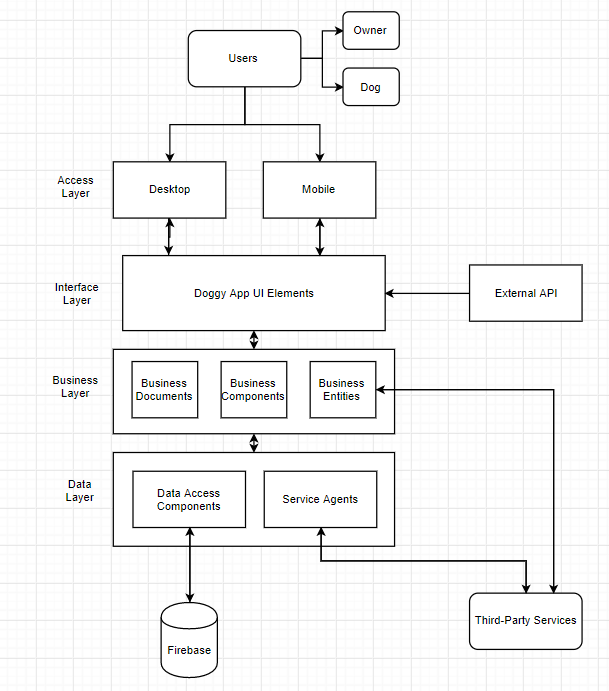
Architecture and Design Document

Table of Contents

1. System Component Diagram
2. Quality and quantity standards
3. Analysis Diagrams
   1. Use case diagrams
   2. Activity diagrams
   3. Sequence diagrams
   4. Data flow diagrams
4. Objects and Methods Identification
5. Design Patterns Architecture pattern and language (architectural decisions)
6. Architectural Alternatives
7. Trade-off Analyses
8. External Server Application
   1. Capabilities
   2. Interface design
9. Machine learning
   1. Business Problem
   2. What features/labels do the model training need for prediction?
   3. Input Data Source
   4. Project Plan/Approach for training data
   5. Model- Convolutional Neural Networks(CNN)
   6. Deployment plan
   7. CNN memory and computation requirements
   8. Tradeoff analysis of deployment models
   9. How do we improve the accuracy of the model?
   10. Model is an underfit or overfit?
   11. Performance matrix of ML capability
   12. Does ML meet the business need in terms of metrics?
10. System Component Diagram

The system component diagram used a layered architecture approach. It is one of the most common styles used in the software development life cycle. It illustrates how different layers are wired together to make the application work together as a whole. The system can have an n-tier architectural style, our component diagram is a 4-layer architecture layer called access layer, an interface layer, business layer, and data layer. This style is divided into various horizontal layers and each layer has some specific function which eventually combines to make the application function as a whole.

1. **The access layer:**   
   The access layer basically tells what are the ways one can access the application through web/ laptop and also look it up as a link on mobile devices.
2. **The interface layer:**  
   The interface layer has all the UI elements of the application and an external API Google API comes into the picture to help the user sign in the application.
3. **The business layer:**  
   The Business layer includes all the business documents like Business Requirements Document, Product Requirement Document, Management Plan, and Design Architecture Document. Business entities and our relation with the entities must be constant and frequently updated in order to provide better services to the users for our application.
4. **The data layer:**  
   The data layer has two major components: the data access components and the service agents. Data access components come from the initial database we have, which contains the owner profile and also the pet profiles and any other relevant data we have included in the database. The service agents are basically third-party services like vet services, grooming services, and dog product buying shops.



1. Quality and Quantity Standards

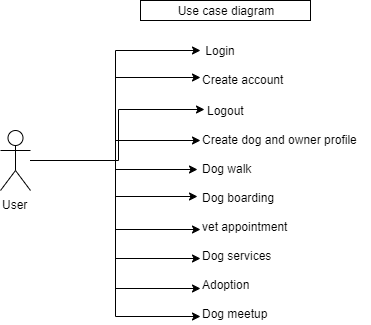
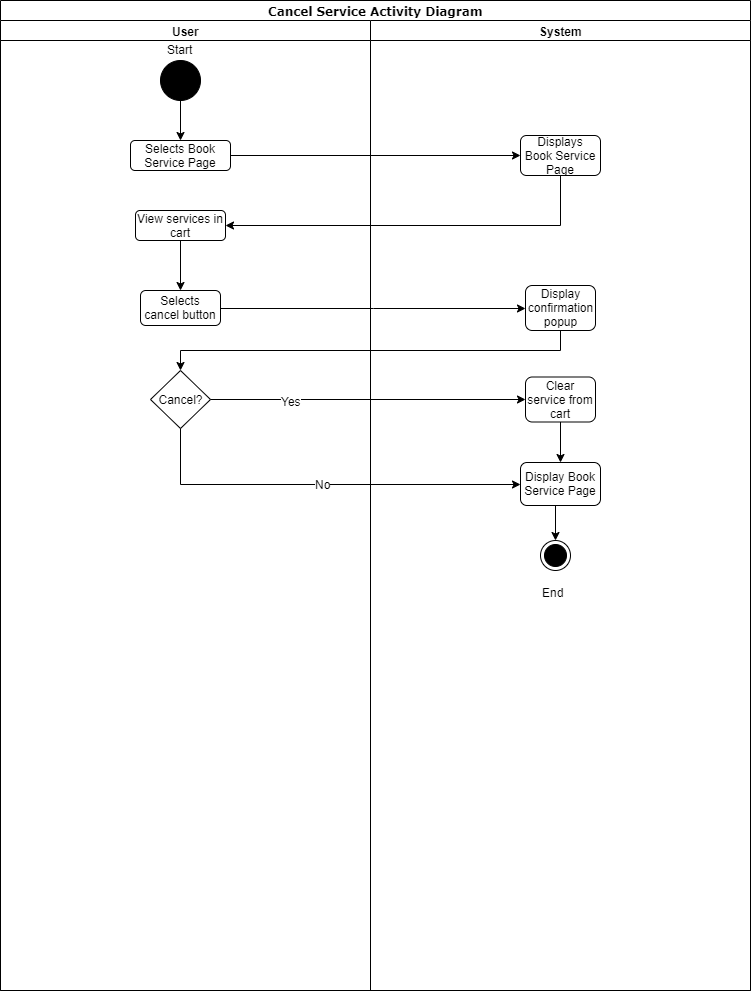
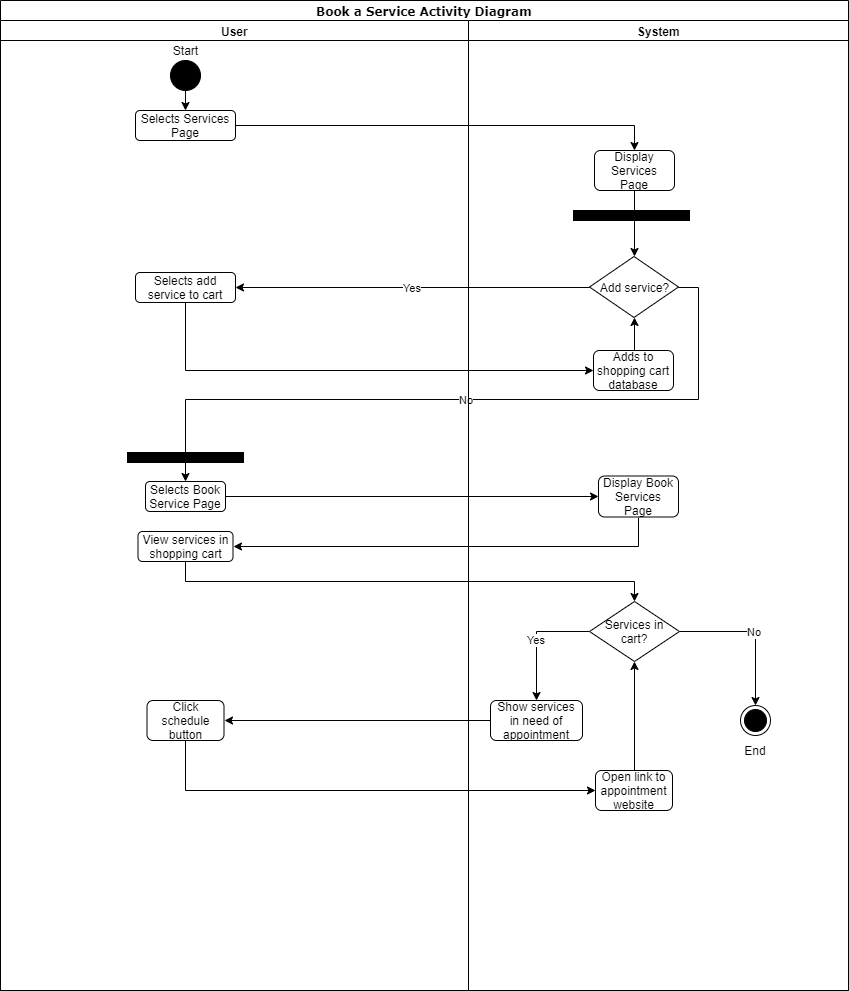
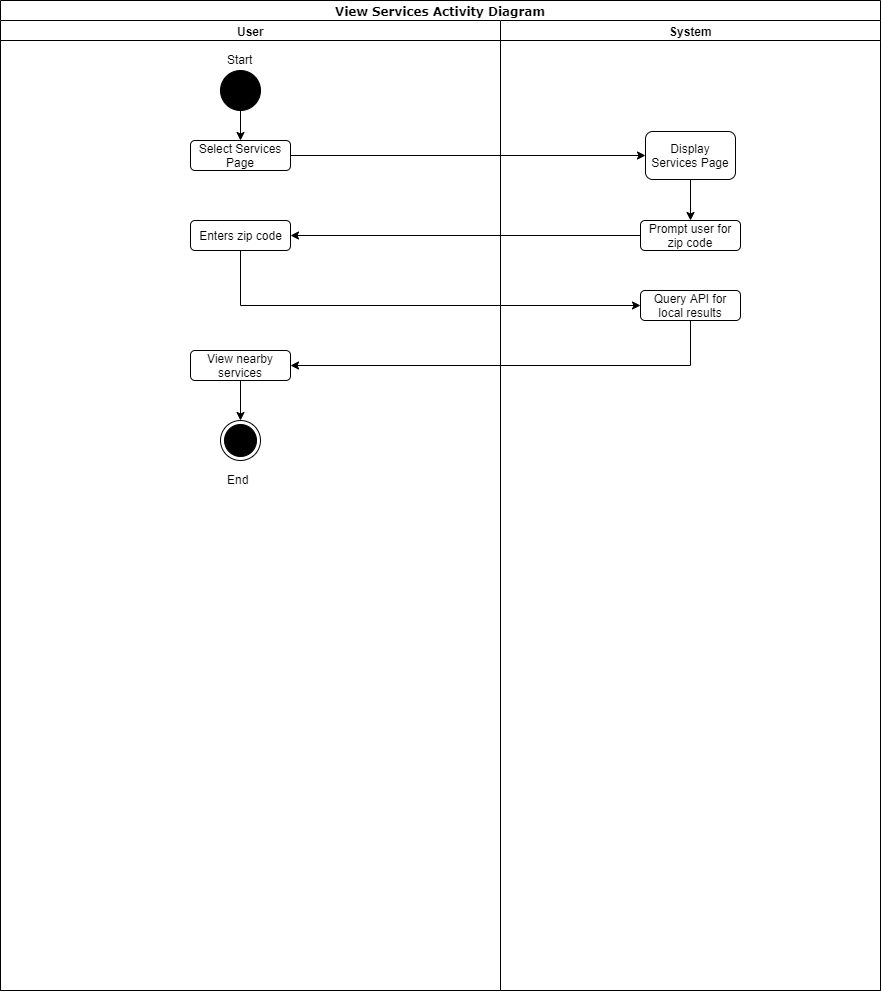
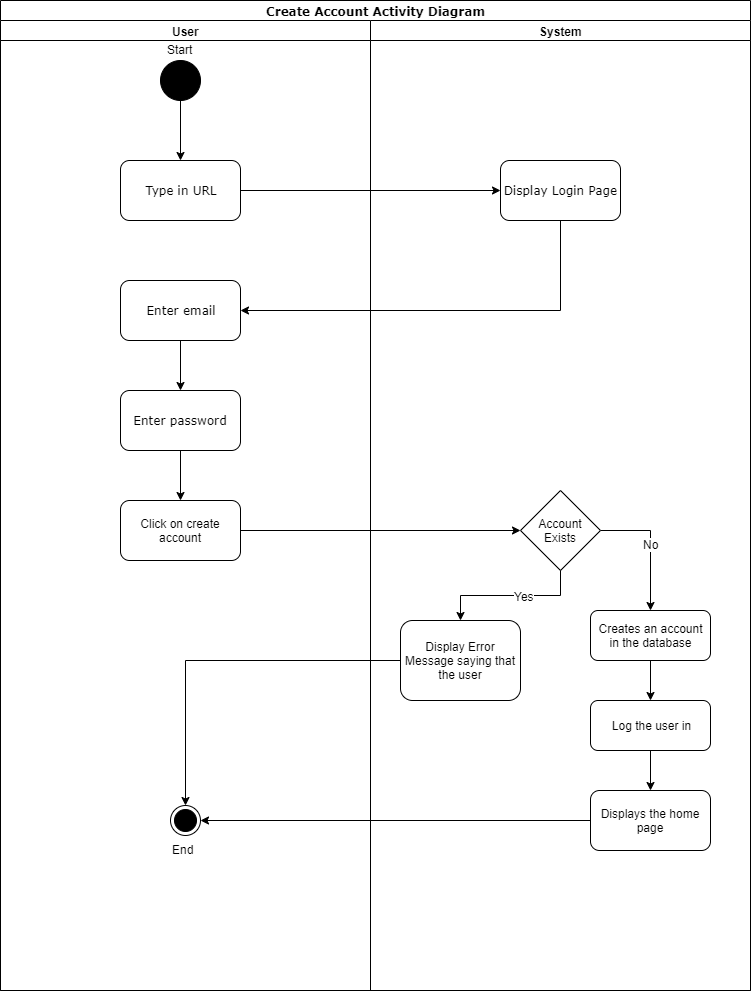
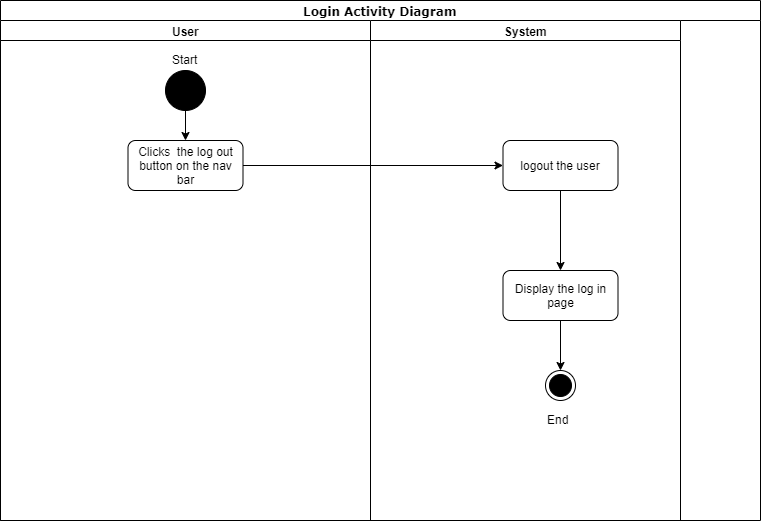
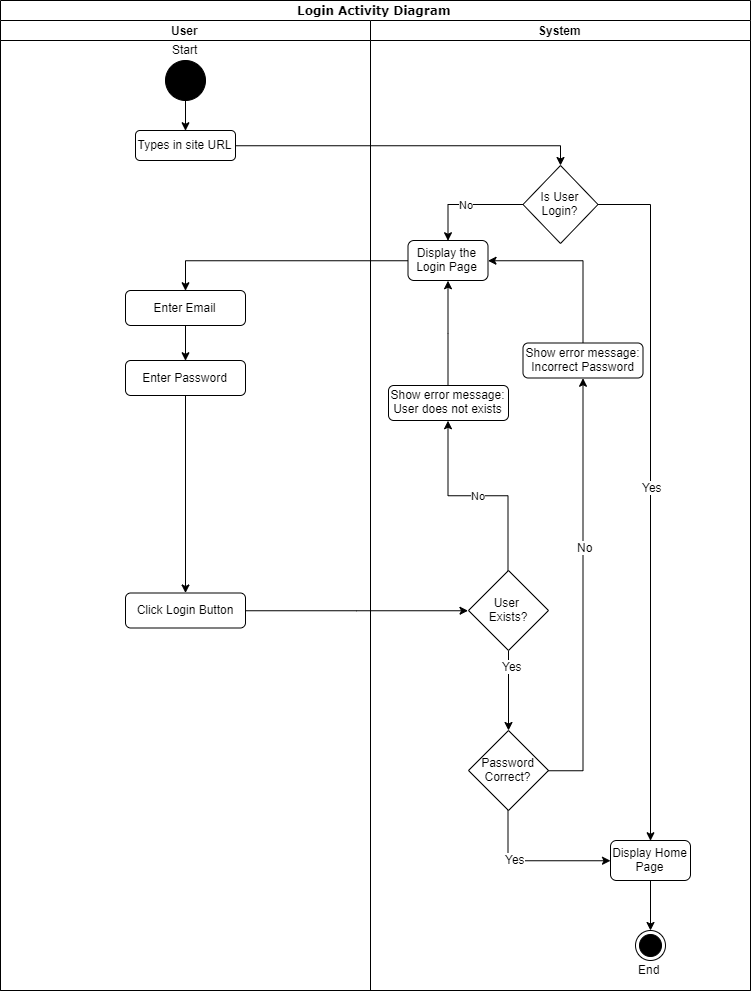
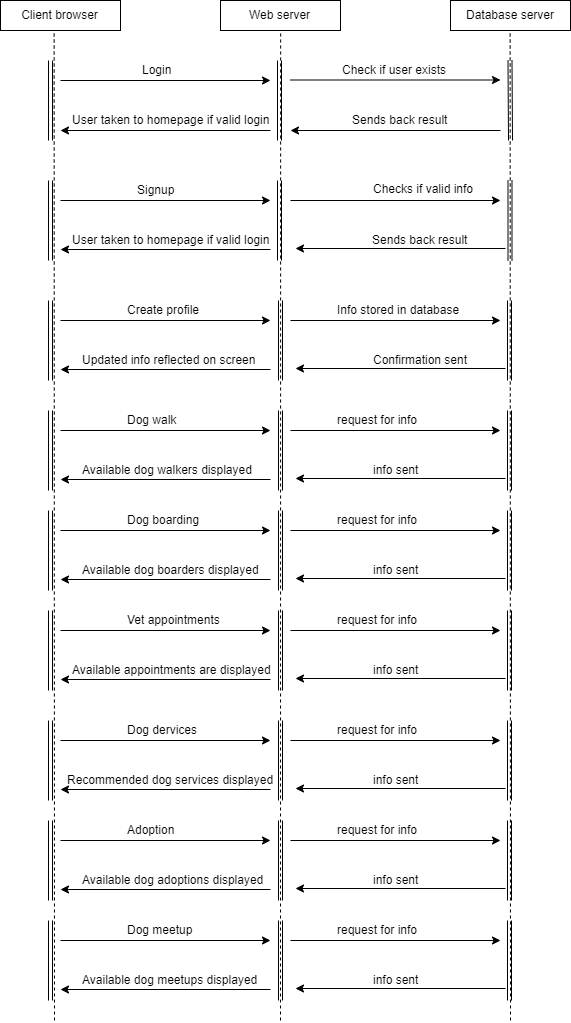
Large business organizations with a focus on the user interface, use this architectural style. Because of the layered approach, different teams can be allocated to work on different layers.

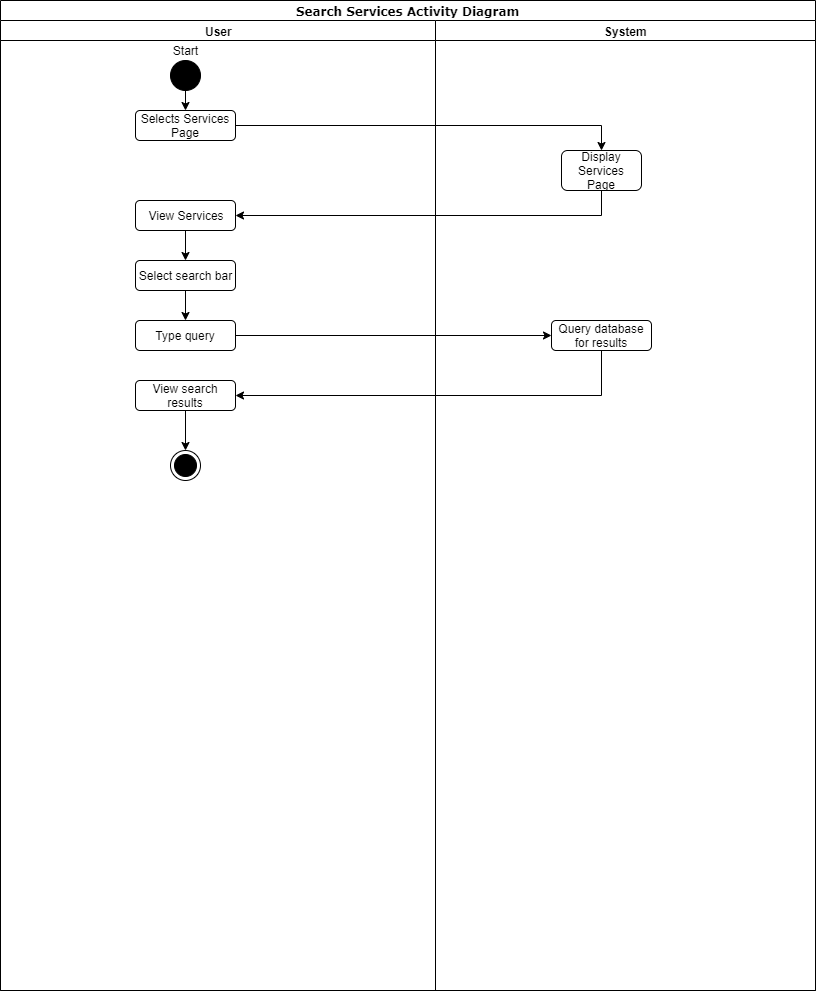
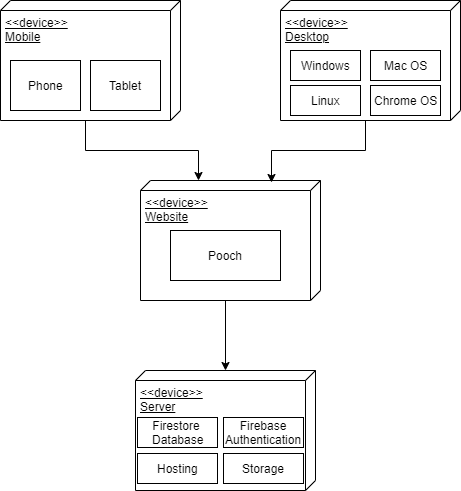
One advantage of the layered structure is the re-usage of lower-level layers. Certain lower layers can be used by different higher layers. Layers make standardization easier and we would be easily able to distinguish between the different layers and their functions. Changes can be made within a specific layer without really affecting the other layers.

It makes the addition or modification of functions and modules easier because we can edit the functionality of a particular layer without affecting much of the other layers.

Our layered architecture is integrated with a client-server type of architecture. This helps us to divide tasks into threads which makes it easy to model the services requested by the user to be handled faster. We expect a lot of clients to be on our application, requesting different servers at the same instance, so splitting tasks into smaller threads makes it faster to process the request and makes a shortened wait queue for the to be done tasks.

The only drawback to this architectural pattern is then certain layers may have to be skipped in certain situations. Like while inputting the data into the database, we as administrators do not really need to know the access layer devices.

1. Analysis Diagrams
   1. Use Case Diagram   
      
   2. Activity Diagrams:  
      
   3. Sequence Diagram:  
        
      
   4. Data Flow Diagram:



1. Object and Method Identification

|  |  |
| --- | --- |
| **Object:** | **Implementation:** |
| User | Dog Owners, Dogs |
| Website | Pooch Web App |
| Database | Firebase |

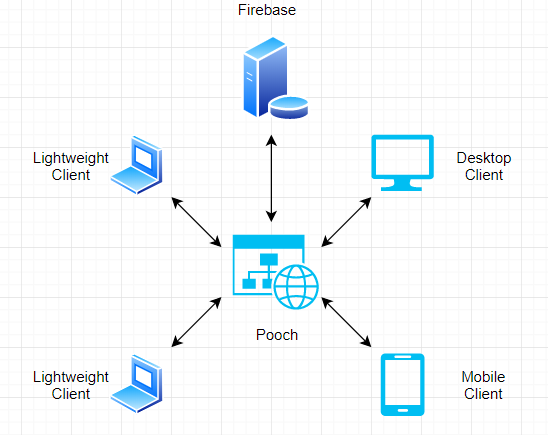
|  |  |
| --- | --- |
| **Method:** | **Implementation:** |
| User Story #1 | Sign in with social media |
| User Story #2 | Sign in |
| User Story #3 | Sign up/Add profile |
| User Story #4 | Visit home page |
| User Story #5 | Logout |
| User Story #6 | Navigate through pages |
| User Story #7 | View services |
| User Story #8 | Search services |
| User Story #9 | Book services |
| User Story #10 | Cancel services |

1. Design Patterns  
   **Server-Client Architecture**

We are using Firebase to develop our web application. Firebase utilizes a server-client architecture. Firebase runs on javascript and has SDKs available in Node.js, Java, Python, and Go. Server- client architecture is good to model a set of services where clients can request them. By using this architecture, we will be having a scalability advantage. We can add resources in the form of network segments, computers, and servers to a client-server network without major interruptions to the network. Access to any new resources can be administered from the centralized security database, stored on a single network server. With a centralized server, permissions to all network resources can be granted by a smaller number of support staff configuring those permissions on the server. The cost is an advantage because fewer staff are required to maintain the network and maintain access to network resources. This is also a fail-safe system having backup servers and therefore, the application will never go offline due to server failure.

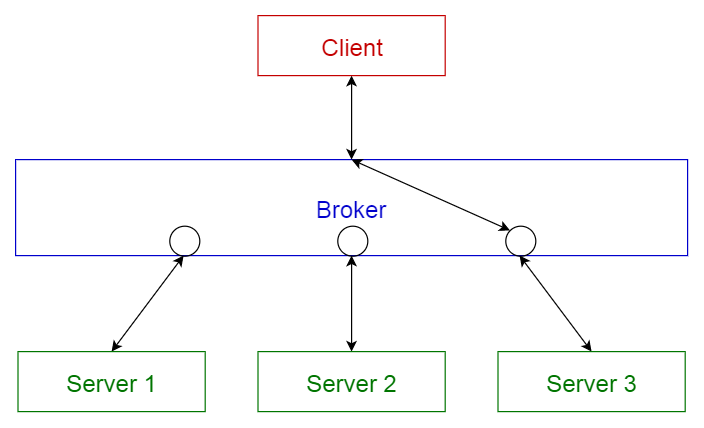
**Alternative architecture model: Layered architecture**

The alternative architecture design we are considering is a layered architectural design pattern for our project because layers make standardization easier as we can clearly define levels. If we use layered architecture, changes can be made easily within the layer without affecting other layers.



1. Architectural Alternatives
   1. Broker Pattern

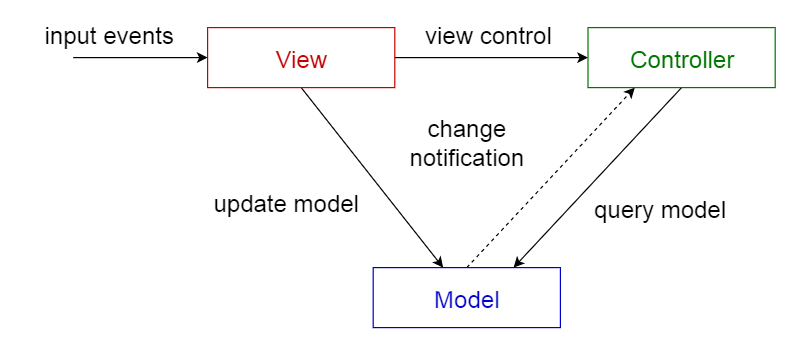
This pattern is used to structure distributed systems with separate components. A broker is responsible for interaction between major components. The server publishes its capabilities to a broker. The client requests a service from a broker, the broker redirects to the appropriate service.  
  
We plan to not use this pattern because:

* We do not have multiple instances of servers for different services, thus this pattern would be very ineffective to use.
* Message broker software is: Apache ActiveMQ and RabbitMQ, unfamiliarity to this software will make it difficult to work with.  
    
    
  1. Model-View-Controller Pattern  
     Three main parts to the interactive application:

Model: Contains main functions and data

View: Displays information to the user  
Controller: Handles user input.

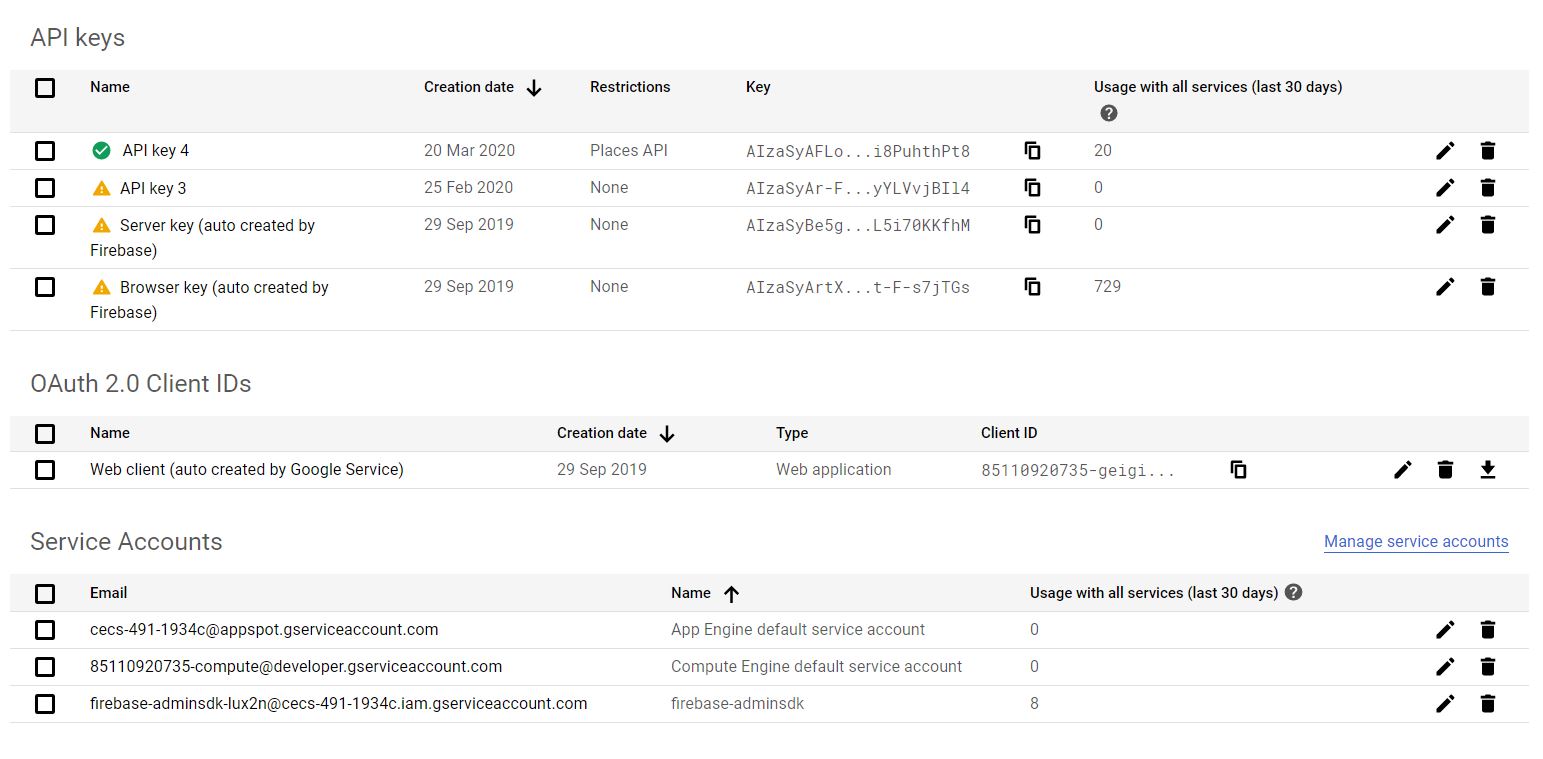
This model is used when the internal representations of information need to be kept separate from what is being presented to the user.   
  
  
We plan to not use this pattern because:

* It works best with web frameworks like Django.
* It increases the complexity of the code and may also lead to an unnecessary number of user updates for every small change made, for specific user actions.
* Considering this web application provides so many features, it is not the best idea to send updates to the user for every minor change made by developers and admins.   
    
  

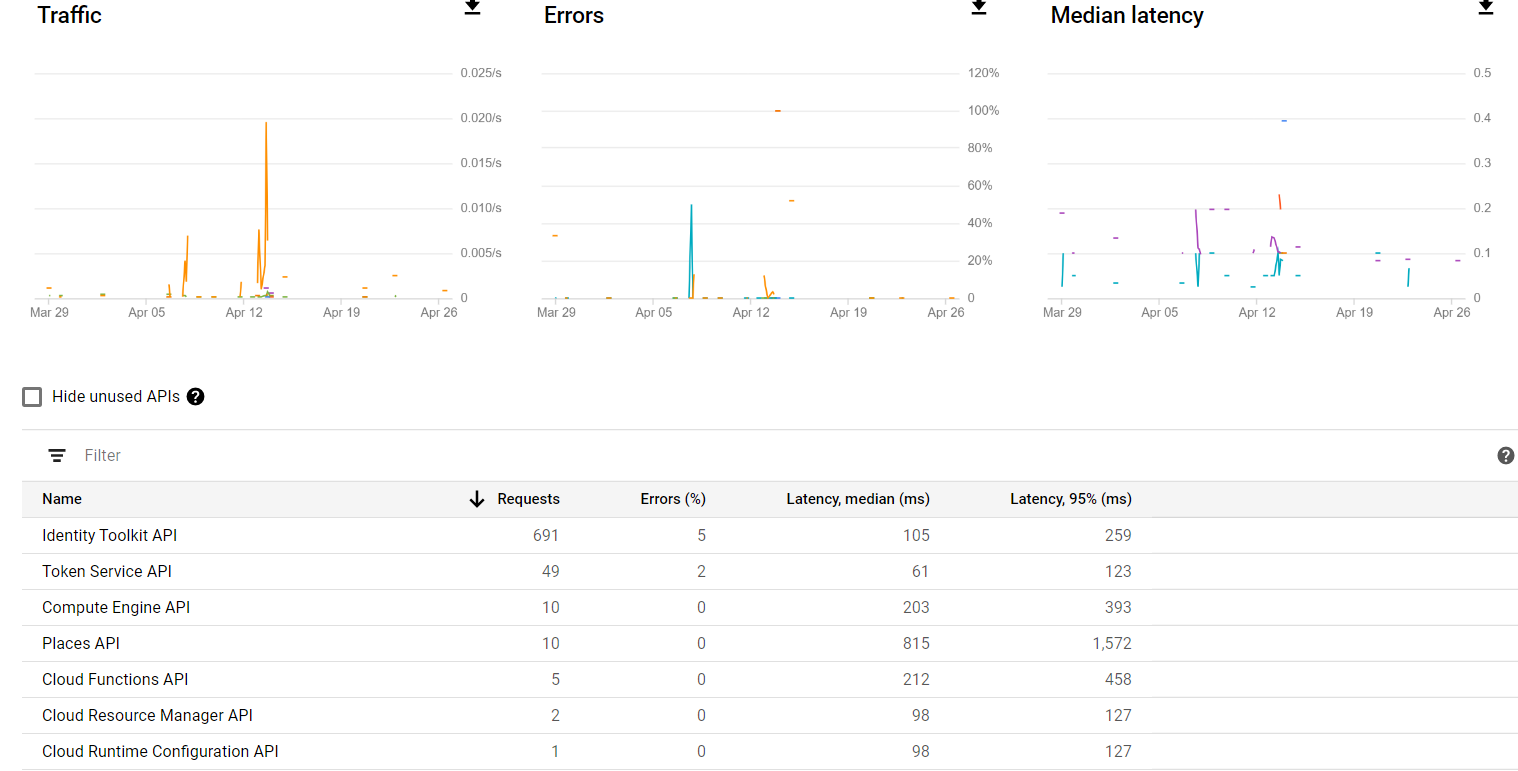
1. Trade-off Analyses

|  |  |  |
| --- | --- | --- |
| **Decision:** | **Benefit:** | **Cost:** |
| Use Firebase for the backend of the application | Hosting and authentication external to the application. Less work for developers. Better security. | No control over hosting or ownership. Must trust Google to protect user data. |
| Create a web app instead of android application (more work) or desktop application (less work) | Accessible from any device | Not accessible without the internet. May not be formatted correctly for all mobile devices. |
| Link to services instead of charging customers on services’ behalf | Increased scalability. Faster distribution. | Loss of potential profits by taking a cut of revenue directly |

* 1. API Choices:
* Google API for signup and login for every user.
* This API would also be used to find nearby dog grooming and dog walking services.
* We are using Google Places API to find the nearby dog services for the user to make it easy for the user to know the services nearby without any hassle of looking for service nearby from other applications.
* The application also uses API keys for Firebase server and firebase hosting browser.



* The Google Place API’s usage can be tracked using the Google Cloud Platform (GCP) and a private key was generated for using the API, using that private key, we have implemented our features pages “Nearby Dog Services”. The implemented Google Places API has the traffic, errors and latency as shown below.
* This API in itself, calls in a lot of other supporting APIs like Identity Toolkit API, Token Services API, Compute Engine API, Cloud Functions API, Cloud Resource Manager API and finally Cloud Runtime Configuration API.
* The identity toolkit API lets the developers use their systems with open standards to verify a user’s identity.
* Token Service API lets the developers to allow their users to log in and log out anytime, without having to verify their identity after every usage, primarily used for user authorization.
* Cloud Functions API manages lightweight user-provided functions executed in response to events. This is used for our dog services page.
* Cloud Resource Manager API helps us create, read and update the metadata for GCP resource containers.
* The Runtime Configuration API allows the developers to dynamically configure and expose variables through GCP.
* All these API are used to support the Google Places API, and have importance in the overall experience Pooch offers.



* 1. Cloud Decisions:

-All user data is stored in the cloud, and hence it requires no additional space on personal computers.

-All service information and vendor contact information is also stored in the cloud.

* 1. Security Decisions:  
     -Fire Authentication will handle the security for user accounts

-User data is stored on and secured by Google servers

-Concerns the risk of sharing state among different components.

* 1. Logs / Monitoring Devices:  
     -Machine learning better machine

-The design should be as simple as possible

-Security should not make worse the user experience

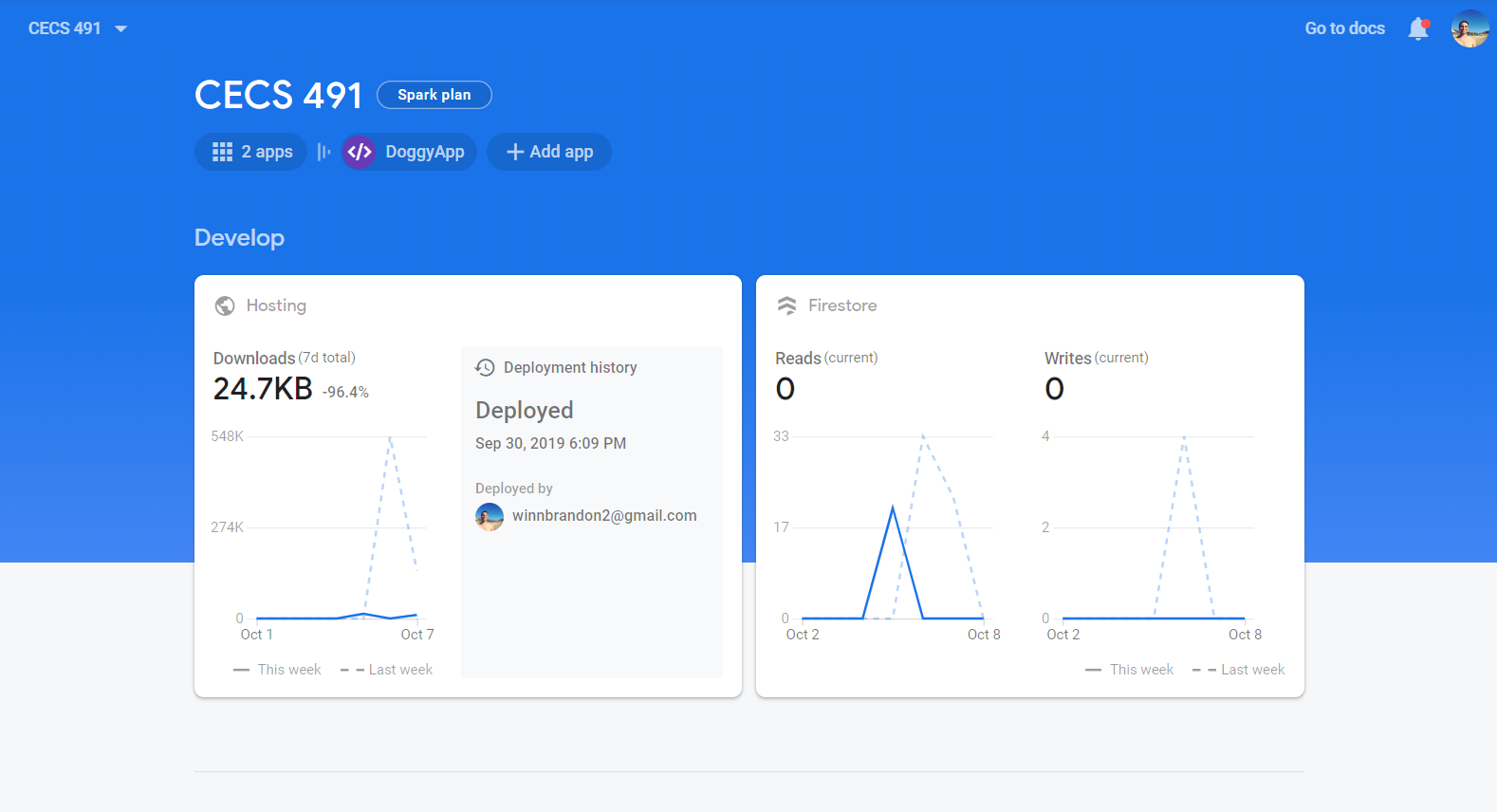
1. External Server Application  
   **What is Firebase?**



-A realtime database and website hosting service, owned by Google, to simplify the backend for web developers. By using Firebase, developers can focus on the UI and application logic, without having to worry about implementing their own security or database rules.

**What capabilities does Firebase have?**

-Firebase simplifies the login process by enabling developers to easily implement social media sign-in functionality. The database aspect allows for the creation of collections, tables, and documents in a NoSQL server. The Firestore cloud storage access allows developers to store limited files in the cloud that are relevant for their application. This can entirely mitigate the need for users to download anything locally for the web app to run properly. Hosting by Firebase means developers of small applications do not have to pay for website hosting. The functions section will not be used for this application, but allows quick access to customizable Firebase cloud console functions. Finally, the machine learning kit grants developers access to many Google machine learning tools.

**What is the goal of Firebase’s interface design?**

-The interface of Firebase allows tight integration between the users’ data, the developer’s application, and other Google services.

* 1. Capabilities
* Our server would be capable of providing facilities for both web applications and mobile website viewing. This gives us multiple server environments to run our application.
* All the components must be able to perform in the same environment as their web servers, and their main job would be to support the building up of dynamic pages well.
* It should be capable of handling load balancing well enough so that we as developers can help and focus on the business aspect of the application better.
* The administrative code would be able to properly deploy, manage all the layered components of the application.
* React makes it easier for the application to have a front end framework while still running on a back end system.  
    
    
  1. Interface Design

-The layout of the UI in the first release is based on the wireframe mockups from the Product Requirement Document which take into consideration three factors.

1. The web app must present a clean, professional look. No unnecessary clutter is allowed.
2. There must be a convergence between desktop and mobile design. This prevents the need to double the UI workload for the developers and prevents user frustration when switching from a feature-rich desktop experience to a trimmed-down and potentially feature-lacking mobile version.
3. No feature should be more than three clicks away. The application should maximize routing so all pages are easily accessible. New users should be easily able to find what they are looking for. This will reduce the learning curve and could help boost user retention.
4. Machine Learning
5. Business Problem

Our web application provides service only to dogs. A lot of people consider dogs and cats as equal. Therefore, there might be situations where users might upload the picture of their cats for the services offered in our application. In order to stop the users/spammers from updating the picture of random pets, we need to validate the picture that they upload. A lot of applications are facing this kind of problem where the users create fake profiles and add random pictures. We believe we can at least take a step forward by recognizing whether a person is updating the right image of their pets. The benefits of using this machine learning model would be for validation of the dog’s images. It is necessary to restrict the creation of fake profiles so that people there would be only genuine requests for dog services.  
  
This model is definitely required since it is of great benefit to our users. Our users will have more reliable dog walkers, dog service providers, dog boarders. Most importantly, the dog meetups scheduled by other dog owners will be more legit, if only the people will actually be allowed to create dog profiles in our application. If only the legit dog owners can create accounts, then only they can create meetups, and hence, it all ties back together to, the owners can be carefree and non-hesitant for using any features of our application. How this business idea benefits our business is that, if we validate the profiles when they are being created, then we would never have to manually go back and give the users a rating, or check if something is not in place or something is wrong, unless a user complaints about another specific user. This all eases the identity verification/validation process on the developer’s end.

1. What features/labels do the model training need for prediction?

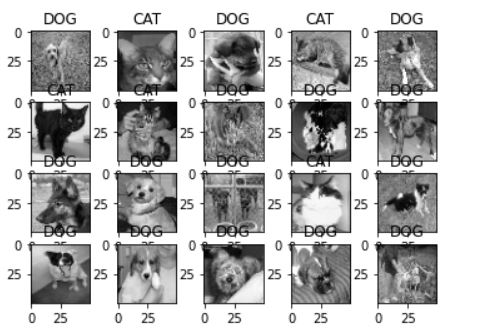
We will be using a dataset of dog and cat images to train and test our model. Our model will be trained on two features in each data sample. The two features are the actual image and the dog / no dog label. Two labels are needed to perform a supervised learning model for Machine Learning. The x-axis is the uploaded or inputted image and the y-axis is the binary labels of “dog” or “cat”. The label selection matches exactly our business case because the theoretical part of the business problem is as simple as distinguishing if the uploaded image is a dog or not. Hence, our label selection is the image input from the user (or initially from our dataset) and the output label “dog” or “cat”.

1. Input Data Source

We gathered the data to train and test our machine learning model from an online dataset which consists of images and labels. The online platform we used to get the data was Kaggle Database which provides a large dataset for dogs and cats.   
The link for the online dataset is: <https://www.kaggle.com/tongpython/cat-and-dog>  
This dataset contains over 25000 training images and about 12500 test images with adequate labels. The data available in this dataset seemed sufficient enough to start an ML model, and hence no more data was manually added to the dataset.

1. Project Plan/Approach for training data:

There are mainly two features for the images we got from the Kaggle dataset. They are the actual images and the label for it which says if it’s a dog or not. The training data set contains over 25000 images of dogs and cats. Depending on the dataset available in Kaggle we will receive a 0 or 1 (a binary result) as the output from training our ML model.   
The data that we got from the kaggle database was first explored by pre-processing and resizing the image to 50x50(square image) and each of these images in the training dataset was converted into grayscale and was printed to explore the training dataset graphically.   
  
All the images were converted to greyscale, to decrease the load time of the whole set of images and faster computation. But grayscale conversion caused no significant data loss, hence it was a wise decision to take. All the images were then resized to be 50 x 50 because we wanted uniformity in the images before applying CNN filters to any layer. The Kaggle dataset had images uploaded with different resolutions, hence this uniformity was essential for more accurately precise data.   
  
We created a helper function to create a list of the training data with the image and its respective label. Even the original labels for the images were “Dog.1.png” or “Cat.34.png” something like this. Hence, we had to use the split function on all the images, so the images remember their new labels as “Dog” or “Cat”. This all resizing, renaming and greyscale were decisions made while preprocessing the data from the dataset.   
  
The following is an image of preprocessed data and exploring the input data/images.



For training our machine learning model, we will be using Relu as our activation function. This activation is primarily for preprocessed training data.

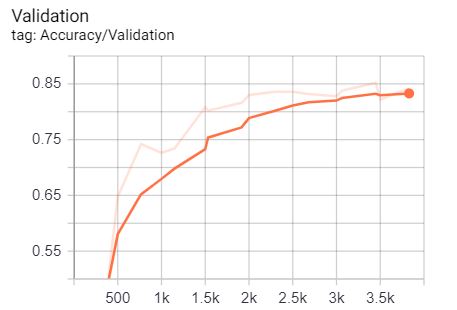
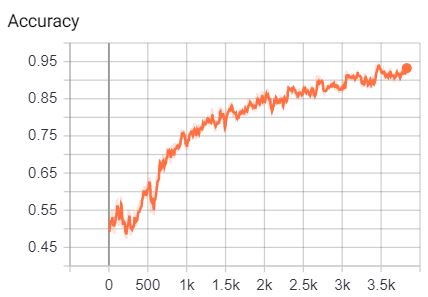
From our training dataset, we will be using the last 500 images as our validation dataset. The validation set can also be called as the test set amongst our training set. The ML models are generally vigorously tested on the training data itself, without them being exposed to the test data, hence this decision. These 500 images are purely used as a validation dataset and are not used for training the model. This is used to check the accuracy of our trained model. This validation set is essential, to achieve uniform accuracy and no unexpected large errors when our model is suddenly exposed to new test data from the real world. So, our complete dataset is split like training and test data, while there is a subdivision in our training data as training and validation data.

1. Model- Convolutional Neural Network(CNN):

CNN is a clever way to reduce the number of parameters. Instead of dealing with a fully connected network, the CNN approach reuses the same parameter multiple times. Neural networks are a set of algorithms, modeled loosely after the human brain, that is designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling, or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text, or time series, must be translated. The big idea behind the CNN model is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model.

For our CNN model, we plan on using the images of size 50x50 pixels to make our training phase faster. For training our model, we are using a filter size of 3x3 because the images of dogs are similar and we need more accuracy. We are using a total of 8 convolution layers in which 3 layers are of size 32, 2 layers of size 64, 1 layer of size 128, 1 layer of size 512. We chose the number of layers by test-based development and we reused layers to reduce the loss in the initial layers. We originally started with 3 layers and then added 2-3 layers depending on what accuracy we were getting from those layers of convolution. Filters were selected on a trial basis, depending on the suggested filter sizes and our need for a desired level of accuracy.   
  
We chose the architecture by looking at graphs in the tensorboard and choosing the right number of epochs. We mainly look for the validation loss and decide the pipeline of our model from that. We randomly added layers and tested for the accuracy and loss and decided the layers based on how well our model worked. After each layer, we max-pool with 3x3 filters. We then fully connected it with a dropout of 0.8 to improve validation accuracy and reduce the loss initially.   
  
We use softmax activation in our model. We are using Adam as our optimizer because it is straightforward to implement, it has little memory requirements, computationally efficient, and invariant to diagonal rescale of the gradients. We initially started the epochs with 1 and kept going until 10 (hence the final model has 10 epochs) and checked the different accuracy and losses of our model in each epoch. We chose categorical \_crossentropy as our cost function because it’s non-negative, that is,c>0. The result of it would be between 0 and 1.   
  
Another advantage of using cross-entropy is that if the neuron’s actual output is close to the desired output for all training inputs, x, then the cross-entropy will be close to zero. These are both properties we would intuitively expect for a cost function.

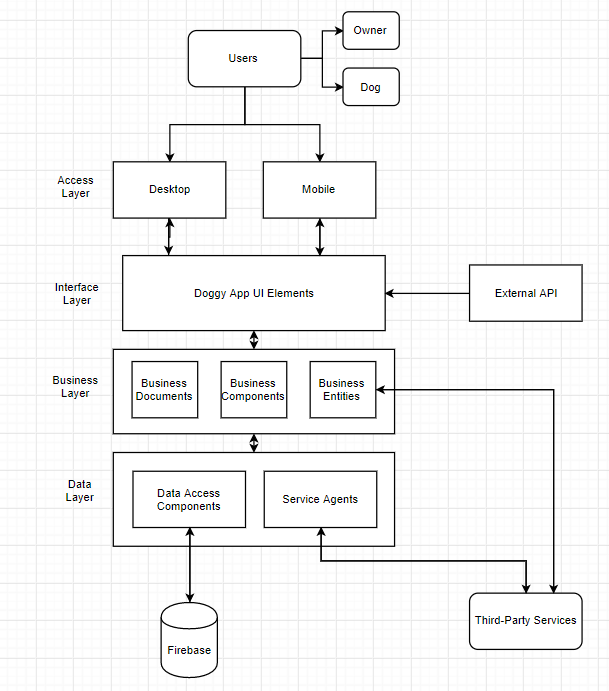
Below are some of the graphs based on which we chose our pipeline structure for our machine learning model:





1. Deployment plan

We plan on deploying our machine learning model as an external API where the image to be validated would be sent to the server and the result (1 or 0) will be received back by our application. This 0 or 1 is easy to interpret and also easy to print out for the users on the web application. The binary output makes it more readable and easy to check the accuracy.



1. CNN memory and computation requirements

CNN works on training the model from the training data which are the images of dogs. The images are of large sizes and since we will be training and testing our model with a large dataset, we will be needing a lot of memory to store the dataset. The processing power required for CNN is also large because we will be training the model on a large dataset and we will be retraining the model on the same images if needed since it's a deep neural learning network. Normal computers cannot handle this load and therefore, we will be needing an online server to run our model.   
  
We have developed our model on Google Cloud Platform/ Google Colab which allows the development of machine learning models. These online platforms can handle the load of training and testing the CNN model, as they do not consume the resources of our personal computers. As mentioned before, we will be training our ML model using a large dataset. This large dataset consumes a lot of space on disk since the size of the SSD/HDD of our computers is small. Therefore, we plan to upload the dataset to bitbucket and then clone that repository in the Google collab and use the dataset to train and test the ML model. Since the model will be trained and tested on Google Collab, there will be no specific computation requirements on our individual PCs. All the online services will make sure that the platforms compute all the high-level computations and handle all time-consuming calculations.   
  
All the images are 50 x 50 images, which means a lot of pixels per image. The computation time also includes the image classification, then gray-scaling all the images, then resizing all the images. There are about 25000 images just in the training data set, hence the computation time required to compute all the factors for every image is a lot.   
Input is a 50 x 50 image along with a label, and output is just the label (or to simplify it even further just a number, 1 or 0).   
  
The output is very simple as compared to the complex input images, hence the pipeline narrows as it reaches the end output. There are several save points in our compilation, and if-else codes which are used when we have already completed that step once, this can only be done while we are in the forward phase.   
  
An example of this is recreating the training data set (with grayscale images, uniform image sizes, and corrected/updated labels), this process only needs to be done once, and then later the result can be saved in a file with .np extension. This saving of the models saves us a lot of computation time, as this does not have to be calculated at every run, it can just be loaded when needed. No such model saves, can be done when we are trying to backtrack our progress and run the model as a whole.

1. Tradeoff analysis of deployment models

We have many options to deploy our machine learning model:

i. Convert the Python ML model into a javascript version

This model works well if we use sklearn model because there is a likely npm version of it in javascript, but converting the whole module into a Javascript version, could make us lose some great Python libraries, and it is not necessary to find the same level of online documentation for Javascript available as there is for Python.

ii. Utilize npm module PyNode

This is not well documented for us to use, and none of us have any experience with it as well, thus it does not sound like a great alternative to the proceeding.

iii. Deploy the Python ML model as an API:

We believe this would be a great way to deploy our machine learning model because we are already aware of API calls and how to format the received data. It would be easy for us to deploy the model this way. Since our model is in python,

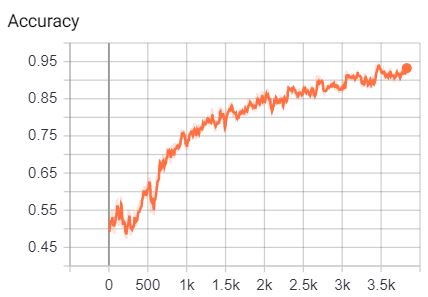
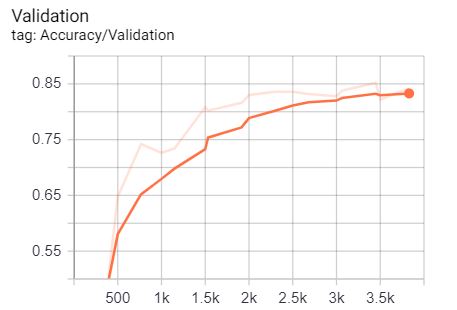
we can use frameworks such as Flask to optimize on both industry-standard

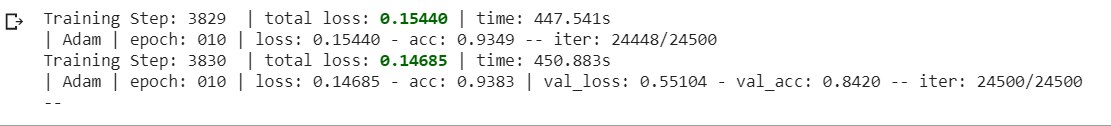
framework and simplicity.

1. How do we improve the accuracy of the model?

We train the model using convolutional layers. We add more layers of convolution with different numbers of nodes, to get better accuracy. We perform cross-validation on the training dataset to improve accuracy without touching the test dataset.   
Our original accuracy was about 64% with a huge data loss, adding more layers of convolution increased the accuracy to about 93% with an increased number of epochs.

1. Model is an underfit or overfit?

The model that we created is not an underfit nor an overfit because we have trained our model to give 93% accuracy which is good, to begin with, and the model does not have a lot of data loss. Any model with a loss lower than 1% is considered an overfitted model, but our model has about 14% loss, thus neither an underfit nor an overfit model. Even the graphs from the TensorFlow board give a more accurate picture of our model being neither an overfit, not an underfit. The below graphs (same as the one in Section IX part E) are plotted to show accuracy and loss functions for both the training dataset and the validation dataset.   


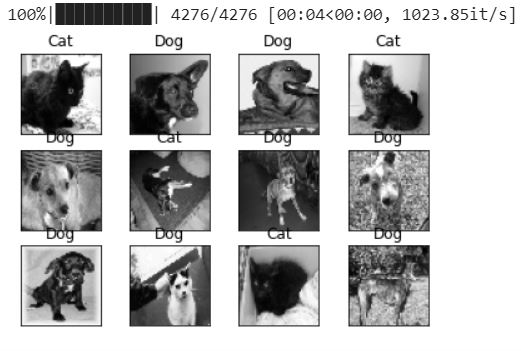
  
  
The following image shows the number of epochs used, the optimizer name, the achieved accuracy, and the total loss of our model.   
 

1. Performance matrix of ML capability

Currently, our machine learning model provides 93% accuracy which can be further be improved by increasing the number of epochs and adding more convolutional layers before training the model. We currently have an average loss of 14%, which can probably be improved a little bit more.

1. Does ML meet the business need in terms of metrics?

The machine learning model definitely meets our business needs. However, since our machine learning model predicts the image uploaded by the user, there might be a chance where our machine learning model could predict it wrong since our model currently has 86% accuracy. We, therefore, need to provide an option for the user to appeal saying that the picture uploaded is a picture of his/her dog. The uncertainty of the final outcome will always be something we have to be careful about because 93% accuracy is not a wholesome result. But the model after its API implementation, must be good enough to predict and filter out dog images and help us keep scammers away. The testing of our model looked like the figure below:



Our machine learning model was accurate enough to predict whether the image in the test dataset is a dog or cat.