**High level Architecture**

**09:PETSWALA**

**<team member names & ids>**

| **Student ID** | **Name** |
| --- | --- |
| **22100004** | **Adil Aslam Chaudhry** |
| **22100111** | **Ayan Tabassum** |
| **22100201** | **Mohid Yousaf** |
| **22100298** | **Roshan A.Aziz** |
| **22100248** | **Sabahat** |

**Table of Contents**

[1.](#_gjdgxs) Introduction 3

[2.](#_30j0zll) System Architecture 4

[2.1 Architecture Diagram 4](#_1fob9te)

[2.2 Architecture Description 5](#_3znysh7)

[2.3 Justification of the Architecture 6](#_2et92p0)

[3.](#_tyjcwt) Risk Management 7

[3.1 Potential Risks and Mitigation Strategies 7](#_3dy6vkm)

[4.](#_1t3h5sf) Tools and Technologies 8

[5.](#_4d34og8) Hardware Requirements 9

[6.](#_2s8eyo1) Who Did What? 10

[7.](#_17dp8vu) Review checklist 10

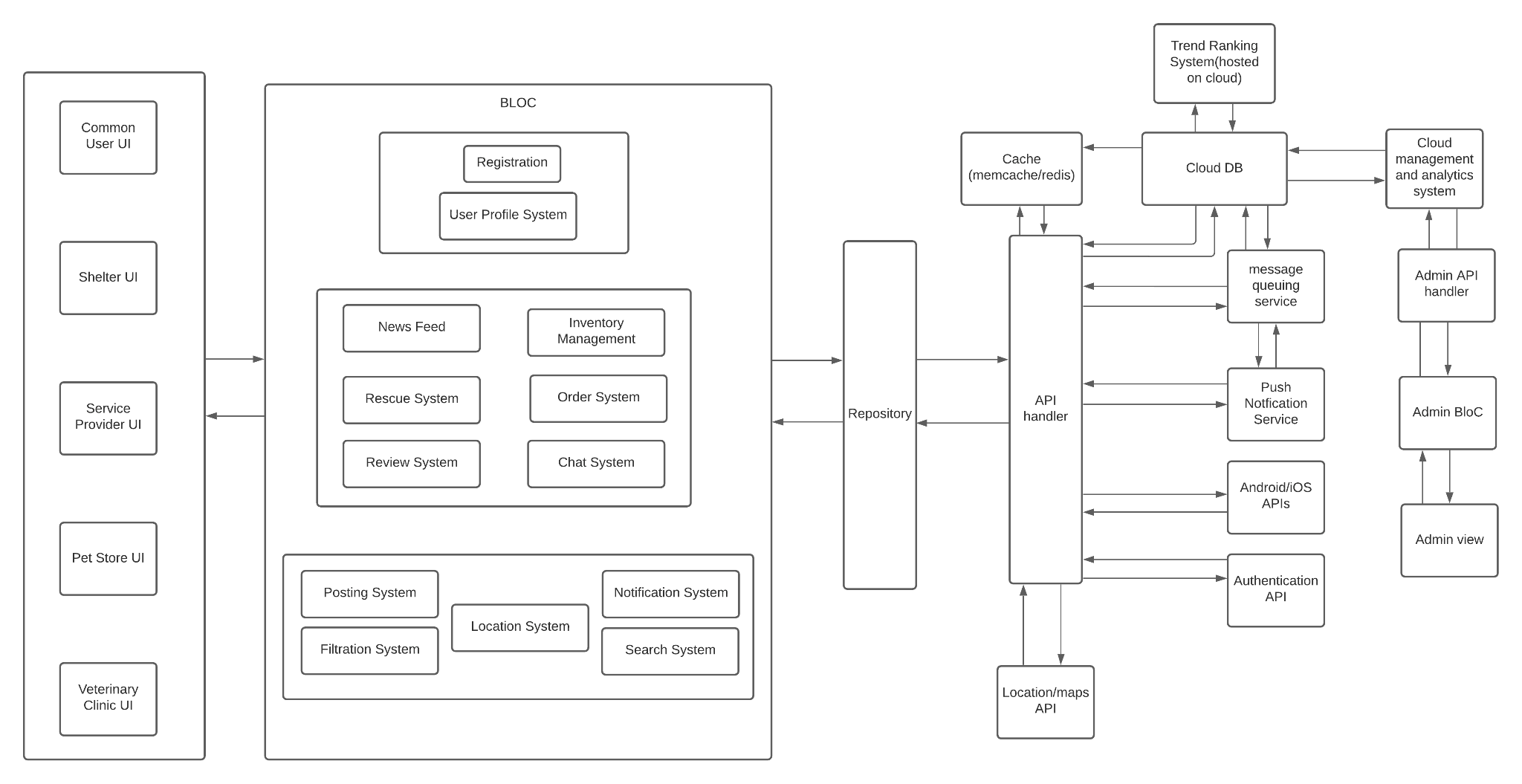
# Introduction

The project comprises an application catered towards pet owners, pet shops, sellers of pet accessories, veterinary doctors, pet rescue volunteers, and pet shelters/rescue teams. The application aims to link all these different entities through a variety of useful and helpful functionalities; pet owners would be able to search for different accessories, as well as finding good veterinary doctors for their pets, making their lives much more comfortable and better. They would also be able to interact with other pet owners and share photos and stories of their own pets as well. They could share helpful information with other pet owners. Any user of the app can also help with the animal rescue process by reporting pets to be rescued to the relevant rescue teams in the area.

# System Architecture

## Architecture Diagram

<Draw a diagram of the system architecture. Remember that your system’s architecture may be defined using multiple architectural patterns.>



## Architecture Description

There are a number of subsystems used in our architecture. These are divided into user, user actions and, utility. The subsystems include:

**2.2.1 User**

1. Registration:

This subsystem handles the entire process of registration, including generating and processing the forms.

1. User Profile System:

This subsystem is responsible for all the user preferences and the user profile data. If a user makes changes to their profile, this would be the subsystem that would be responsible for that.

**2.2.2 User Actions**

1. News Feed:

This subsystem would handle the way the news feed is displayed and whatever actions are performed on the news feed. This would use the Utility subsystems,as well as the User Profile System to generate its news feed.

1. Rescue System:

This subsystem would be responsible for creating the rescue requests that the pet shelters can then use to rescue pets.

1. Inventory Management:

This subsystem would be responsible for all the shops, stores, shelters and services on the application. This subsystem would manage their inventory, generate the store page.

1. Order System:

This subsystem is closely linked to the inventory management subsystem. This subsystem manages all the orders and the purchases. It allows the users to buy from their stores and handles all the necessary processing for order confirmation.

1. Review System:

This subsystem deals with the reviews that the user leaves on the application. It allows the users to leave the reviews and the stores to check them.

**2.2.3 Utility**

1. Posting System:

This subsystem makes posts and submits them to be posted on the news feed or for the rescue system.

1. Notification System:

This subsystem is responsible for sending notifications and responding to received notifications.

1. Filtration System:

This subsystem handles all the filtration tasks, for all the various interfaces. It also includes processing reports for hateful content or spam.

1. Location System:

This subsystem handles getting the user location or fetching coordinates from a location to send to Google Maps.

1. Search System:

This subsystem handles all the search related tasks of the application, whether it be searching for stores, services, posts etc.

These subsystems are interconnected through data streams, they take data from each other. The user actions all take into account and depend on the preferences of the user. While the user actions all use the utility functions, which are the reusable components of the software, for various tasks and needs.

## Justification of the Architecture

Our primary concern was not only to make different layers of our application separate but to make complex subsystems out of simple blocks, hence we came out of BloC Architecture keeping in mind the technologies we are using and the context of our system. We have a user profiling system that has multiple UI interfaces, for instance common user UI, Shelter UI, Service Provider UI etc.

The business logic layer of the architecture contains the business logic of the subsystems present in a semi restricted layered structure, keeping the most used subsystems in the lowest layer, hence maintaining a useful layered structure. As our application is based on flutter, our application and business logic resides client side in order to ensure low-latency and a responsive UI design. The business logic is kept entirely separate from the UI and data layers to keep the system modular and scalable, both in terms of functionality and load. In order to communicate with the UI and repository layers, we use an event-based architecture with streams. Events trigger functions which perform business logic and state is updated via streams. This architecture is common and optimised for flutter and works well in separating state from UI.

The repository layer is used to handle multiple sources of data from APIs. For example data could be fetched from a cache or if the cache is expired then from the database. So in order to facilitate multiple API calls and data sources the repository layer handles streams from the API handler and forwards optimal data to the Business Logic layer.

The API handler layer will handle all API calls and connections with the DB etc. it will handle connection timeouts as well as responses for API calls e.t.c. This is kept separate from the logic since these data sources and APIs can change and in order to facilitate modular design, only the API handler layer has to handle it, the logic layers will not have to cater to it.

We used cloud DB and storage options in order to ensure consistent latency and up-time and to ensure that there is not a single point of failure. Moreover, having scalable cloud storage helps in easy load scaling and it also facilitates a flexible load, so that costs are dependent on usage only. The system can also quickly adapt to very frequent changes in load and we will not have to physically adapt to such change in resource requirement.

We also added caching servers to serve pre-processed content with < 1ms latency. This will ensure quick loading times for the news feed. The caching servers will reside on the cloud and will fetch data from the db periodically.

In order to serve a live news feed from the many posts by many users, we thought that we should pre-process and rank tags and posts in order to prevent high processing time when a user opens the news feed. So we added a cloud-based Trend-ranking system which will periodically run algorithms and calculate top trends in order to bring relevant content on news feeds. This will run separate from client requests and will ensure that there is not a high client-side load when a user requests the news feed. The calculated trends will be served to both the db and the cache.

We also have a message queuing and push notification system to ensure deliverability of messages including rescue notifications and comments/post notifications e.t.c.

Keeping the above in mind, our architecture has the following pros:

1. Easy to make changes in UI because of separation of state and UI
2. Easy to change data sources and APIs because dependence on only one layer
3. Scalable and flexible due to scalable cloud services and client side logic
4. Client-side application logic to ensure low latency and increased responsiveness in UI
5. Data heavy tasks and logic delegated to a separate cloud backend to minimise heavy client-side computation.
6. Caching to ensure low latency and quick data delivery
7. Pre-computing heavy tasks ensures low run-time latency

However, there are a few cons:

1. Client side is thick as application logic resides there making the app larger.
2. Regularly computing trends is very resource intensive.
3. Client requests need to be re-validated server-side because we can never trust client-side data.

# Risk Management

## Potential Risks and Mitigation Strategies

| **Sr.** | **Risk Description** | **Mitigation Strategy** |
| --- | --- | --- |
|  | User Engagement: After the release of the product, there might be a difference between stakeholders expectations and users actual engagement. | Incorporate user feedback in updates. Project will be designed in a way that is adaptable to changes. Moreover, testing, surveys and frequent updates will help avoid it. |
|  | Requirements Inflation: With the progress of the project, more requirements can be identified which were not stated in the beginning. | Involvement of customers at an early point in the development of the project, so that most of the requirements can be identified as soon as possible and relevant changes and estimations can be made. Moreover, frequent reporting in between the team would also help mitigate this risk. |
|  | Holes in software security: The application could have holes in it’s software security component and could be an unsecure application. | Use the appropriate APIs to have secure software. Additionally, plan your software architecture in a way that you have secure software. Logging the application use can also help in identifying these holes. |
| 4. | Unpredictable external risks: Sudden changes in the market, there can be a competitor with more resources and fast growth, some changes in consumer behaviour and priorities, or any new laws. | Thorough research about market, competitors and laws to mitigate the chances of such changes affecting the product. Enhance user experience and good marketing of the product |
| 5. | Lack of professional experience: Since we are students aiming to make a professional project, we are still learning. This inexperience could result in various problems. | Stay constantly updated on if we are on the right track by seeking help from seniors and the instructor and researching on Google. |
| 6. | Change in platform software: Due to the rapid changes in technology, changes can occur in the platform that we are developing for. Our application might risk falling off the cutting edge of technology and design. | Keep up to date on the release dates of these platform updates and either cater them during development or have an update path figured out beforehand. |
| 7. | Poor Management: Due to poor communication within the team, the developers might not share the important key information which is necessary for a smooth release of the product. | Scrum, weekly reports and frequent meetings |
| 8. | Resources: Unable to obtain required resources | Because the project does not require many resources, it is critical to obtain them as early in the development phase as feasible, and alternate resources, such as local test servers, should be prepared for. |
| 9. | Inaccurate Estimates: The estimates of number of users can be inaccurate. | We will do thorough research and conduct surveys to estimate the number of different types of users. |
| 10. | Code Quality: The lack of time or rushing, might result in poor code quality. | Leaving enough time to check for bugs, and do thorough testing |

# 

# Tools and Technologies

1. Flutter 2.0 for development on both iOS and Android
2. AWS Aurora for cloud DB
3. AWS elasticache for Caching servers
4. AWS EC2 for hosting and computation
5. AWS location API

# Hardware Requirements

The hardware required for this application would be as follows:

* A smartphone with Android 7.0 or a higher version to test the final build.
* A device with GPS to test the location services element of the application.
* A device with a camera to test the picture and video posting capabilities of the application.
* A device with internet connection
* Cloud servers with expandable storage
* Auto Scaling EC2 computing
* location based Caching servers to serve pre-processed content with minimum latency

# Who Did What?

| **Name of the Team Member** | **Tasks done** |
| --- | --- |
| Adil Aslam Chaudry | System Architecture Diagram, Justification of Architecture, tools and technologies,Tools and Technologies |
| Mohid Yousaf | System Architecture Diagram, Justification of Architecture, Tools and Technologies, Hardware Requirements |
| Sabahat | Risk Management |
| Ayan Tabassum Saeed | Potential risks and mitigation strategies |
| Roshan A. Aziz | I worked on the system architecture, as well as the architecture description and I wrote a few of the hardware requirements and risks and mitigation. |

# Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

| **Section** **Title** | **Reviewer Name(s)** |
| --- | --- |
| Architecture Diagram | Adil |
| Architecture Description | Sabahat, Mohid |
| Justification of the Architecture | Sabahat,Mohid |
| Risks and Mitigation Strategies | Sabahat, Mohid |
| Tools and Technologies | Mohid |
| Hardware Requirements | Adil |