being applied as expected. Here, the equivalence classes correspond to the relationship between the number of items returned by the database call and the limit passed. We consider the equivalence class where the limit is larger than the amount returned by the database, where the limit is the same number as returned by the database, where the limit is smaller than the amount returned by the database, and where the limit is 0.
  - Boundary Analysis:
    - We know that the limit passed to this function is always 0 or greater based on the restrictions performed in the /messages endpoint, so we’re only interested in values 0 and greater. This is compared with the number of items returned by the database.
  - Test Plans for Equivalence Classes:
    - We test to make sure that, when the limit is larger than the number of items returned by the database, that we return all the items from the database.
    - We test to make sure that, when the limit is the same number as the number of items returned by the database, that we return all the items from the database.
    - We test to make sure that, when the limit is smaller than the number of items returned by the database, that we return only the number of items specified by the limit.
    - We test to make sure that, when the limit is 0, that we don’t return any items.

- MessageFinderImpl#filterMessage()

        - The full range of equivalence classes for this method encompasses the 6 equivalence classes described in isInsideBoundedBox below, but multiplied by 2 because we have both longitude and latitude. But we refactored out that class to ease the testing load. So instead here we just consider 3 equivalence classes: when the message is in the bounded box, where the longitude is out of the bounded box, and where the latitude is out of the bounded box.
        - Boundary Analysis:
            - We consider values where we're inside and outside the boundary box.
            - Right on the edge of the bounded box we consider it to be inside the boundary box.
        - Test Plans for Equivalence Classes:
            - Test where the message is in the bounded box. The return value should be true.
            - Test where the latitude is outside the bounded box. The return value should be false.
            - Test where the longitude is outside the bounded box. The return value should be false.
            - Test where the message is on the edge of the bounded box. The return value should be true.

    - MessageFinderImpl#isInsideBoundedBox()
        - There are 6 equivalence classes for the inputs to this method and 2 equivalence classes for the outputs (true and false). The entire design of this method is to encapsulate the question of if we're in the boundary box or not into a single method.
        - Boundary Analysis:
            - Test the value when its inside the bounded box with wrap turned on and off
            - Test smaller than the bounded box with wrap turned on and off
            - Test larger than the bounded box with wrap turned on and off
            - Test on the edge of the bounded box with wrap turned on and off
        - Test Plans for Equivalence Classes
            - Test where the message value is inside the bounded box with wrap turned on. The return value should be false.
            - Test where the message value is inside the bounded box with wrap turned off. The return value should be true.
            - Test where the message value is smaller than the bounded box with wrap turned on. The return value should be true.
            - Test where the message value is smaller than the bounded box with wrap turned off. The return value should be false.
            - Test where the message value is larger than the bounded box with wrap turned on. The return value should be true.

- Test where the message value is larger than the bounded box with wrap turned off. The return value should be false.
- Test where the message value is on the edge of the bounded box with wrap turned on. The return value should be true - it's always true on the edge of the bounded box.
- Test where the message value is on the edge of the bounded box with wrap turned off. The return value should be true - it's always true on the edge of the bounded box.

- MessageFinderImpl#getMessagesFromQuerySnapshot()
  - For this method, there's only one equivalence class: the QuerySnapshot full of documentData maps are all transformed into the expected Messages
  - Boundary Analysis: there are no boundaries for this method
  - Test Plans for Equivalence Classes:
    - Test the QuerySnapshot turns into the expected list of Messages
      - Generate list of random document data
      - Build a mock QuerySnapshot that contain this list of document data
      - Call method to get messages
      - Assert messages size is the same as initial document data list size
      - Assert messages are as expected from the document data list

- MessageFinderImpl#getMessageFromDocumentData()
  - For this method, there's only one equivalence class: a returned Message with the data from the documentData map
  - Boundary Analysis: there are no boundaries for this method
  - Test Plans for Equivalence Classes:
    - Test the message is generated as expected
      - Generate random document data
      - Call method to get message
      - Assert message is as expected from the document data

- POST /messages/:user_id (MessageFinderImpl#findByUserId())
  - This method is a POST instead of a GET because the request library we're using on the android side doesn't allow setting a request body unless it's a POST.
  - There is only a single equivalence class for this endpoint: returned posts. We do not differentiate between users who have not yet posted a message and users who do not exist in our system - in both cases we

return an empty list of posts so they are part of the same equivalence class.

- Boundary Analysis:
  - Get all posts associated with a user id
    - Get Posts from the user, returning the list of messages for the particular user and the size of this list of messages. If there's no messages associated with that user id it returns an empty list.
- Test Plans for Equivalence Classes
  - Test the endpoint (equivalent to what we do for testing MessageFinderImpl#findByUserId())
    - Start a server
    - Create a new post with a specific user id with the new post endpoint
    - Send a request for posts for that user id
    - Verify that the returned posts is of size 1 and that the posts list contains the one post that was originally posted

Part 3: Branch Coverage

Continuous Integration artifacts for branch coverage can be found under our GitHub project 'Actions' tab (https://github.com/hwqin16/Magikarp/actions) by choosing the latest build from the list. Selecting a build will present an option to download the artifacts as a .zip archive.

Unit test branch coverage for the server portion of the application is 100%, disregarding trivial constructors and getters/setters. Unit test coverage for the Android application is over 90% disregarding these factors, but 100% was not achievable in practice. Several methods of the Android Framework components require a fully instantiated Activity lifecycle state, which is impractical on local Java Virtual Machine (JVM) tests. For this reason there are framework methods that are not unit tested, including:

- All Activity and Fragment #onCreate() methods (primarily stemming from requirement to call super#onCreate() which cannot be easily mocked with our mocking library, Mockito, due to having the same name)
- Framework instantiation of 'ViewModel' classes tying the view model to a specific Android lifecycle. Primarily this is done inside of previously mentioned #onCreate() methods. Actual unit testing of the 'ViewModel' classes themselves was generally at 100% branch coverage.

To get around these issues and maximize unit testing for code that needed to be in the #onCreate() methods, we had #onCreate() call another method #performOnCreate() which contained the testable code. In some cases, this led to a trivial #onCreate() method which simply looked like this:

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      performOnCreate();
}
```

Android Framework component methods can, however, be covered as part of instrumented (integration) tests on an Android device or emulator.

There is also an exception branch in the 'GsonRequest' class that was not covered with unit tests. There are two exceptions requiring a 'catch' block, however looking through the open source code for our particular framework implementation of the classes, it does not appear that the exceptions are actually thrown under any circumstance. Additionally, since one involved creating a Java String, which cannot be mocked by our mocking library (Mockito), it was very difficult to even simulate producing an exception. Given the expense of finding an alternative to cover a single branch, and the simplicity of the branch itself (one line of code), we decided not to pursue testing coverage in this specific case.

Part 4: Continuous Integration

Our Continuous Integration (CI) plan uses GitHub Actions to build and generate reports. The configuration file that manages the plan can be found at https://github.com/hwqin16/Magikarp/blob/master/.github/workflows/magikarp.yml.

Continuous Integration artifacts can be found under our GitHub project ‘Actions’ tab (https://github.com/hwqin16/Magikarp/actions) by choosing the latest build from the list. Selecting any of the most recent builds will present an option to download the artifacts as a .zip archive.