Date-A-Dog: Developer Manual

Date-A-Dog is a revolutionary new way to bring users in contact with local shelter dogs to take out on play dates. Users log into the application using their Facebook login information and are then presented with dog profiles based on the zip code they enter. Users can then swipe left to like or right to ignore the dog. Users can then view the dogs that they have liked and request a date with the dog. Shelters can log into a provided website and approve or deny date requests.

How to Join the Project:

- Code for the most recent/stable version of our application can be obtained from the master branch of the GitHub repository located at: https://github.com/Date-A-Dog/Date-A-Dog
- Developers looking to join the project need to:
  1. Join our Slack channel at: https://cse-403.slack.com.
  2. Make sure that you have Android SDK installed. Android SDK can be installed as a stand-alone application or bundled with Android Studio. Instructions for both installations can be found here: http://www.androidauthority.com/how-to-install-android-sdk-software-development-kit-21137/
  3. Clone our repo from our GitHub repository. First, fork from the master through the URL provided above. When time to merge, send a pull request to our team through Git.
- The application website can be located at: https://date-a-dog.github.io/
- The developer website for the application can be located at: https://date-a-dog.github.io/developers

Repository Structure:

- Within the repository:
  - 'app' folder contains code needed for the Android mobile application. The source files can be found in the 'src' subfolder.
  - 'web' folder contains code needed for the shelter web application.
  - 'dad-rest-api' folder contains code for the server and backend.
- Within the repository:
  - Readme and build files can be found at the root level.
  - Test files can be found in the test folder within the 'app' and 'web' subdirectories.

User Manuals:

- User manuals can be found at: http://date-a-dog.github.io/

How to Build the Application:

- To build the software:
  - After cloning the repo into your own branch, run
    './gradlew build'
    in order to obtain all dependencies and build the software.

How to Test the Application:

- The mobile application can be tested in a variety of ways:
  1. Test suite - the application will contain a test suite that will allow developers to run tests. These are automatically run when the Gradle build command is executed. Another way to execute the test suite is to run
     './gradlew test'
     to run the test suite without running the build.
  2. Additional tests – Users can write tests into the 'test' folder within the 'src' subfolder under each application category ('web' or 'app').

- The web application can be tested by:
  Our current approach to testing the website is manual, i.e. a developer goes to the website and checks functionality. As the project progresses we plan on implementing automated tests as we have done with the application side. The major reason for this delay is that the web app was added on as a last minute feature during construction of the SDS.

How to Set up the Daily Build and Test:

- The daily build is currently set up through Travis at: https://travis-ci.org/Date-A-Dog/Date-A-Dog
- A build command is run each time code is pushed to the repo. Build errors are automatically emailed to the Product Manager and the current team of developers.
- Unit tests are also run on each push. Test results can be viewed along with build results at: https://travis-ci.org/Date-A-Dog/Date-A-Dog

How to Release a new Version of the Software:

Each time a new version of the software is ready to be released, the following steps need to be taken:

1. Ensure all working features have been implemented correctly with unit tests.
2. Merge all working code into master branch.
3. Ensure that build is passing without failure.

4. Update all documentation and code references to version number with the updated number.
5. The application website located at:
   https://date-a-dog.github.io/
   Contains links to download a .zip or a .tar.gz file of the most recent version, automatically pulled from the master branch of the GitHub Repository.

Code comments:

- All developers contributing to the project must satisfy these comment requirements:
  1. At minimum, for every file, there must be a comment describing the general purpose of the file and how it fits into the overall design.
  2. Data structures should contain representation invariants and procedures testing the invariants.
  3. Every method requires at least a one sentence comment describing what it does.

Bug Reporting:

- Users (both daters and shelters) submit bugs to us via email at: dateadogapp@gmail.com
- Outstanding bugs can be tracked through GitHub Issues at: https://github.com/Date-A-Dog/Date-A-Dog/issues
- We use the Mozilla guidelines for all bug report submissions. Guidelines available at: https://developer.mozilla.org/en-US/docs/Mozilla/QA/Bug_writing_guidelines

Design Pattern: Singleton

What is A Singleton?

The singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects.

Why Do We Have Singletons?

In the Date-A-Dog Project we have two instances of Singletons so far.

1. The first is that the DADAPI is the class that makes https requests using volley to communicate with the RestAPI to receive data. We need this to be a singleton because it needs to be an object that we can call from any fragment or activity in the UI to receive data about dogs that are swipe-able, liked dogs, and get shelter profile info for the dog. Each user has a DADAPI object associated with his/her account. Each DADAPI singleton is associated with each

user so that specific info about their likes dogs and date requests are managed. The singleton is instantiated in MainActivity.onCreate() in the package dateadog.dateadog.

2. The second is that for volley, to receive, JSON, which is the form in which data is received, we have to create a request queue to request data from the RestAPI. Since our application makes constant use of the network to the RestAPI, it's most efficient to set up a single instance of RequestQueue that will last the lifetime of the app for each user. The use of Singleton can be viewed in Singleton.class in package dateadog.dateadog.

Design Pattern: Model-View-Controller (MVC)

What is Model-View-Controller (MVC)?

Model-View-Controller (MVC) is a design pattern for implementing user interfaces on computers and mobiles. It divides a given software application into three interconnected parts, which are model, view and controller.

Why do we use MVC?

We are using Model View Controller Design Pattern in Android App part of our Date-A-Dog project. DADAPI, Dog, User, DateRequest, ShelterProfile, and LinearRegression classes are model of our design. These classes directly manage data of the application. All layouts associated with activity and fragment classes are view. All layouts are included in Android XML files, such as activity-login.xml. Also there are two views included in dateadog.dateadog.tindercard package, which are BaseFililngAdapterView, SwipeFilingAdapterView and FlingCardListener. Controller is between model and view. It takes input from user and updates the model. It also updates the view. All activity and fragment classes are controller in our design pattern. They change both model and view. LoginActivity, MainActivity, DogSwipeActivity and LikedDogsFragment are controllers.