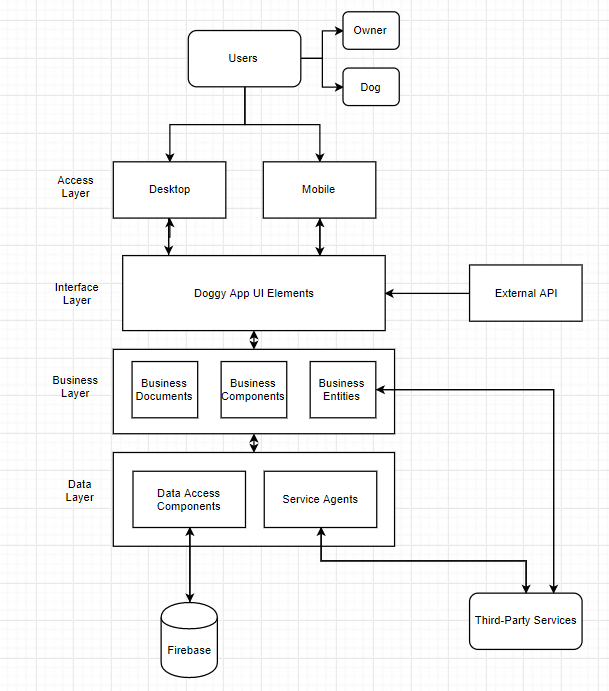
Architecture and Design Document

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
10. Cloud Vision API
11. 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 in 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

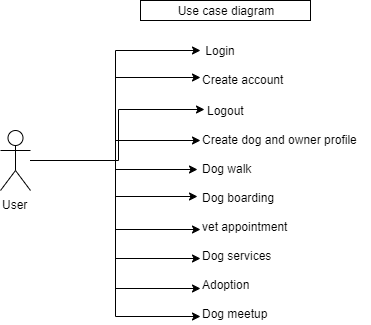
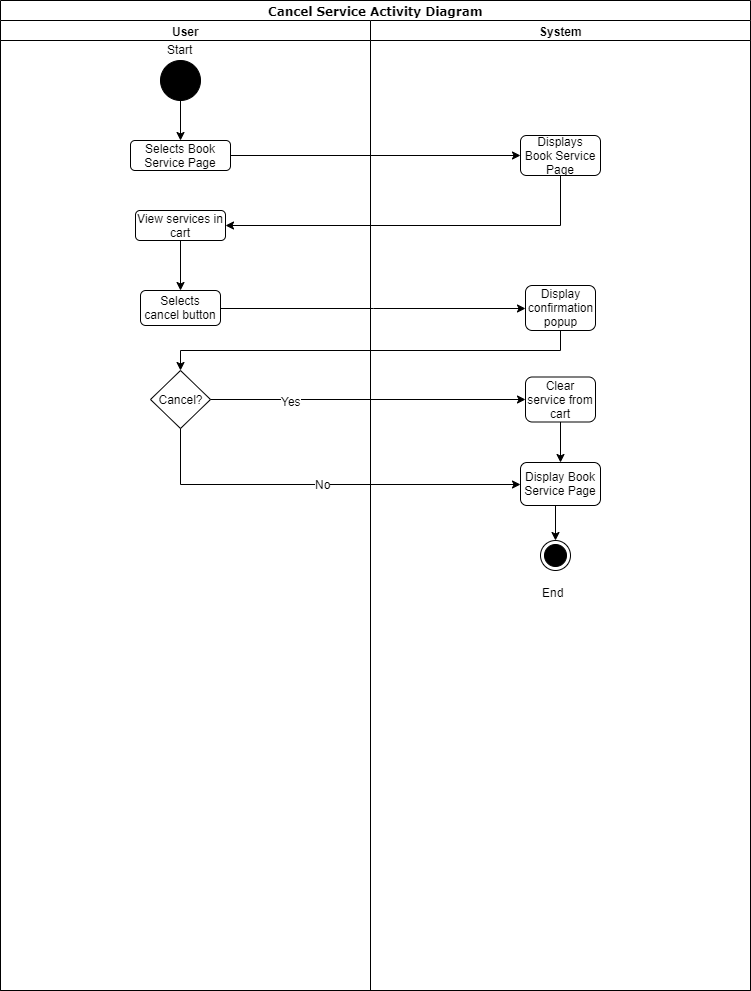
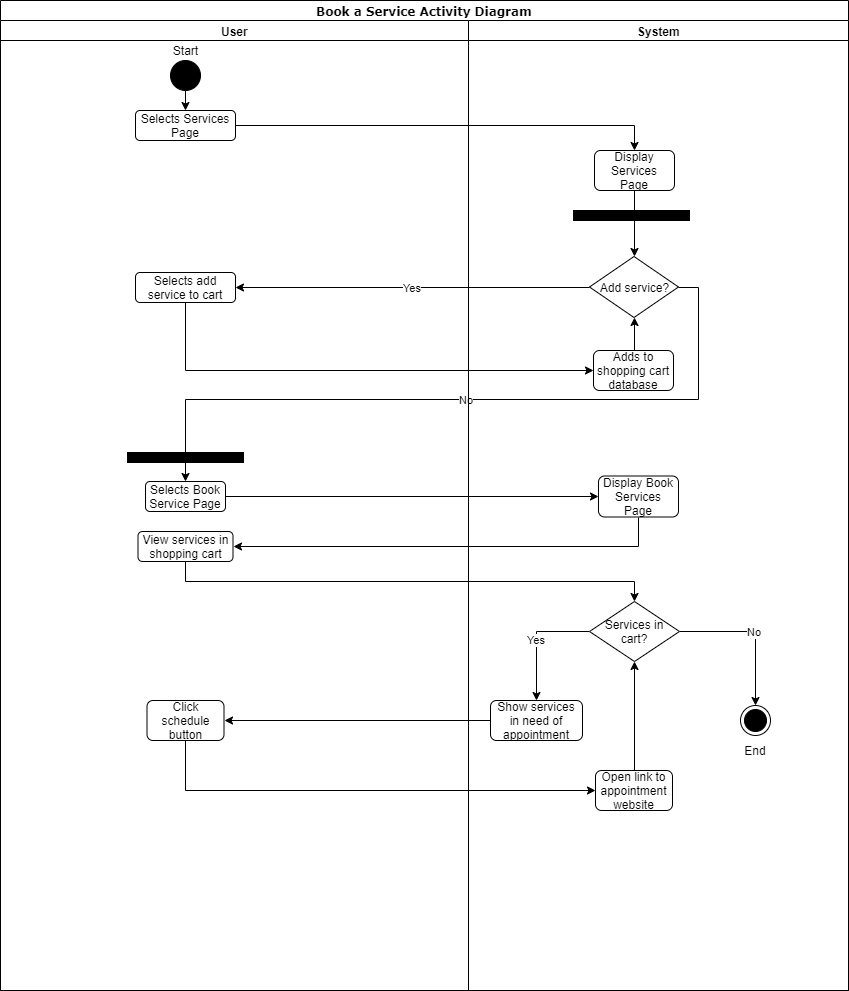
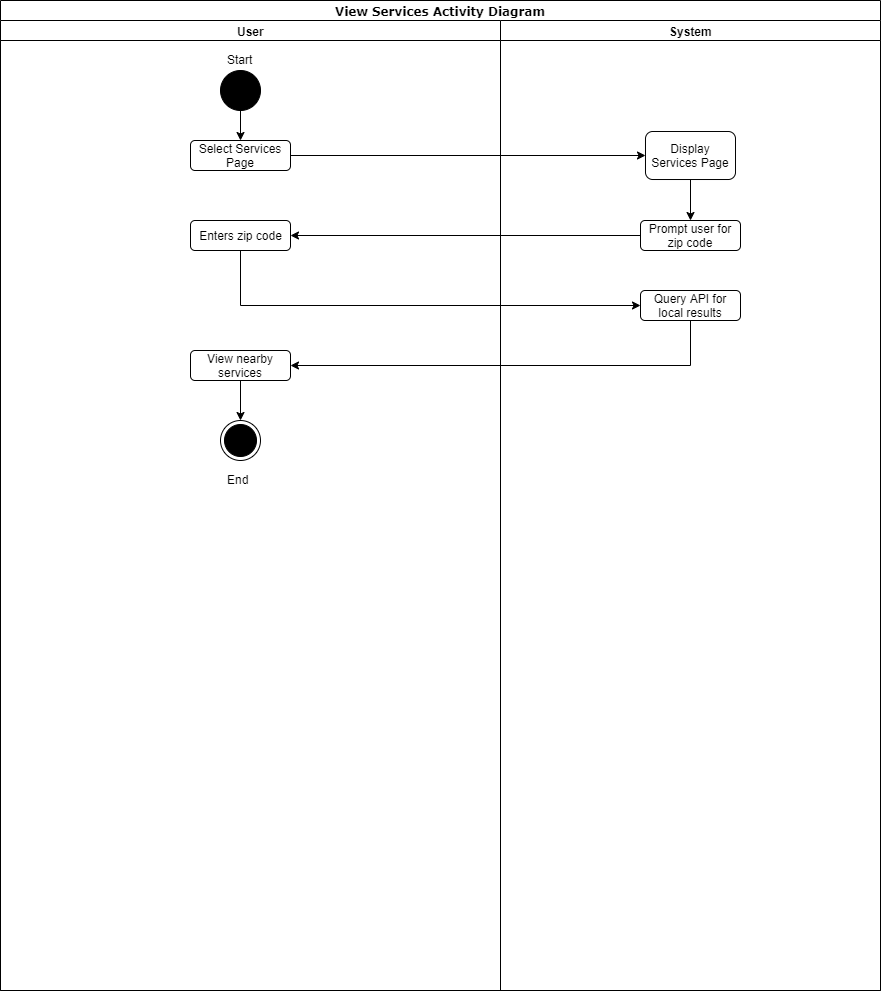
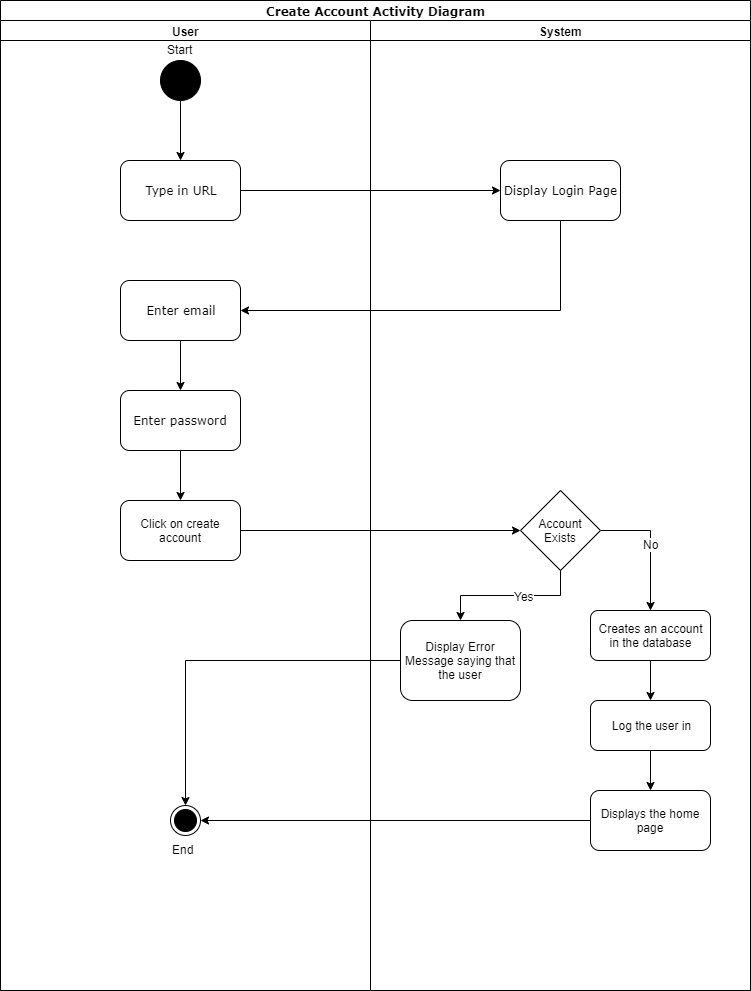
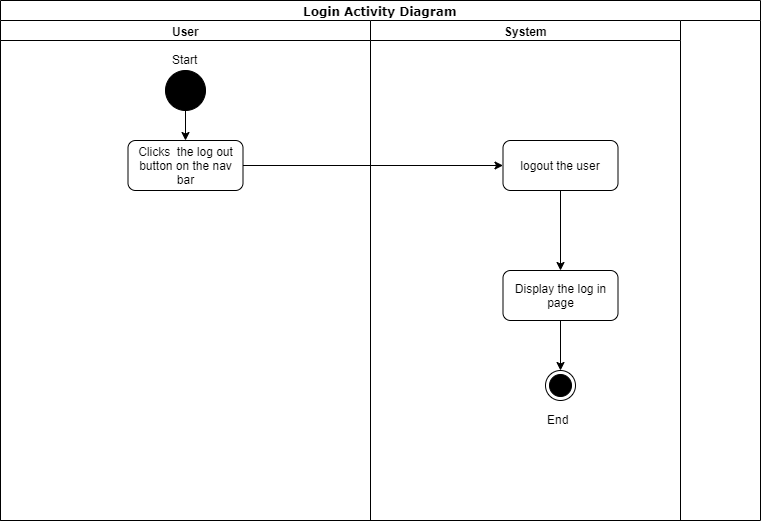
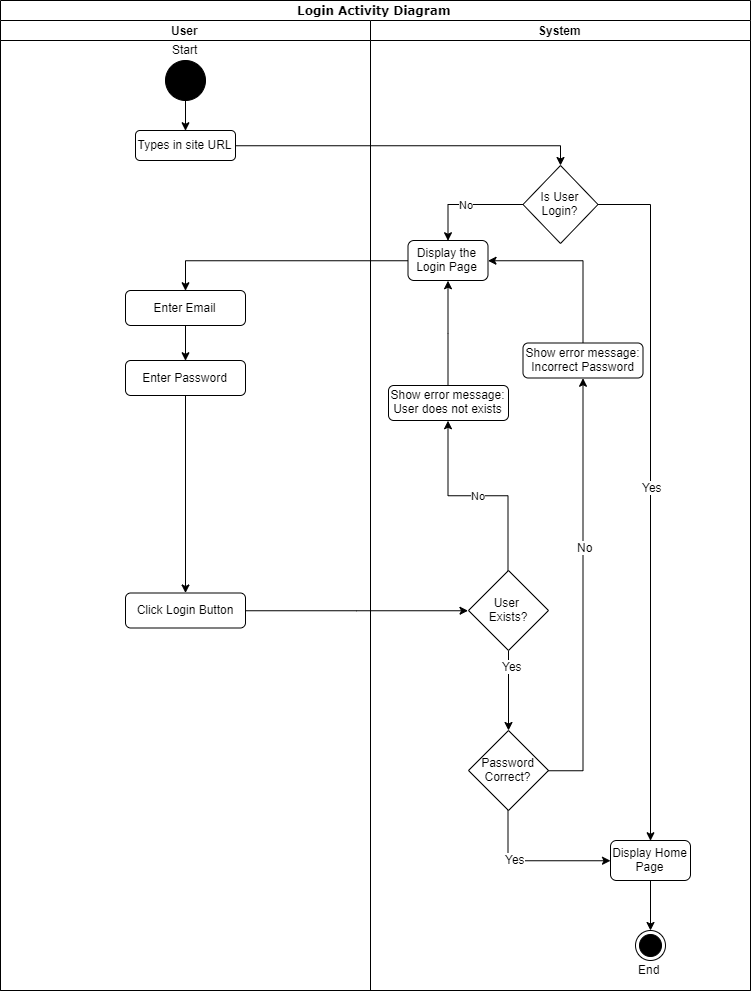
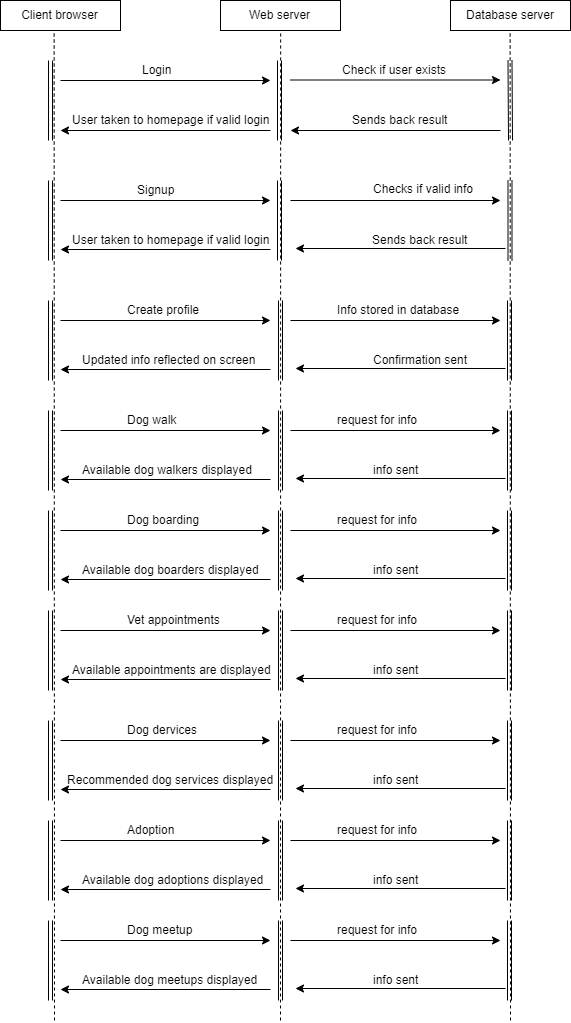
Large business organizations with a focus on user interface, use this architectural style. Because of the layered approach, different teams can be allocated to work on different layers.

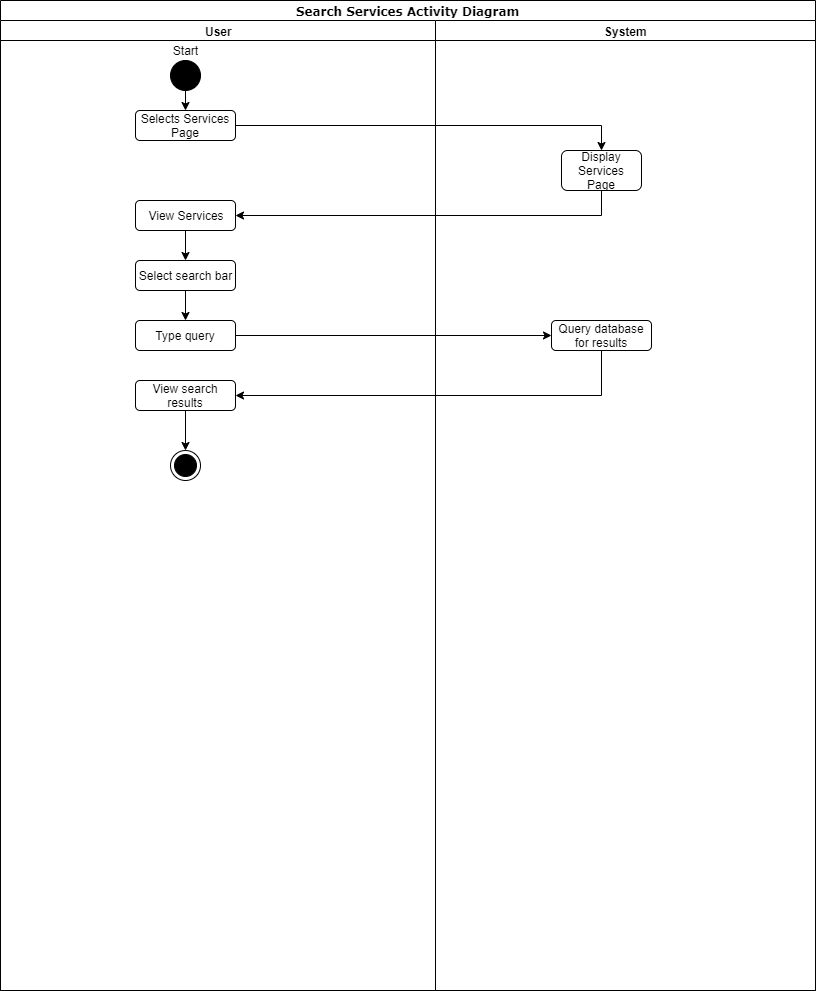
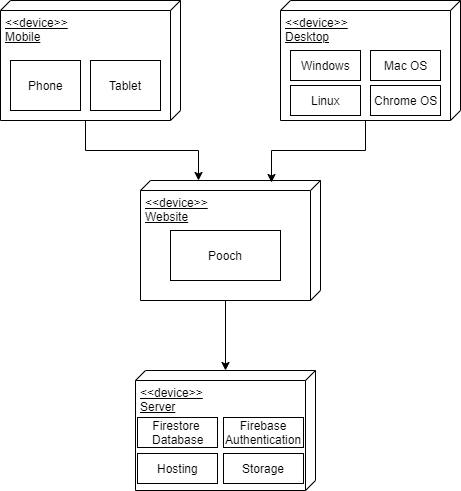
One advantage of the layered structure is the re-usage of lower-level layers. Certain lower layers can be used by different higher layers. Layers make standardization easier and we would be easily able to distinguish between the different layers and their functions. Changes can be made within a specific 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 shorted 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 |

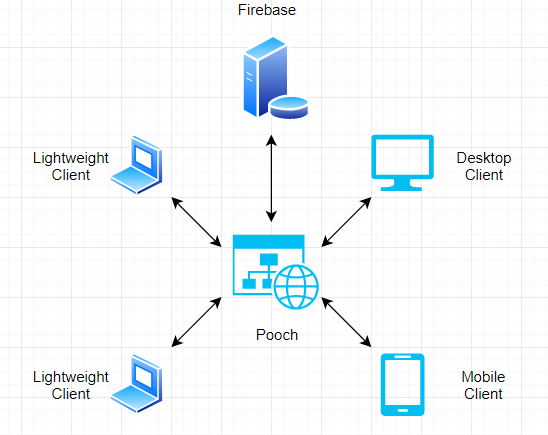
|  |  |
| --- | --- |
| **Method:** | **Implementation:** |
| User Story #1 | Sign in with social media |
| User Story #2 | Sign in |
| User Story #3 | Sign up/Add profile |
| User Story #4 | Visit home page |
| User Story #5 | Logout |
| User Story #6 | Navigate through pages |
| User Story #7 | View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

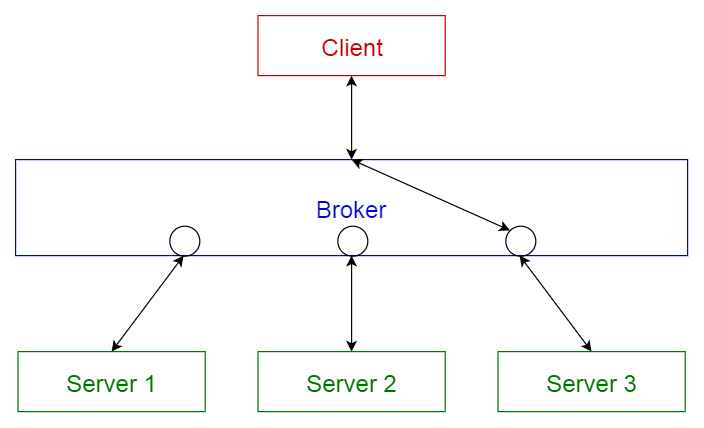
**Alternative architecture model: Layered architecture**

The alternative architecture design we are considering is 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. Server publishes their capabilities to a broker. Client requests a service from a broker, 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 are: Apache ActiveMQ and RabbitMQ, unfamiliarity to these softwares 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 needs 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 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 backend of the application | Hosting and authentication external to application. Less work for developers. Better security. | No control over hosting or ownership. Must trust Google to protect user data. |
| Create 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.   
  1. Cloud Decisions:

-All user data is stored in the cloud

-All service information and vendor contact information is 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 simple as possible

-Security should not make worse the user experience

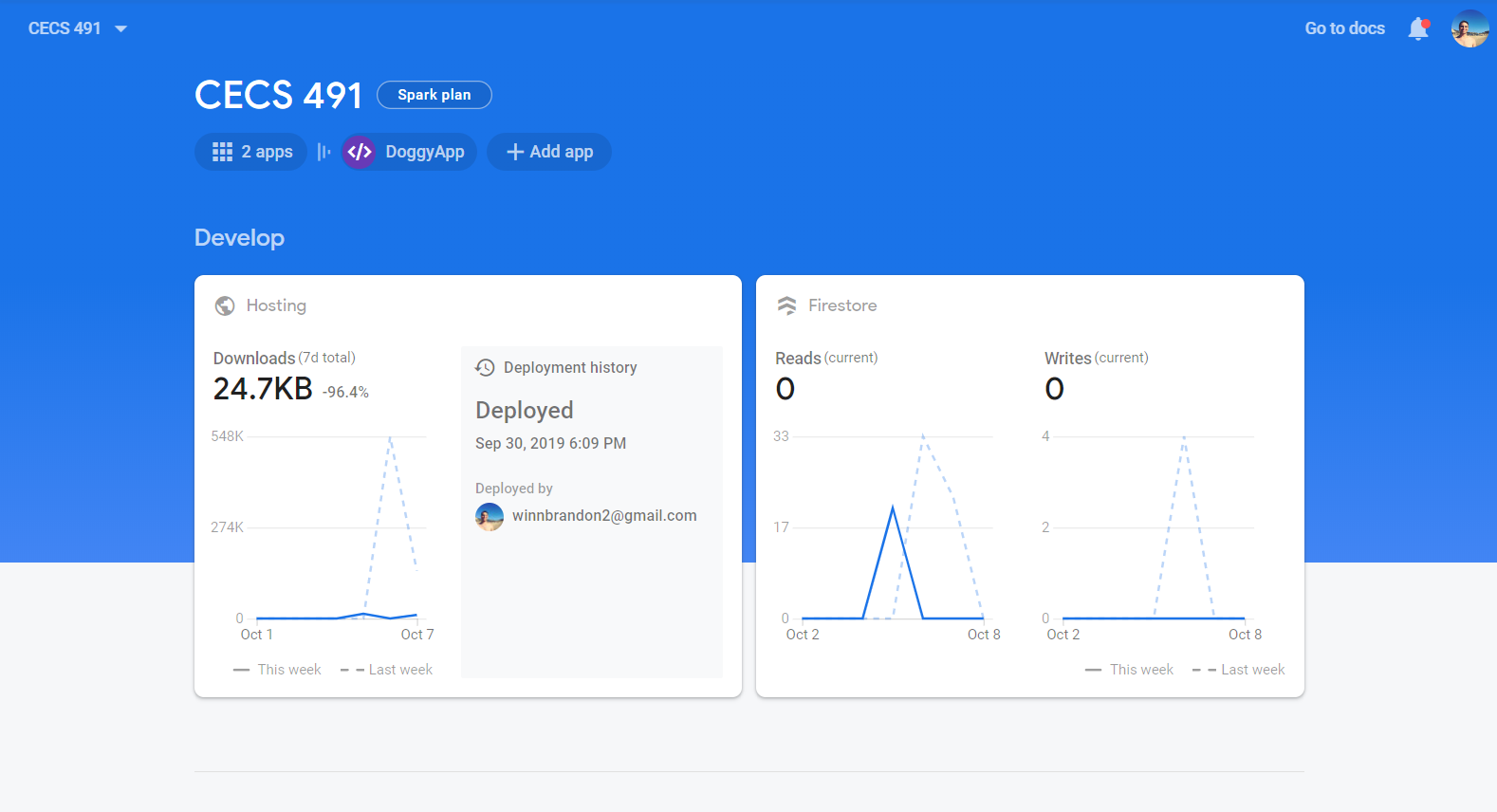
1. External Server Application  
   **What is Firebase?**



-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 collection, 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’ interface design?**

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

* 1. Capabilities
* Our server would be capable to provide facilities for both web applications and mobile website viewing. This gives us multiple server environments to run our application on.
* 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. Cloud Vision API

-Pooch will use Google’s Cloud Vision API to derive information from the images our users upload to the site. This will allow the integration of several security features which would otherwise not be possible for a small project.

1. Cloud Vision will ensure the profile pictures of owners are people and the profile pictures of dogs are actually dogs. This will reduce instances of troll accounts.
2. The API will help identify inappropriate content.
3. Duplicate photos could be detected to catch fake users stealing the profile pictures of other users.

This Cloud Vision API mainly works with the neural networks Machine learning model.

Neural networks are a set of algorithms, modeled loosely after the human brain, that are 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.

gi