**Test Plan**

**Team Pooch**

1. **Test Philosophy**
   1. **Can you test everything you can think of? What criteria are you using to select what gets tested?**

Yes, we tested everything we thought of. Everything from registering for a new account, signing in to your current account, all features on the navigation bar, adding new profiles, updating profiles, uploading documents, string lengths of the text fields in forms, the logout button, has undergone testing and reviewing. The criteria we used for the features to be selected are based on the user’s usage and expectations of the features. User surveys were held, asking users their expectations, and accordingly tests were made.

* 1. **Are there different densities of testing across your code? Will less well-understood code go through more testing? Is there code that does not require testing?**  
     Yes, there are definitely different densities of code testing, but not a huge variation between the different levels of code complexity. One of the important goals for the team was to test every feature, thus testing all aspects of the released code was a priority. The main feature functions of the application like dog walkers, document uploads and user profiles were tested comparatively more than other features of the application. There was a little lesser focus on simpler features like user log in, signing in and logout. For lesser understood codes, like machine learning, there definitely have been more tests than simple portions of the application.
  2. **Why would your selected testing program be sufficient? What criteria do you use for making that decision?**  
     We believe our selected testing program is sufficient because we covered all the features which the website provides. The current testing methods like unit testing, integration testing, and regression testing covers all the components in our application. The criteria we used for testing was prioritizing the important and most used features first, and the add-on performance features later. The Wow and must-have features were definitely given a priority when it comes to which feature will be tested first.

1. **Team Workflow**
   1. **What is your team’s test process?**  
      The testing process was to test a function/feature as soon as we finish it. Before pushing it to the repository, the developer tests it, and then a second developer tests the same code before the feature is marked tested. All the functions are tested individually, and then finally an integration test is performed on all the functions, so we could test the application as a whole.
   2. **How do you determine code coverage?**Code coverage is a measurement of how many lines/blocks/arcs of the code are executed while the tests are running. Testing can either be done manually or automatically. For our web application, we will be testing every aspect of the web application manually based on the use cases. This would cover 100% of the code because we wrote the code from the test cases.
   3. **Do bugs come back from users or from your own team affect your test program workflow?** **If so, how is it implemented?**  
      Sometimes, bugs do come back from other team members or even from users. Our team constantly monitors bugs, and if any bug was reported to us directly from users, that bug is tested first. The newly added bug is added to the testing list, it pushes the other testing a bit, but in general, it does not affect the overall workflow. Whenever it is someone’s chance to test code, the developer picks the first from the list, and the process continues. Bugs reported by the user and the bugs from priority features are definitely considered first. The bug, in general, does not affect the workflow, because the bugs will generally be anticipated in advance by the team.
2. **Test Design**
   1. **Code paths**We use Basic Path Testing. This is a white box testing method that defines test cases based on the logical workflow of tasks, which a user takes to reach a certain feature, then use the feature and finally exit from a feature. It involves the execution of all possible blocks of the program and achieves testing of the maximum path coverage with the most efficient number of test cases. This is similar to branch testing, as certain features do not have a definite one line path, rather branches to reach the end. In the case of branches, each branch has various test cases, and those are tested individually. For example in our web application, while signing up for a profile, a user may be a dog walker, dog boarder or dog owner. So we develop test cases for each of these branches for individual testing.
   2. **Code coverage estimation in terms of % lines of code tested (related to (a) above)**  
      Approximately 100% of the code can be tested since we test the code based on the use cases.
   3. **Environment**

We used Visual Studio Code to create our web application. So our environment for testing would be VS Code and the web application itself.

* 1. **Test code and data**  
     Since we are developing a web application, we have to test a lot of UI elements and functionality. We test our web application by inputting the wrong data as well as incorrect data to check how the web application responds to it and fix bugs based on it, this is similar to the unit testing method. We randomly generate data using online tools through google to use it for testing various features of our web application.
  2. **Version control methodology (of your product code, test code and test drivers. Do you track bugs?)**

We have a development branch and a master branch. We first commit our changes to the development branch. After we see everything is working, we commit it to the master branch.

1. **For every test case:**
   1. **Requirements or behaviors, i.e. what your tests are trying to verify (requirements) or validate (user needs) for every test? Is there a more systematic way to establish this?**

For every test case, one of the basic test requirements is, if the code is working or not. All the current versions of the code have been tested using all the use cases, and whenever a feature is updated or added on, the whole use case is tested again. All the requirements and behaviors of the use cases are being tested. The primary features of testing are the goal of the feature, followed by its functionality and finally on the basis of what a user expects from a certain use case/feature.   
For example, test functionalities for all the required portions for each use case, and launch the features for users to use, then test for added functionalities and user expectations. As far as we know, there is no better systematic way to establish this testing as our web application contains a lot of features. If we, as developers test the whole use case and then release a feature, then it will cause a feature launch delay and to keep the feature launch on track, we follow this approach.

* 1. **Evaluation criteria (e.g. pass/fail) and disposition**

We determine if a test case is passed or failed based on the outcome of that test case. We consider the test case to be passed if our web application works how we intended it to work. We immediately fix the bug so that it passes the test case in case it fails. We are developing our web application mainly using ReactJS which helps in developing our web application with components that can be reused. Reusing code is a good software development practice that would save a lot of time for developing as well as testing the code. The usage of components reduces the number of test-cases since we are reusing the components in different pages and we just have to test it once.