CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

EDEN – The Pet Application

by

MARSH ROYDEN J (1941122)

SEAN LEO NORONHA (1941136)

YOBAH BERTRAND YONKOU (1941146)

SAAKSHI AGARWAL (1941160)

Under the Guidance of

Dr Gobi R

Project report submitted in partial fulfillment of the requirements of V Semester BCA, CHRIST (Deemed to be University).

December - 2021

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

CERTIFICATE

*This is to certify that the report titled **EDEN** is a bona fide record of work done by **Marsh Royden J (1941122), Sean Leo Noronha (1941136), Yobah Bertrand Yonkou (1941146), and Saakshi Agarwal (1941160)** of CHRIST (Deemed to be University), Bangalore, in partial fulfilment of the requirements of V Semester BCA during the year 2021.*

Head of the Department **Teacher in Charge**

Valued-by:	Name:	Marsh Royden J
		Sean Leo Noronha
		Yobah Bertrand Yonkou
		Saakshi Agarwal
	Register Number:	1941122
		1941136
		1941146
		1941160
	Examination Centre:	CHRIST (Deemed to be University)
	Date Of Exam:	25.11.2021

ACKNOWLEDGEMENTS

The project, EDEN, was successfully completed with the help of a lot of people. Firstly, we express our sincere gratitude to our Vice-Chancellor Dr.Fr. Abraham V M, Pro-Vice Chancellor Dr.Fr. Joseph CC, Head of the Department Prof. Joy Paulose, Coordinator Prof. Deepthi Das and the faculty for providing us with this priceless opportunity to put in and improve our technical skills and knowledge through this project.

We express our gratitude to our project guide Dr. Gobi R who took keen interest in helping and guiding us throughout our project work by providing all the necessary information for coming up with such a good result. We would also like to thank the faculties of the Specialization Project for the ideas, continuous feedback and supervision on our project work.

We are thankful and fortunate enough to get constant encouragement, support and guidance from the faculty of BCA who helped us in successfully completing our project work. Also, we would like to extend our sincere esteems to all staff in the laboratory for their priceless support.

ABSTRACT

While observing websites offering sales care/grooming, adoption and reporting injured or stray pets, it can be noticed that most websites only provide proper subsets of the features mentioned above. Very few sites offer a single solution containing all these features thus pet owners will have to maintain different accounts to use the services above. The proposed solution, 'EDEN' provides a one stop solution where people and pet owners can adopt pets, book grooming facilities for pets, and report stray or injured animals thus making it easy for pet owners to manage services related to pets and thus contribute to the care for animals be it strays or owned.

This project will deal with helping domestic animals. It will allow a user to report any animal abuse to the nearest NGO, and contact a veterinarian for a check-up or an emergency. A user can also adopt an animal using this and can see all the events organized by different NGOs. It will also have different modules for NGOs, veterinarians, accessories and grooming sections.

The NGO module will maintain the records of all the animals present for adoption and organize different events. The veterinarians and grooming module will be used for scheduling appointments from NGOs as well as from users. To make sure that a user attends the selected slot they will make an initial payment. The accessories module will have all the basic requirements that are needed to take care of the animals starting from foods to shampoos.

This project aims to help the animals, stray or owned and provide adequate treatment if needed in case of an emergency. This will also promote the concept of adopting a pet from an NGO which in return would help the NGOs to grow and have funding. Additionally, NGOs will have opportunities to collaborate and organize events. The veterinarians and grooming modules will allow the user to contact the vet in case of emergency or schedule a regular appointment with them.

The project will thus focus at developing a web and mobile application that will work towards users assisting domestic animals especially strays through providing them a means to get in contact with a local NGO and avail services for animals wherein they will be taken care of, and recuperate.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

The current paper is a software requirement specification for a mobile and web client application to allow local NGO groups, and concerned citizens to coordinate and work towards the care of animals both owned and strays, along with attention to their nourishment and survival.

1.1 BACKGROUND

The COVID-19 pandemic caused a landslide impact on the lives of all living beings, businesses, and animals. During the initial lockdown period, the disappearance of the people from the city streets was observed, and the deserted streets were now a playground for all variety of animals, especially stray dogs and cats. However, this eventually turned out to be detrimental to such animals as sources of food began to dwindle and strays began to die as humans were a means for such animals to get food from. Hence, to provide care for such abandoned, stray animals, the idea of 'EDEN' was planned, which would coordinate the operations and efforts of the local NGOs, veterinarians and concerned human beings to take care of the street animals.

This project will aim at developing applications which will help streamline the associated operations with the NGO, veterinarians, and the public.

1.2. OBJECTIVES

- Help animals, stray or owned and provide adequate treatment if needed in case of an emergency.
- Promote adoption of pets from an NGO which in return would help the NGOs to grow and have funding.
- Allow users to contact a veterinarian in case of an emergency or schedule a regular appointment with them.
- Develop an easy to understand and attractive interface for all users with operations which are accessible with a button click, with little to no navigation required.
- Provide notifications and updates to the public who are registered with the applications especially for the scheduling of appointments with the groomers, veterinarians, etc.

1.3. PURPOSE, SCOPE AND APPLICABILITY

- PURPOSE:

The proposed solution, 'EDEN' provides a one stop solution where people and pet owners can adopt pets, book grooming facilities for pets, and report stray or injured animals thus making it easy for pet owners to manage services related to pets and thus contribute to the care for animals be it strays or owned. This application will be a single compilation of all the

subset applications available for all pet related applications related to grooming, veterinary checkups and animal adoptions.

- **SCOPE:**

The ‘EDEN’ project will include a web client application and a mobile client application. Both clients will contain functions and operations unavailable to the other. Certain operations like scheduling appointments, reporting emergency calls for stray animals, and getting updates will be centered on the mobile client. On-site operations like sending reminders to users and responding to report incidents regarding injured stray animals, updating the NGO events, etc. will be centered around the web client.

As such with the limitation of time, certain functionalities might not be completed in time; however, applications with important operations will be developed and demonstrated for the requirement of this project.

- **APPLICABILITY:**

The project will assist the target users at the NGO, veterinarians, groomers, and concerned citizens to access the appropriate services like reporting incidents to the NGO, schedule an appointment with a recognized veterinarian, adopt strays currently at the NGO, etc.

Stray animals can be found almost anywhere, especially in India. Many households around the globe avoid the unnecessary cater to the needs of the stray animals, and often ignore the health conditions of such animals. These animals may also be ill-treated and always lack shelter. Hence, to help such animals and coordinate the efforts of NGOs, government shelters and animal lovers around the world, the project, ‘EDEN’ will be developed with a focus to better improve the life of such stray animals and making the general public appreciate the existence of animals around them.

2. SYSTEM ANALYSIS AND REQUIREMENTS

2.1. EXISTING SYSTEM

There are few systems present for animals like TailsLife which has listings for veterinarians, pet spas, boarding, pet supplies, and pet-friendly restaurants and resorts. Other existing systems will be Pet First Aid and Pet Phone which manages pet details and pinpoint what's wrong with them. MapMyDogWalk helps you to log your dog walks, track the calories and map the favorite walking routes using GPS.

2.2. LIMITATIONS OF THE EXISTING SYSTEM

While observing systems offering care/grooming, adoption, medical support and reporting injured or stray pets, it can be noticed that most websites only provide a particular subset of the features mentioned above. Very few sites offer a single solution containing all these features thus pet owners will have to maintain different accounts to use the services above.

2.3. PROPOSED SYSTEM

The ‘EDEN’ provides a one stop solution where people and pet owners can adopt pets, book grooming and medical facilities for pets, and report stray or injured animals thus making it easy for pet owners to manage services related to pets and thus contribute to the care for animals be it strays or owned. It will also help NGO to grow by giving users opportunities to adopt pets from them as well as raise funds.

2.4. BENEFITS OF THE PROPOSED SYSTEM

As a one-stop solution for pet care and management, EDEN presents many benefits to pet owners, animals and the environment. These benefits range from reducing time spent on creating separate accounts for different pet services and dealing with different parties to even reducing the cost spent on pets which may arise from switching from one site to another with different pricing models. The following are the significant benefits of using EDEN;

- Convenience is one of the benefits of using EDEN. That is, pet owners do not have to go to multiple locations (or websites) because they can go to only one location (EDEN) for all their needs.
- Pet owners will be able to manage adoption, grooming, reporting and medical services related to their pets from one portal and at the comfort of their homes until the appointment time.
- Doctors will be able to manage appointment requests and close appointments at their convenient time.
- NGOs will be able to receive and respond to calls concerning stray or injured animals. Also, they will have a portal where they can manage reports about the same.

- EDEN will provide a platform where groomers can show off their excellent grooming skills and gain popularity.
- Pet owners will experience a decrease in their spending on their pets, as they will not have to deal with different pricing models from different sites; instead, they will have one price that caters for everything.
- Pet owners will receive notifications about their appointment at a specified time (an hour or day) before their appointments.
- Pet owners need not worry about situations like illnesses that are beyond EDEN doctors. In such cases, EDEN will be responsible for finding a specialist for our clients.
- Pet owners will get the exact food description, and samples as doctors at EDEN will have access to EDEN's accessories shop.
- Pet owners need not worry about the authenticity of products kept in the accessory shop because all products will be verified by our doctors before adding them to the shop.
- Increase in pet health as doctors will check on the pets until their recovery.
- Pets will get immediate care when they are spotted and reported, thus reducing the number of helpless and injured pets on the street.
- Pet owners will enjoy free grooming and pet feeding for their pet as a part of the treatment process if required.

2.5. FEATURES OF THE PROPOSED SYSTEM

- **Appointment scheduling:** Pet owners will be able to schedule appointments with doctors and groomers. This will be done by sending an appointment request to the doctor or the groomer, who will accept or reject the appointment based on their availability. The appointment scheduling system will work hand in hand with the notification system to inform pet owners about the status of their appointment requests or process.

- **Animal reporting**: Using EDEN, users will be able to place reports based on injured, maltreated or helpless animal sightings to NGOs near them. The system will include several trusted NGOs who can respond to reports within the shortest possible time. To ensure that reports are authentic, users will be able to attach photos of the scene.

- **Pet adoption:** The system will contain an adoption section where users can adopt pets. Pets will be put up for adoption by trusted NGOs. Users will be allowed to put up their pets for adoption. A user willing to put up a pet for adoption must submit a request to any NGO. Protocols will be put in place when putting a pet up for adoption and during the adoption process to ensure the well-being of the pets and the satisfaction of the people adopting the pets.

- **Pet's accessories:** EDEN will contain a section dedicated to pet accessories. This will be an online shop containing verified pet food and other things required for the well-being of pets.

- **In-App payment:** Users will be able to use their favorite payment methods to make payments where necessary. The payment feature will be incorporated with the accessory shop, doctors and groomers appointments and the fine system.

- **Mobile and web clients:** The application will be developed for different platforms based on users' working environment to ensure that they feel comfortable using our application. Keeping in mind that users like NGOs, Groomers, and Doctors, while at their job site, mostly interact with desktops or laptops while offering their services, we will be providing a web application for them. More so, since pet owners are more likely to use their phones at all times, we will be providing a mobile application for them to interact with our system.

- **Searching (real-time) and Sorting:** Real-time searching will be tightly coupled with all search boxes provided in this application. When application users are searching for doctors, pet owners, groomers, NGOs or even records, the results will be displayed instantly (After entering a letter, all records containing that letter will be displayed). More so, users will be able to sort any list according to date, time, and status.

- **Notifications:** EDEN will contain a notification system that will be incorporated in modules like appointments, shopping and reporting.

- **Rating and User feedback:** Pet owners will be able to rate and give feedback about their recent encounter with an NGO, Doctor or Groomer. Using the ratings NGOs, doctors and groomers will be ranked.

2.6. SYSTEM REQUIREMENTS SPECIFICATION

2.6.1. USER CHARACTERISTICS

Pet owners - These are the primary end users of the app. The pet owners are the users who own one or more domestic animals. They will be able to purchase accessories for their pets, contact veterinarians and also contact the NGO. These users will be primarily using the mobile client to make use of these services. Each user will be assigned a user name, an internal user ID and a password.

Concerned citizens - These are the secondary users of the app. These users are the type of people who care about animals. They will be able to report animal abuse to the NGO and also inform them about any animal that might need immediate help.

Veterinarians - The veterinarians will be able to accept and manage the different appointments. These users will be primarily using the web client. The appointment information will be synced between the web and the mobile client.

NGOs - The employee at NGO will be able to see the request made by the user and accept it or decline it after verifying it. They will also be responsible for putting animals up for adoptions and organizing events for raising awareness and funds.

2.6.2. SOFTWARE AND HARDWARE REQUIREMENTS

Table 1: Software and Hardware Requirements

N/O	Name	Use
1	Mobile Client	- Any android phone which is running android M and above (API level > 23)
2	Web Client	- Windows - An Intel Pentium 4 processor or later that's SSE3 capable - Linux - 64-bit Ubuntu 14.04+, Debian 8+, openSUSE 13.3+, or Fedora Linux 24+

2.6.3. CONSTRAINTS

- Both the mobile client and the web client will be completely based on English. Any user whose primary language is not English won't be able to work with the app.
- The only supported currency for the online store will be INR. No other currency will be supported.
- There will not be any separate API's that will allow other clients to interact and get the services provided by this software.
- The mobile client is a native android app. The app will not be available to IOS users.
- The android app will work only on devices running API level 23 (Android Marshmallow) or higher. All android devices lower than API level 23 won't be able to download the app.
- There will not be any accessibility features built-in to the app. People who are physically impaired must resort to the OS level accessibility features.
- The web site will not be accessible from a mobile device. The user has to download the app in order to make use of the services.

2.6.4. FUNCTIONAL REQUIREMENTS

Table 2: Functional Requirements

Requirement ID	Requirement	Description
FR_1	Authentication/ Access mode	- All users must be authenticated and logged in before accessing their data. Users will have access only to data entitled to them. That is, each user will have their boundaries on what they can access. Any attempt to access unauthorized data will be dealt with severely. - Each user accessing the system will have a set of functions and a dedicated interface. That is, pet owners will have a different interface from NGOs.
FR_2	Searching, Sorting and Filtering	- Pet owners will be able to search and get results instantly from a list/database of Appointments, Doctors, NGOs, and Groomers. - NGOs will be able to search and get instant results from a list of reports and doctors. - Doctors will search and get instant results from a list of Appointments, pet owners, groomers and NGOs. - Groomers will be able to search from a group of Appointments, pet owners and doctors. - A sorting and the filtering feature will be available for each list (Alphabetically, day-wise.).
FR_3	Booking appointments	- Pet owners will be able to book appointments with doctors and groomers. Appointments will be either rejected or accepted by the doctor or the groomer based on their schedule.

FR_4	Record keeping	- Doctors will be able to see all appointments whether accepted or approved. This will be very helpful in a case where the receptionist is responsible for accepting and rejecting appointment requests from clients. The doctor can go through the data to verify that everything was done correctly. - Records on transactions made will be maintained for all users of the application. - Records of appointments between pet owners and groomers or doctors will be maintained on both sides.
FR_5	Reporting	- NGOs, Doctors, Groomers and Pet owners will receive weekly and monthly reports about their activity on the application.
FR_6	Shopping	- Application users will be able to purchase pet products from the pet accessories online shop.
FR_7	Payment	- Application users will be able to make In-App payments for services received. A variety of payment options will be included taking in mind the preferred payment method of the users.
FR_8	Pet adoption	- Pet owners will be able to adopt pets of their choice. All adoption procedures will be followed thoroughly to ensure the well being of the pet and the satisfaction of the pet owner.
FR_9	Notification	- The application will contain a notification system responsible for notifying pet owners about appointment status and offers.
FR_10	Dashboard	- NGOs, Veterinarians, Groomers and Accessories sellers will need a dashboard to get a glance of some values related to their services.

2.6.5. NON-FUNCTIONAL REQUIREMENTS

Table 3: Non-Functional Requirements

Requirement ID	Requirement	Description
NFR_1	Performance	- Furthermore, for both mobile and web clients, assets used will be compressed as much as possible to reduce response time without compromising granular details of the application. - - Moreover, heavy packages will be avoided as much as possible, and the use of lightweight packages that can perform the same feature or operation will be favored.
NFR_2	Security and Privacy	- The back-end server's security setup will ensure that unauthorized users access no data, and no user will receive data that was not meant for the user. - Client confidentiality principles like not talking to clients about their cases in public, making rational and moral decisions whenever you are in a situation where the confidentiality of your client's case is at stake, and so on will be followed. - Documents sent through the internet will be sent through secured portals.
NFR_3	Usability	- The applications (both mobile and web clients) will be designed with all possible user experience principles to make the applications easy to use, navigate and understandable. - Help texts will be placed where every necessary error message will be displayed with the cause and possible ways of fixing the errors. - All user interface features will be designed in such a way that users can know its use from first glance (using icons, colours, images, etc)
NFR_4	Availability	- Both the mobile and web applications will be up and running at all times. - The mobile application will be readily available on Google Play Store, while the web client will be readily available on the internet after development and deployment.

NFR_5	Reliability	- Each application function and feature on both application clients will be closely inspected, monitored and tested to ensure that it efficiently operates (s) it was meant to.
NFR_6	Responsiveness	- The mobile client will be able to run smoothly on approximately 84.9% of android devices. - The web client will be readily accessible from multiple devices regardless of the device size with the same look and feel.
NFR_7	Supportability	- Technical support personnel will be kept in place to listen, identify and solve any technical faults reported by users within the shortest time possible.

2.7. BLOCK DIAGRAM

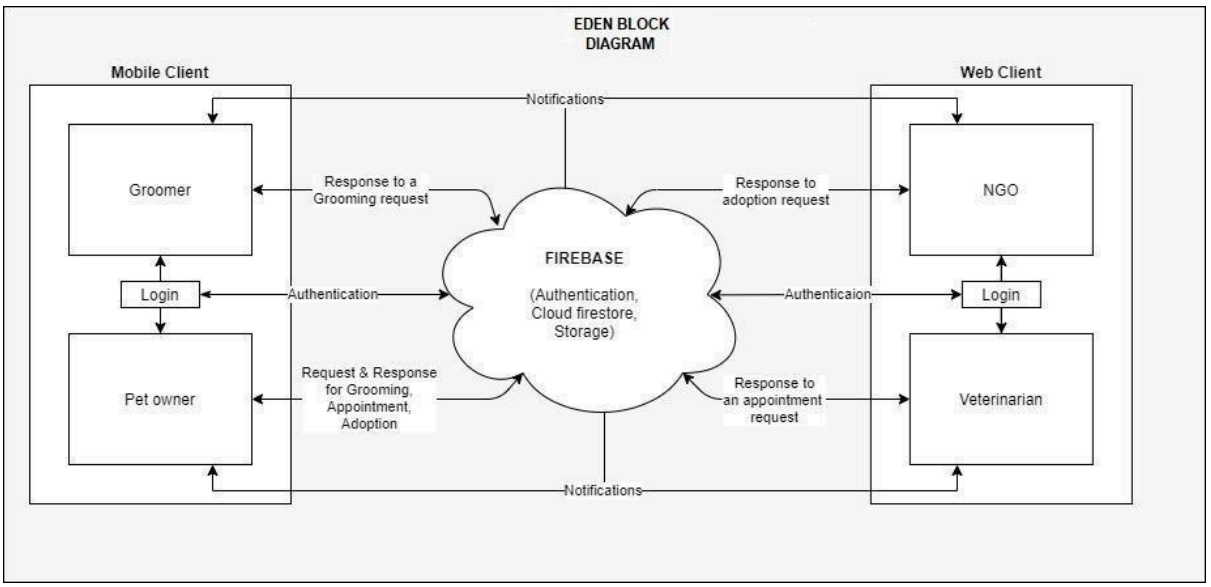


Figure 1. – Block Diagram - Complete System

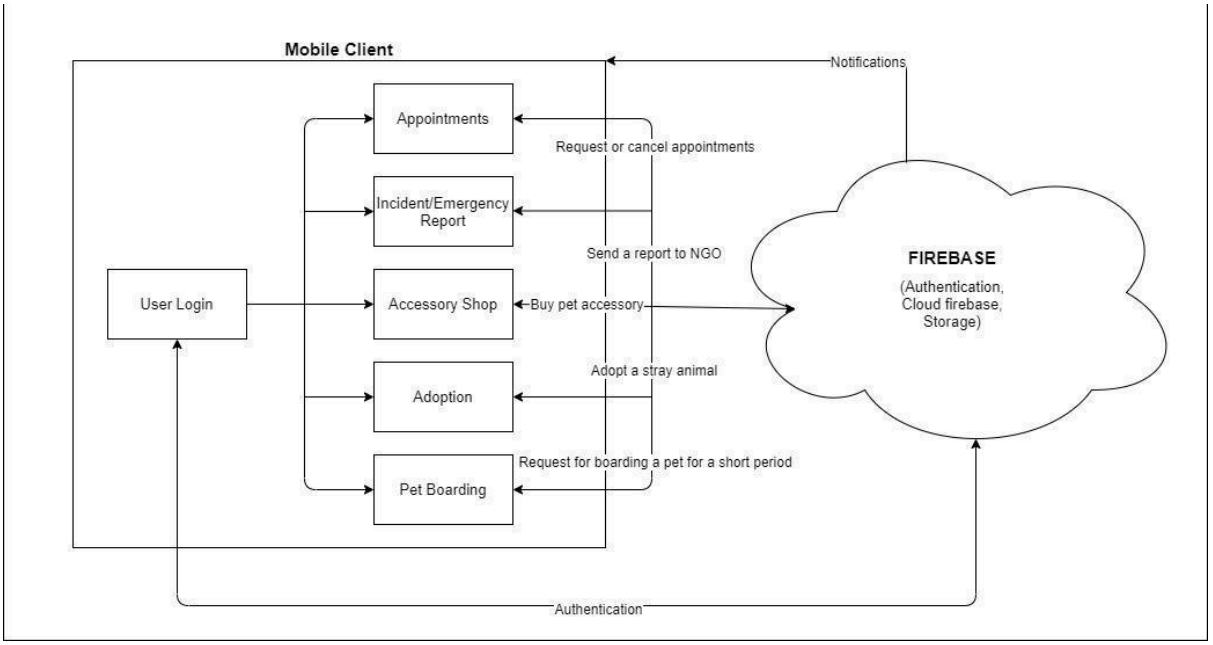


Figure 2. – Block Diagram – Mobile Client

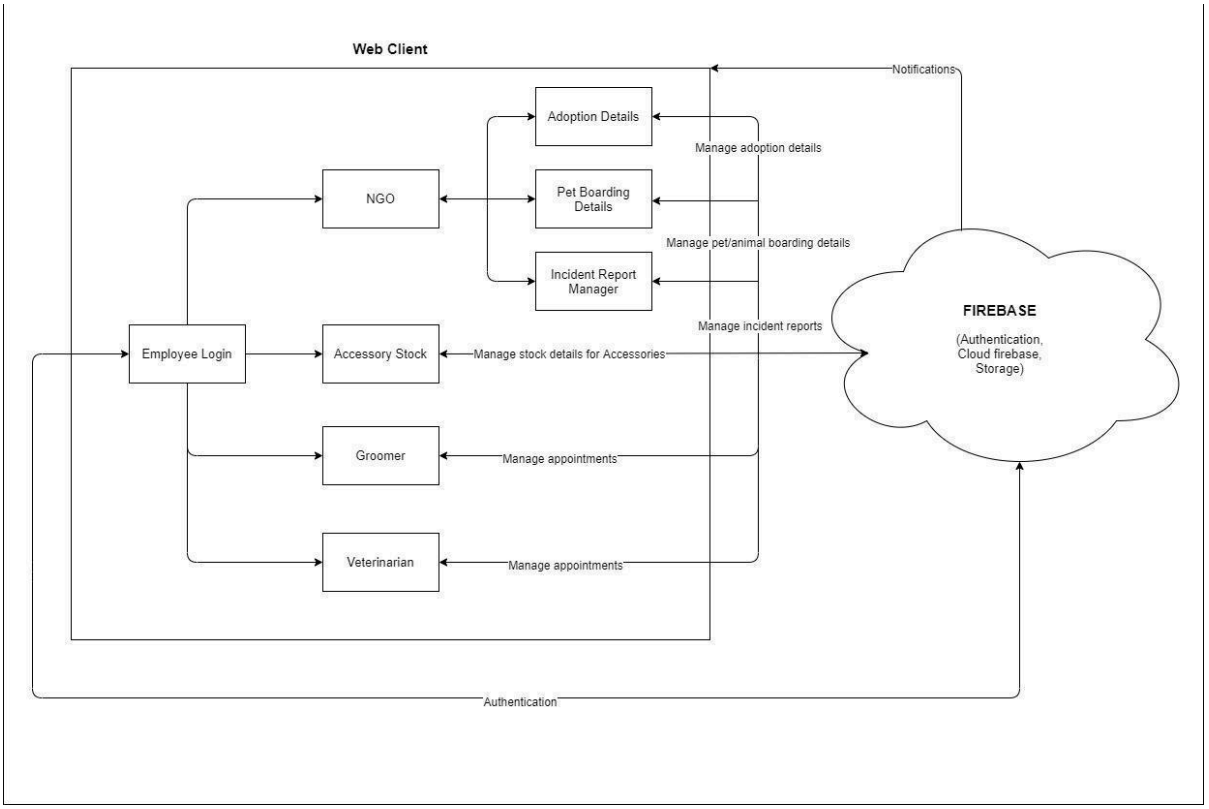


Figure 3. – Block Diagram – Web Client

3. SYSTEM DESIGN

3.1. SYSTEM ARCHITECTURE

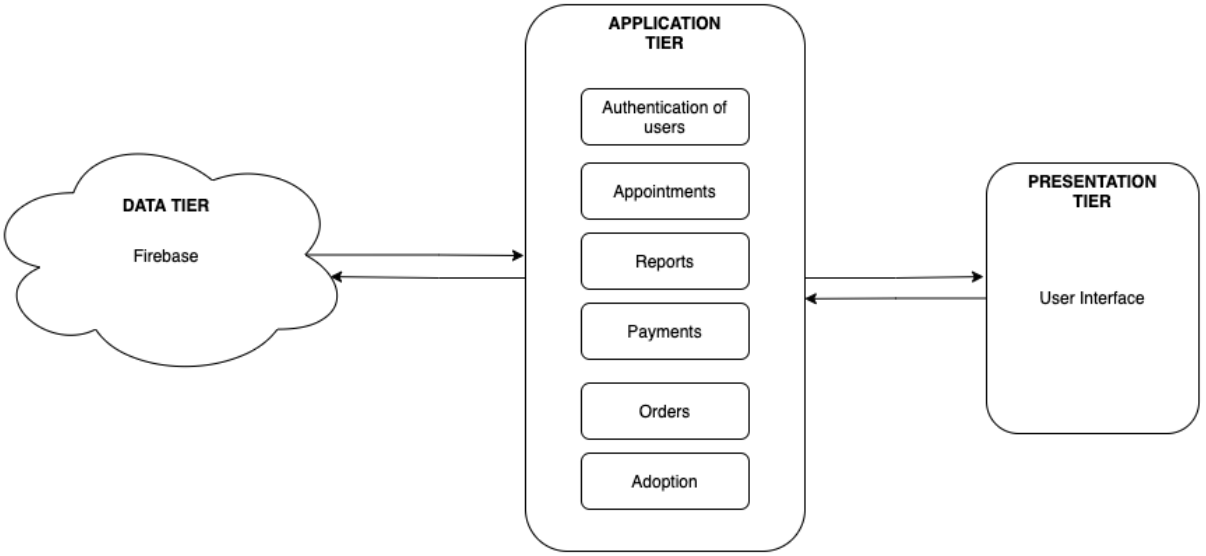


Figure 4. – EDEN System Architecture

The above diagram represents the system architecture of EDEN which is a centralized application which deals with all processes required for taking care of stray animals along with those owned by pet owners. The EDEN application system architecture is a three-tier architecture that uses an external infrastructure to function. The data tier (firebase) will store all data required for the functioning of our application, as well as data generated by the application.

Furthermore, the application tier (where the data is processed) is responsible for authenticating users (i.e. pet owners, NGOs, Veterinarians, and Groomers), fetching appointments and reports from the database, handling payment processes, orders and adoption requests. More so, the application tier will be responsible for inserting, updating and deleting data in the database.

Moreover, the presentation tier (a graphical user interface) serves as an interface for users to interact with the application. The presentation tier is divided into two sections; a mobile application and a website. The mobile application focuses on providing an interface for pet owners to interact with EDEN. On the other hand, the websites present an interface for the NGOs, Veterinarians and Groomers to interact with the application and accordingly operate based on information provided by the data tier.

3.2. MODULE DESIGN

EDEN has five main modules, namely; Mobile Client, NGOs, Veterinarians, Groomers and Accessories. These modules work hand in hand to form the entire project. Each module can be further divided into several sub modules. The following paragraphs discuss each module's functionalities and its sub modules.

The mobile client will be an Android app that will be specifically made for pet owners. It will act as a central hub for all the services required by a pet owner. The data used in this module will be synchronized with the web clients, thereby ensuring that the data will always be up to date across both platforms. This facilitates the end-users, groomers and veterinarians to schedule and manage appointments. It acts as a bridge that connects pet owners to groomers and veterinarians. Overall, this module aims at providing the essential services that are a requisite for owners of domestic animals.

Core Functionalities:

1. Ability to report animal abuse to the NGO (Non-governmental organization).
2. Track pending orders that have been ordered from the online pet store.
3. Schedule and manage appointments with the groomer or veterinarian.
4. Ability to adopt pets.
5. Get help and support from the "EDEN" team.
6. Timely notifications on updates related to appointments and orders.
7. Allows pet owners to find a temporary place of stay for their pet, where they will be taken care of when the owners are not in a situation to have it with them.

The Veterinarian module is developed to provide a user interface to employees working at the veterinarians in order for them to access and identify the appointments scheduled for the day. The data or records for the appointments will be entered by the pet owners or users via the mobile client. Employees may also create schedule appointments as per the requirements of pet owners provided; they possess an account with the pet application, EDEN. Additionally, the results of the examination will be entered by the Veterinarian after examining the pet/animal being brought in.

Core Functionalities:

1. Ability to create temporary accounts for customers with no EDEN account.
2. Review the appointments scheduled for the day, and those in the past.
3. Make changes to services rendered and accordingly adjust service amount price.
4. Schedule appointments based on availability.
5. Make cancellations to existing appointments and hence charge penalties.
6. Calculate the time required for the service rendered and calculate service amount price.
7. Access animal and/or owner profile to prepare groomer for service.
8. Allow Veterinarians to enter and review pet examination records.
9. Send Veterinarians list of appointments for the day.

The Groomer module will be developed to provide functionalities to view the schedules and appointments for the day and accordingly prepare for the animal grooming. Additionally, functionality is to be added which might calculate the price of the appointment. Cancellation option will also be available however, an additional fee will be charged on next appointment.

Core Functionalities:

1. Ability to create temporary accounts for customers with no EDEN account.
2. Review the appointments scheduled for the day, and those in the past.
3. Make changes to services rendered and accordingly adjust service amount price.
4. Schedule appointments based on availability.
5. Make cancellations to existing appointments and hence charge penalties.
6. Calculate the time required for the service rendered and calculate service amount price.
7. Access animal and/or owner profile to prepare groomer for service.

The NGO is a web application that mainly focuses on providing services like adoption, rescuing animals, contacting groomers/veterinarians and organizing events. It can be subdivided into six modules: alert, adoption, contact, events, fund and boarding pets. The main objective of this module is to provide shelter for animals and rescue them in case of an emergency. It also helps NGOs to organize events to raise funds for the NGO.

Core Functionalities:

1. Accept/Decline the abuse report made by the user.
2. Schedule and manage appointments with the groomer and veterinarian.
3. Organize events for animals and post the same for the end-users.
4. Update the animals for adoptions details.
5. Blocking the user if they are spanning the report section.

The Pet Accessories module is a web application that mainly focuses on providing e-commerce services to the application users (users include pet owners, veterinarians, NGOs, and groomers). As an e-commerce module, it comprises functionalities like product content management capabilities, customer management, order management, promotion and discount code management, An easy-to-use checkout, and reporting.

The pet accessories module can be subdivided into eight modules: checkout, shipping, payment, customer accounts, catalogue management, and product browsing, reports, and order management. These modules vary in size and are vital for the successful completion of the entire module.

3.3 DATA FLOW DIAGRAM

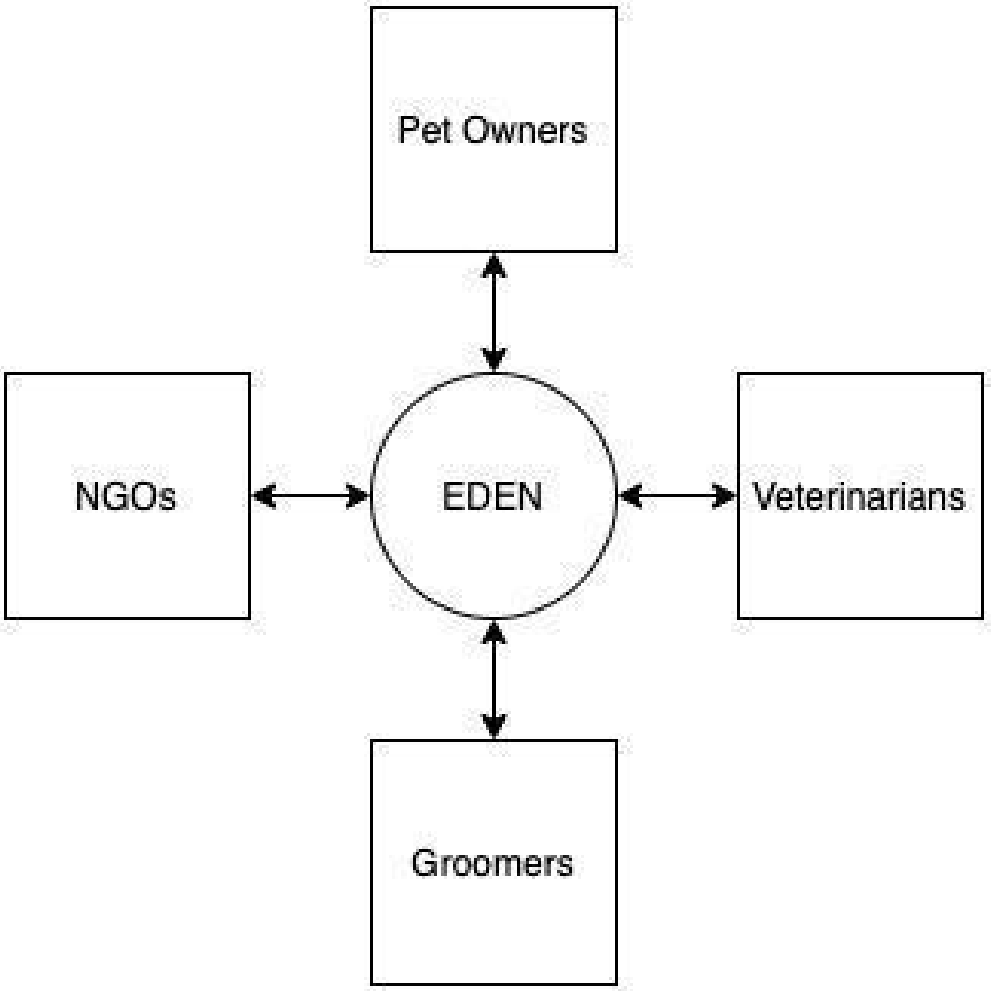


Figure 5. – DFD Level 0

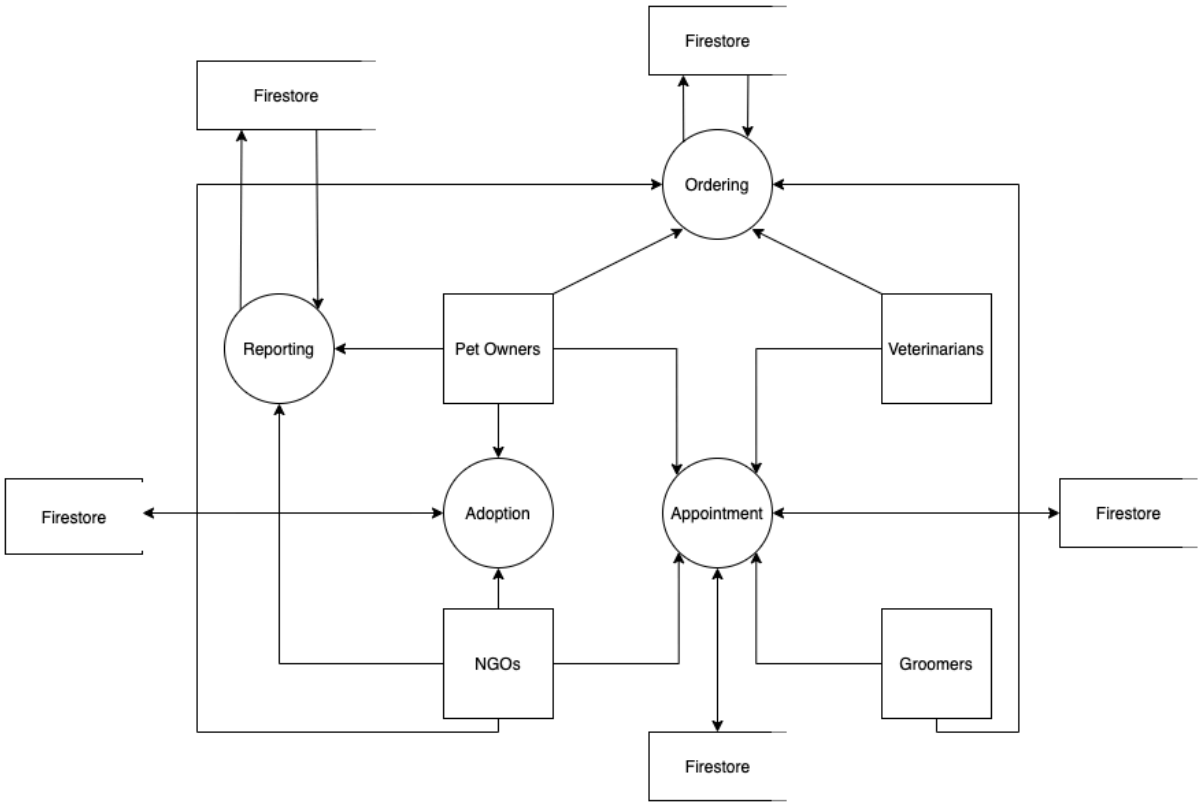


Figure 6. - DFD Level 1

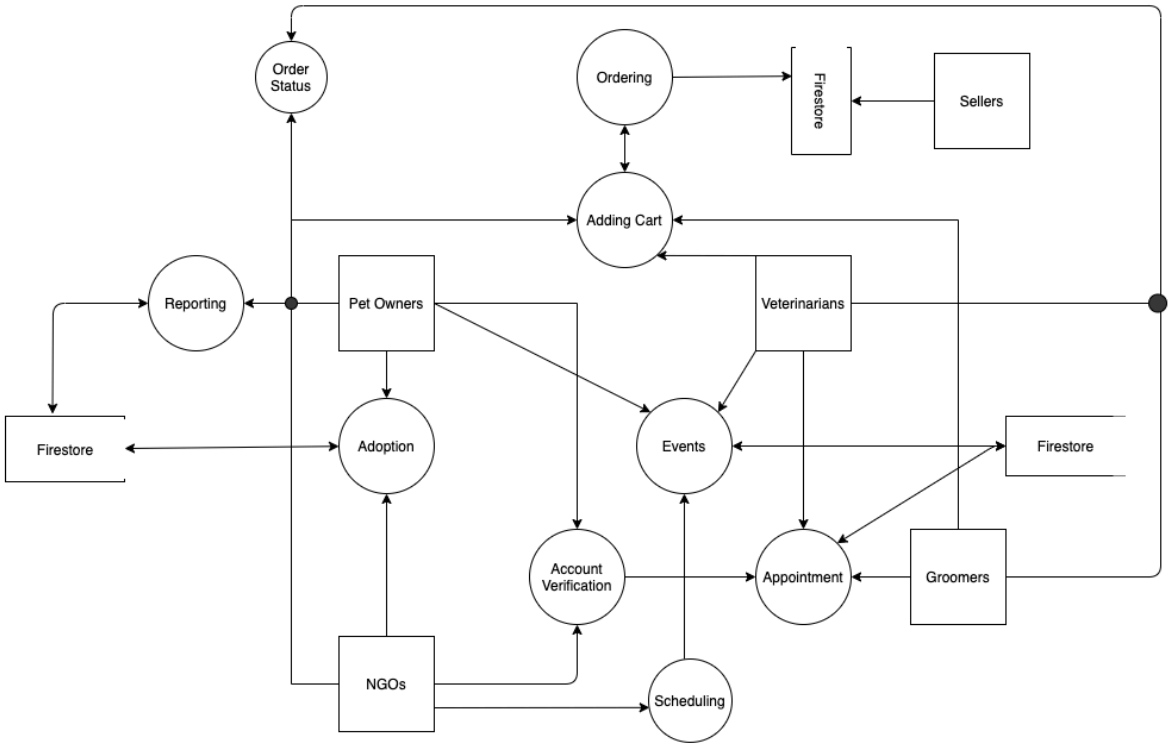


Figure 7. DFD Level 2

3.4 DOCUMENT DESIGN

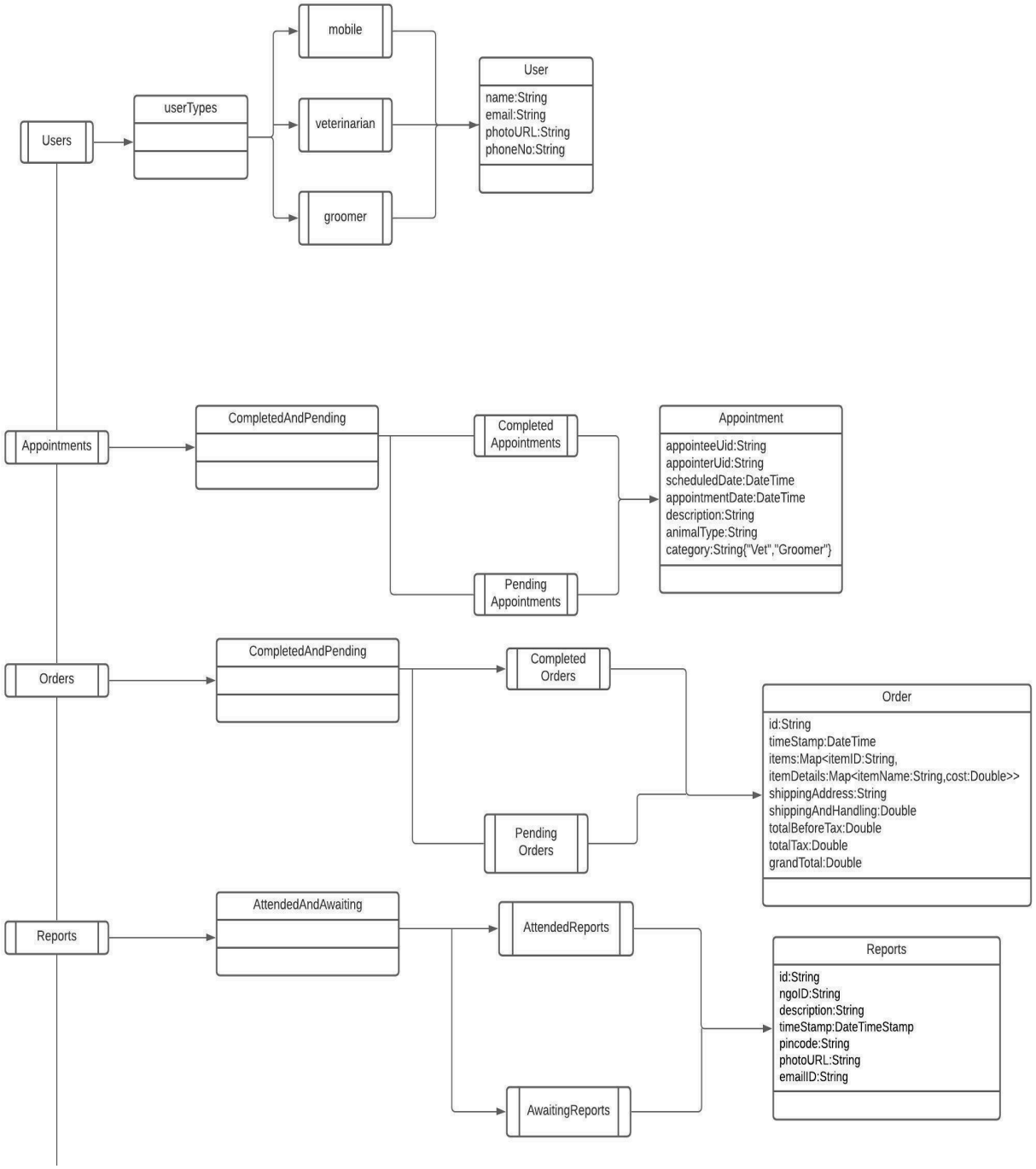


Figure 8. - Document Tree 1

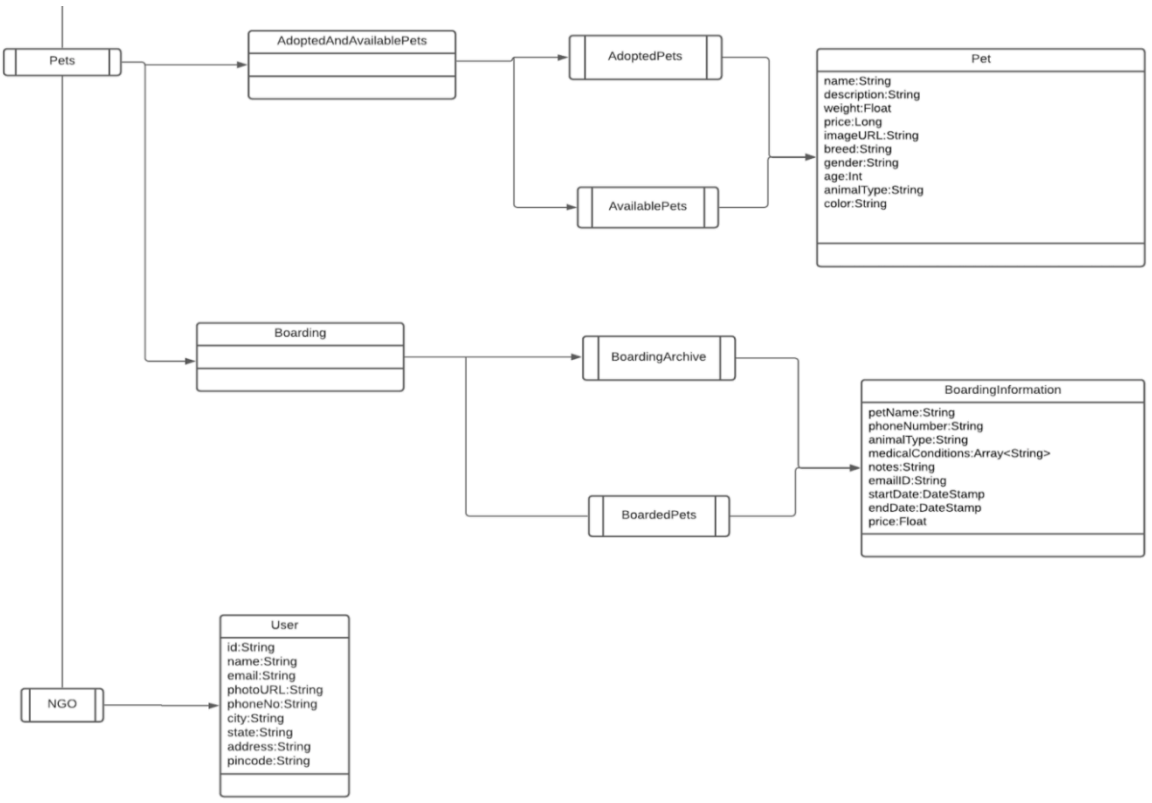


Figure 9. - Document Tree 2

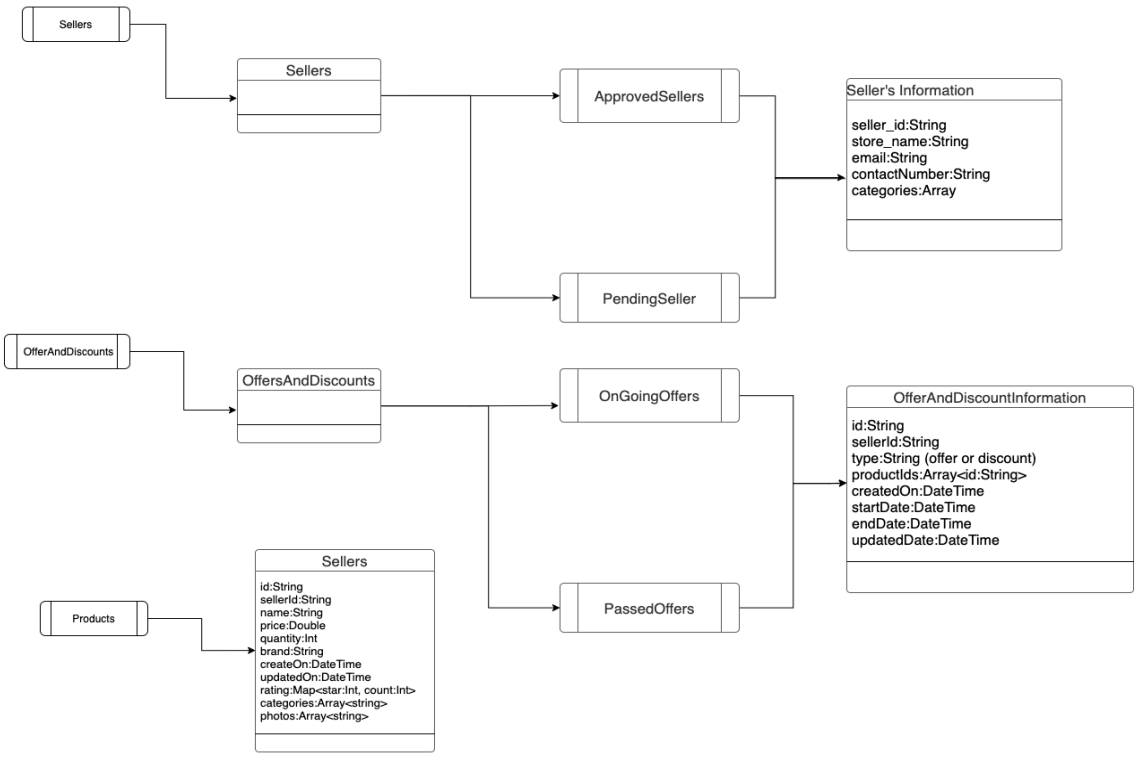


Figure 10. - Document Tree 3

3.5. DATABASE DESIGN

3.5.1. DATA INTEGRITY AND CONSTRAINTS

- Both the email and password must be valid. Both of them will be compared against a regular expression pattern. The email must conform to the standard email structure and the following are the constraints for the password.
    - It must have at least 1 upper-case letter.
    - It must contain at least 1 numerical char.
    - The length of the password must not be less than 8.
- The name of the user must not contain numeric values.
- The age of the domestic animal must be a positive natural number. It cannot be less than 1.
- The appointment date will not be allowed to contain a date value that is lower than the current day. The same is applicable for the time value.
- The appointment category will contain must be either veterinarian or groomer. It cannot have any other values.
- The phone number can contain a maximum of 10 digits. Phone numbers with more than 10 digits will not be allowed.
- The checkout date cannot contain a date value that is lower than the check-in date.
- The appointment category will contain must be either veterinarian or groomer. It cannot have any other values.
- For events scheduled by NGOs, the date should be greater than the date they are scheduling an event and the NGO is allowed to schedule events only between the scheduling date +2 and two months from the date they are scheduling. All the fields will be necessary to be filled.
- More so, the appointment date must be the current date or greater than. The NGO is allowed to schedule appointments only between the appointment date and a week from the appointment date. One animal must be chosen and not more than 3 animals should be chosen for one appointment by the NGO. All the fields will be necessary to be filled.
- Furthermore, to add new animals for adoption, all the fields will be necessary to be filled. The NGO will also have an edit option to update the data of animals who are already up for adoption if needed.
- More so, the Alert and Pet Boarding page deal with user requests to the NGO. The NGO will be able to accept or decline the user request as per the situation. In case of any spam/inappropriate request made by the user, the NGO will be able to block the user.
- For registering a seller, they must fill in all the input fields. Also, the phone number must be 10 digits long (without any special characters or letters). Moreover, all emails will be validated and email verification will be enabled (that is to check whether the email is a working email). Furthermore, users will be allowed to select from a list of categories in order to limit variations in product category names.
- For product registration, sellers will be allowed to save a product only when they fill in all the required information. Furthermore, all prices and quantities must be numeric. That

is no product will be saved if the price or quantity contains special characters or letters (except a point (.) for decimal values).

- More so, the offer/discount start date and time must be equal to or greater than the current date and time. Also, the offer/discount end date and time must be equal to or greater than the start date and time. Moreover, sellers will be allowed to set offers and discount only for products currently in stock.

3.5.2. DATA DICTIONARY

MOBILE CLIENT

Table 4. – Mobile Client - Adoption Screen

Price	Label	String
Image	Image Box	Image
Name	Label	String
Breed	Label	String
Gender	Label	String
Age	Label	Int
Animal Type	Label	String
Color	Label	String
Weight	Label	Float
Description	Label	String

Table 5. – Mobile Client - Login Screen

Email	Text Box	Alphanumeric
Password	Password Box	Alphanumeric

Table 6. – Mobile Client - Sign-up Screen

Name	Text Box	String
Email	Text Box	Alphanumeric
Password	Password Box	Alphanumeric

Table 7. – Mobile Client - Appointments Screen

Description	Text Box	String
Animal Type	Text Box	String
Appointment Date	Date Time Picker	Long
Scheduled Date	Label	String
Appointment Category	Radio Button	String (Groomer/Vet)

Table 8. – Mobile Client - Donations Screen

Amount	Text box	Double
Phone Number	Text box	Long
NGO Name	Textbox	String

Table 9. – Mobile Client - Pet Boarding Screen

Pet Name	Text box	String
Phone Number	Text box	Long
Animal Type	Text box	String
Medical Conditions	Text box	String
Special Instructions	Text box	String
Check-in-date	Date/Time picker	Long
Check-out-date	Date/Time picker	Long

VETERINARIAN

Table 10. – Veterinarian - Employee Login Screen

Email	TextBox	Alphanumeric
Password	Password Box	Alphanumeric

Table 11. – Veterinarian - Scheduled Appointments Screen

Appointment ID	Label	String
Pet Owner	Label	String
Pet Name	Label	String
Appointment Date	Label	Date
Appointment Time	Label	Time
Animal Type	Label	Date
Reason for Appointment	Label	String
Status	Label	String (Pending/Cancelled/Missed/Completed)

Table 12. – Veterinarian – Temporary Account Creation Screen

Temporary ID	Label	String
Temporary Password	Label	String
Email	Textbox	String
Owner First Name	Textbox	String
Owner Last Name	Textbox	String
Pet's Name	Textbox	String
Animal Type	List	String
Animal Age	Textbox	Int
Animal Breed	Textbox	String

Table 13. – Veterinarian - Animal Examination Results

Appointment ID	Label	String
User ID	Label	String
Examination Results	Text Area	String
Treatment Details	Text Area	String
Medication Details	Text Area	String
Vets Note	Text Area	String

GROOMER

Table 14. – Groomer - Employee Login Screen

Email	TextBox	Alphanumeric
Password	Password Box	Alphanumeric

Table 15. – Groomer - Scheduled Appointments Screen

Appointment ID	Label	String
Pet Owner	Label	String
Pet Name	Label	String
Appointment Date	Label	Date
Appointment Time	Label	Time
Animal Type	Label	Date
Groomer Treatments	Label	String
Service Cost	Label	Float
Status	Label	String (Pending/Cancelled/Missed/Completed)

Table 16. – Groomer - Temporary Account Creation Screen

Temporary ID	Label	String
Temporary Password	Label	String
Email	Textbox	String
Owner First Name	Textbox	String
Owner Last Name	Textbox	String
Pet’s Name	Textbox	String
Animal Type	List	String
Animal Age	Textbox	Int
Animal Breed	Textbox	String

Table 17. – Groomer - Animal Examination Results

Appointment ID	Label	String
User ID	Label	String
Treatment Details	Text Area	String
Treatment Cost	Label	String
Groomer Note	Text Area	String

NGO

Table 18. – NGO - Event page

Title	Textbox	String
Event Date	Date-Time	String
Event Type	Drop Down	String
Event Type filter	Drop Down	String
Date filter	Date-Time	String

Table 19. – NGO - Appointment Page

Date	Date	String
Animal Type	Checkbox	String
Description	Textbox	String
Category	Radio Button	String
Category filter	Checkboxes	String
Animal filter	Checkbox	String
Date filter	Date	String

Table 20. – NGO - Adoption Page/Pet Boarding Page

Photo	Cards	Image
Animal Name	Textbox	String
Breed	Textbox	String
Gender	Radio	String
Colour	Textbox	String
Weight	Textbox	Double
Description	Textbox	String
Price	Textbox	Double
Contact Number	Text	String
Animal Type	Text	String
User Email Id	Text	String
Start Date	Text	String
End Date	Text	String

Table 21. – NGO - Alert page

User Email id	Text	String
Description	Text	String
Pin code	Text	Numeric
Address	Text	String
Photo	Card	String

PET ACCESSORIES

Table 22. – Pet Accessories - Landing Page, OneAnimal, Product Page, Shopping Cart Page, Orders Page

Products	Cards	Image
Offers	Cards	Image
Price filter	Double-ended range slider	String
Brand filter	Checkboxes	String
Sellers filter	Checkboxes	String
Rating filter	Double-ended range slider	String
Discount filter	Double-ended range slider	Double
Sellers name	Text	String
Price	Text	Double
Product name	Text	String

Table 23. – Pet Accessories - Check out Page

Shipping Address	Textbox	String
Promotional codes	Textbox	Numeric
items	Cards	Alphanumeric
shippingAndHandling	Text	Double
totalBeforeTax	Text	Double
totalTax	Text	Double
grantTotal	Text	Double

3.6 INTERFACE AND PROCEDURAL DESIGN

3.6.1 USER INTERFACE DESIGN

EDEN is made up of several modules, with each module defining a set of different users. The following paragraphs describe the users related to each module, the tasks to be performed, the environment and how a user interface will be built.

MOBILE CLIENT

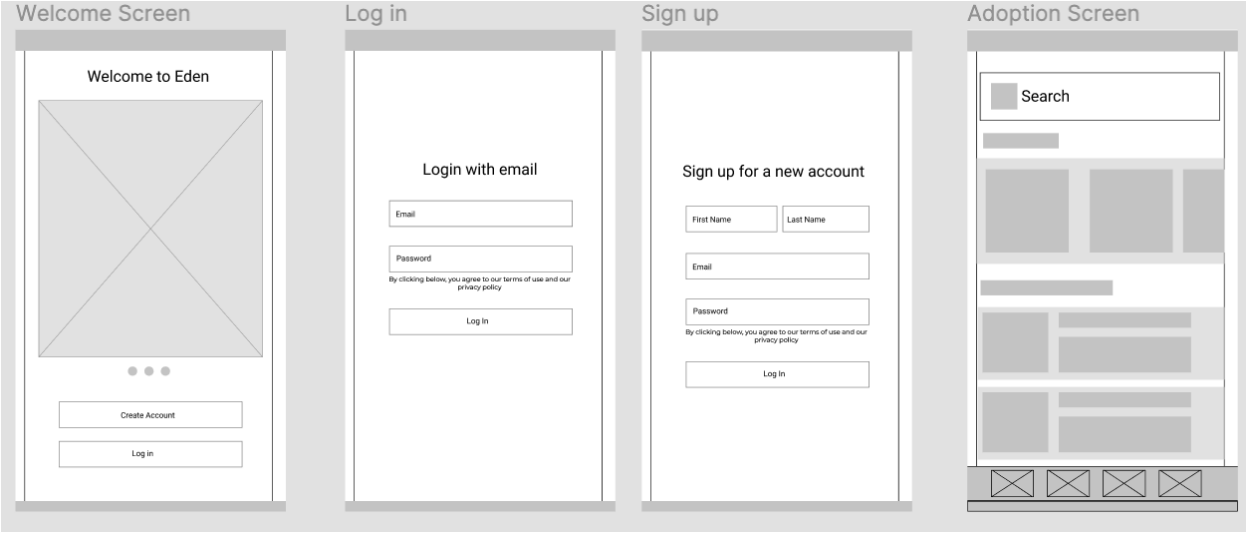


Figure 11. Mobile Client Wireframe 1

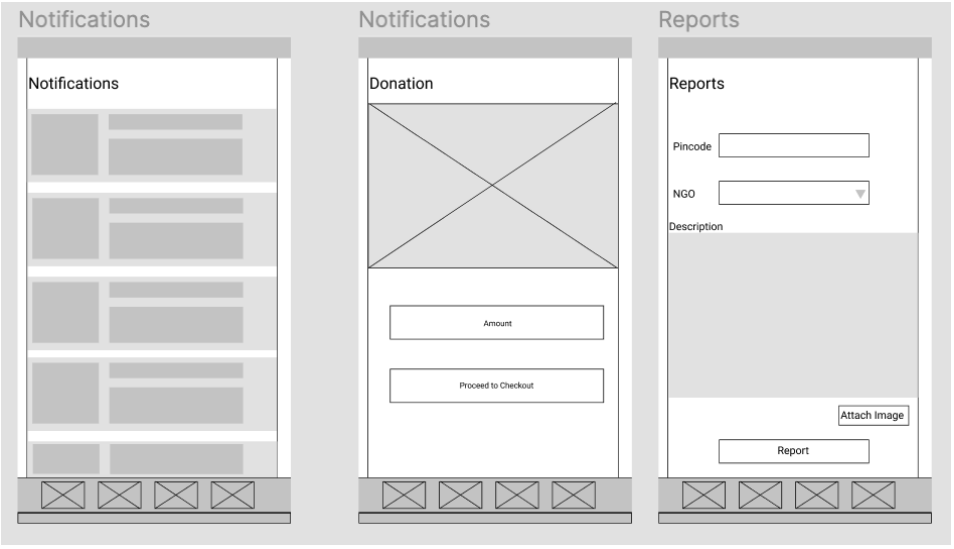


Figure 12. Mobile Client Wireframe 2

GROOMER

The Groomer's module will be available only for the Web application and hence will be used only by the employees, especially by the Groomer employees. The user interface in this module will be design to support operations to assist the employees in daily work such as determining the appointments for the day, calculating the total price for an appointment to be paid, etc.

As such, the module will contain nearly 5 to 6 user interfaces, which will display retrieved data from the data tier or be used to enter or modify data in the data tier.

The Employee Login interface will be common for both modules i.e., Groomer and Veterinarian. After successfully logging in, a dashboard that will have a summary report will be visible. Options will be available to interact with the reports to determine which report is to be viewed. Additional user interfaces will include an appointment list for the day where all the scheduled appointments will be visible to the concerned viewer. Filters will be available to traverse the dates to see the appointments or to check the appointments based on their statuses namely pending, cancelled, completed or missed.

A user interface will also be available where the employees can create a temporary Account for clients provided, they don't possess an earlier account with EDEN. The temporary ID and password will be sent to the concerned client wherein they must change their username and password and fill out their profile data before any appointments can be scheduled for them.

Once a client account is available, two interfaces will be available namely the schedule appointment and cancellation interfaces. The latter will be available for all appointments in the data tier and for those with status 'Pending'. As such a client can register for an appointment using the mobile client; however, options are available for the groomers to create appointments on their client's behalf.

Treatment calculation and additional notes (if any) can be filled out once an appointment is completed. These are to be filled out by the groomer attending the client's pet or the NGO's animal.

The following are UI screens for the Groomer's module: -

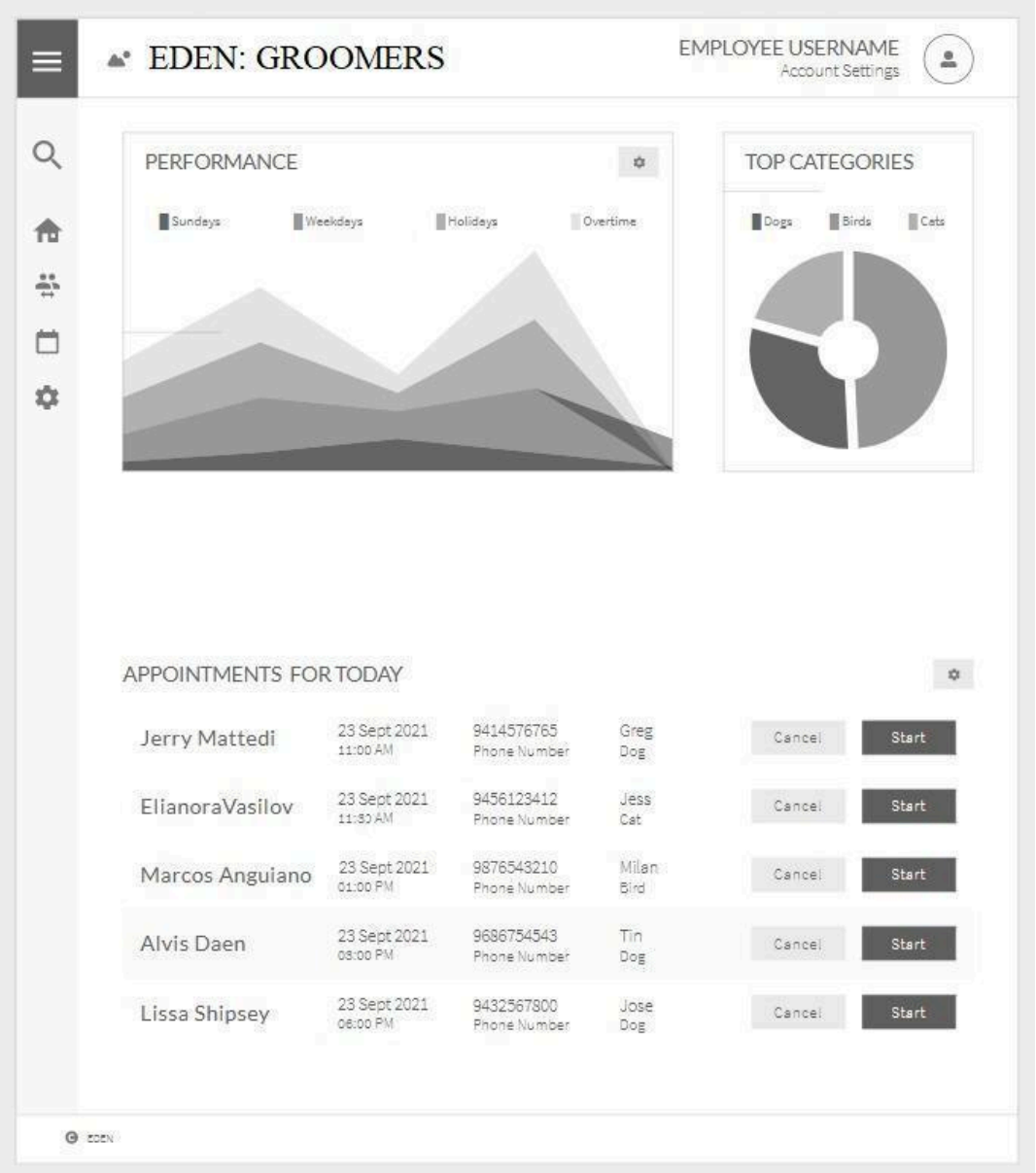


Figure 13. Appointments & Reports for Groomer

VETERINARIAN

The Veterinarian's module will be available only for the Web application and hence will be used only by the employees, especially by the Veterinarian employees. The user interface in this module will be design to support operations to assist the employees in daily work such as determining the appointments for the day, storing details of an animal's examination, treatment details, etc.

As such, the module will contain nearly 5 to 6 user interfaces, which will display retrieved data from the data tier or be used to enter or modify data in the data tier.

The Employee Login interface will be common for both modules i.e., Groomer and Veterinarian. After successfully logging in, a dashboard that will have a summary report will be visible. Options will be available to interact with the reports to determine which report is

to be viewed. Additional user interfaces will include an appointment list for the day where all the scheduled appointments will be visible to the concerned viewer. Filters will be available to traverse the dates to see the appointments or to check the appointments based on their statuses namely pending, cancelled, completed or missed.

A user interface will also be available where the employees can create a temporary Account for clients provided, they don't possess an earlier account with EDEN. The temporary ID and password will be sent to the concerned client wherein they must change their username and password and fill out their profile data before any appointments can be scheduled for them.

Once a client account is available, two interfaces will be available namely the schedule appointment and cancellation interfaces. The latter will be available for all appointments in the data tier and for those with status 'Pending'. As such a client can register for an appointment using the mobile client; however, options are available for the groomers to create appointments on their client's behalf.

Treatment calculation, animal medical examination reports and additional notes (if any) can be filled out once an appointment is completed. These are to be filled out by the veterinarian attending the client's pet or the NGO's animal.

The following are UI screens for the Veterinarian modules: -

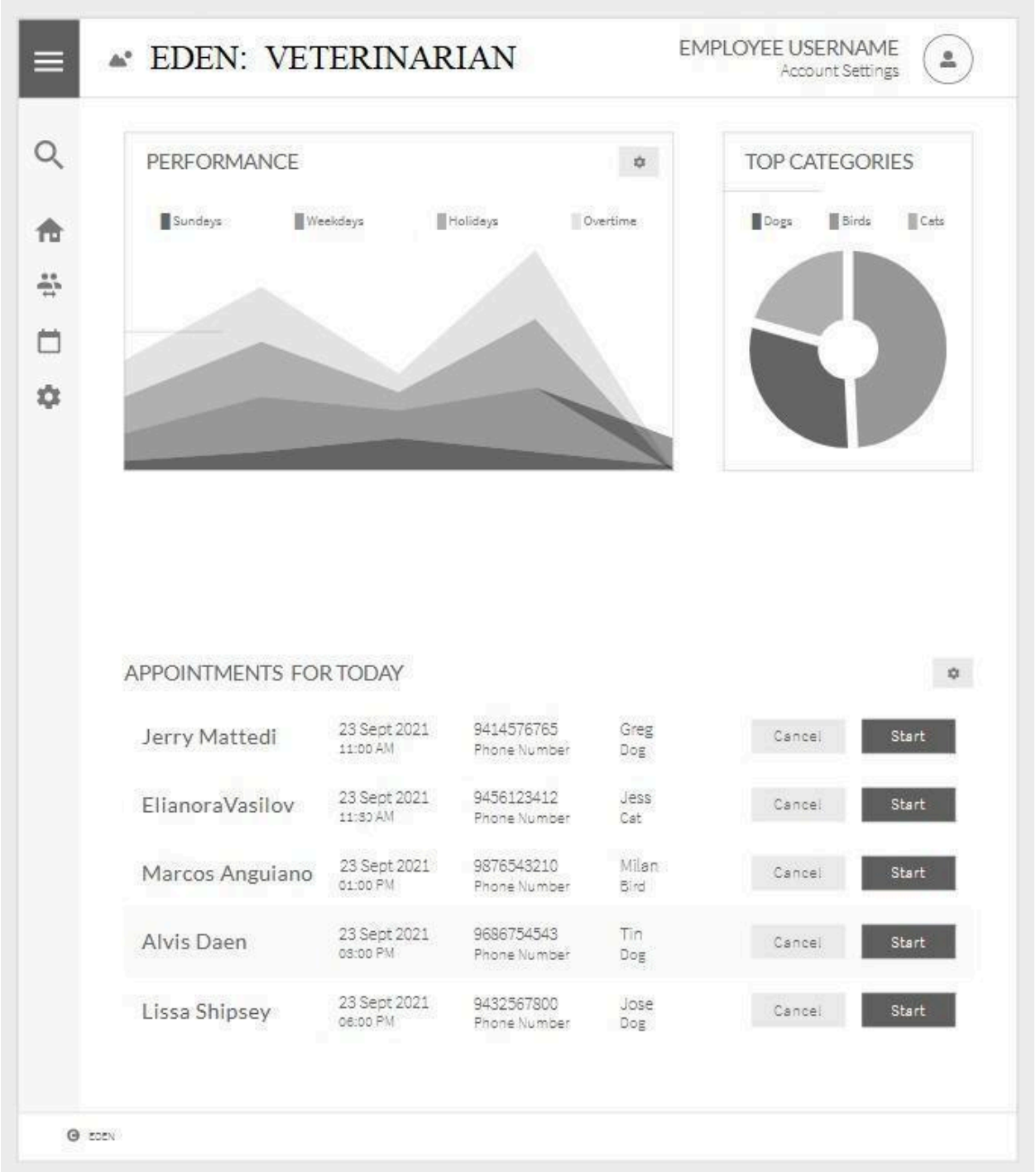


Figure 14. Appointments & Reports for Veterinarian

VETERINARIAN AND GROOMER

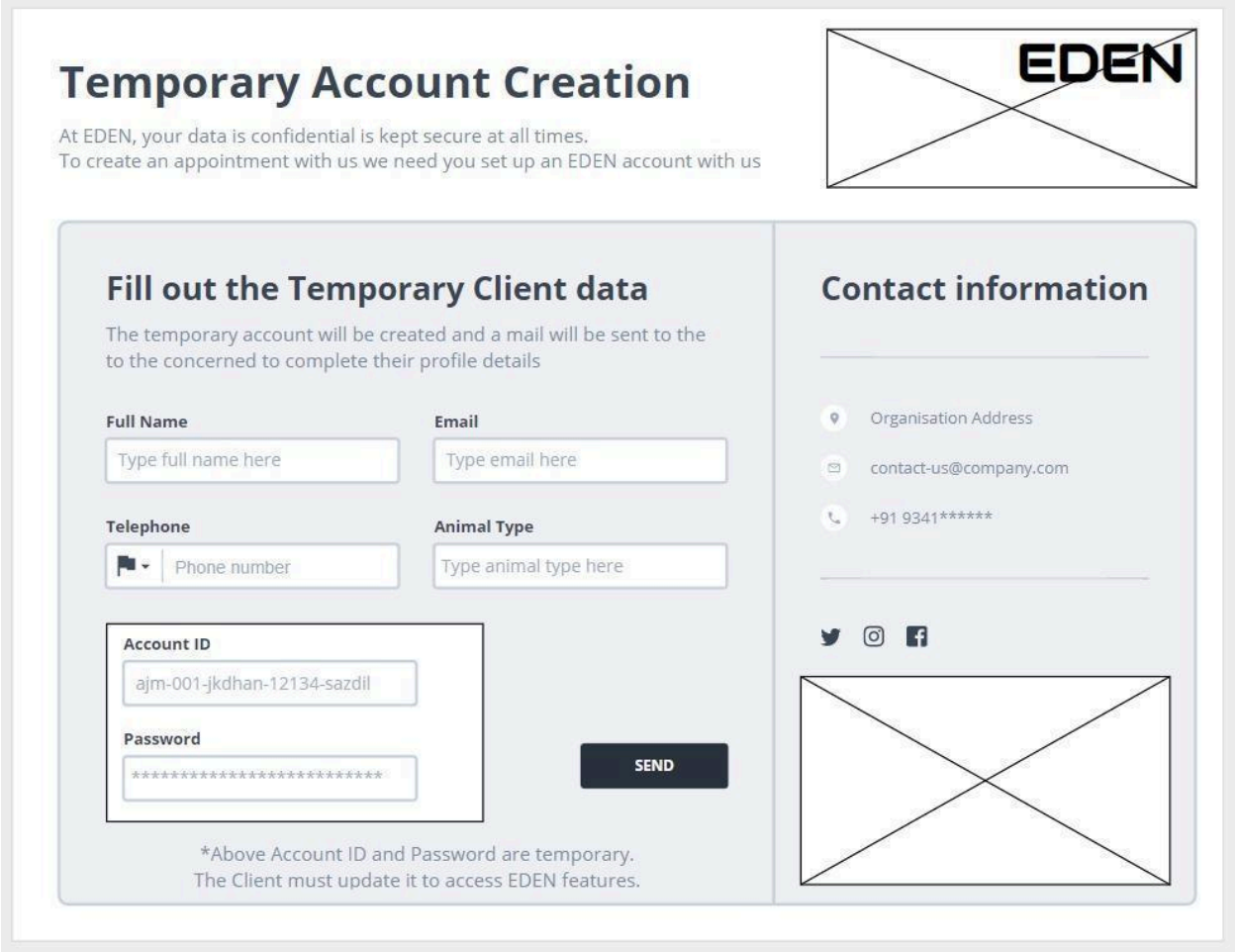


Figure 15. Temp Account Creation for new Clients not previously registered on EDEN

NGO

The NGO module is a website centered on helping animals, it allows a user to adopt an animal, appoint a groomer and vet for regular visits and raise funds by organizing events

The main tasks performed by users on the website are; signing in, checking if any new report is made by a user(accept/decline), check old report, add new animal up for adoption, check history for adoption, scheduling appointments with doctors or groomer, providing shelter to a pet when owners are out of town, organize events and check old events.

For the tasks mentioned above, we'll have seven user interfaces screens. After signing in, the user will be presented with a home page that is a dashboard that will have a summary report of all the services carried out by the NGO and the recent alert (if any).
There will be different UI screens having the same layout for viewing old events, adoption, appointment booked, abuse reports received. It will also have a filter option to filter data according to the date, animal type, or weekly.

Furthermore, when NGO wants to add new data like organize a new event, add animals for adoption and make a new appointment, a separate UI screen will be designed. This screen

will have a form like structure and the user will need to add necessary information with respect to the sub module.

For viewing new reports or user requests for taking care of their pet the NGO will be able to accept or decline the request. The NGO will also be able to block a user in case of spam requests.

The following are UI screens: -

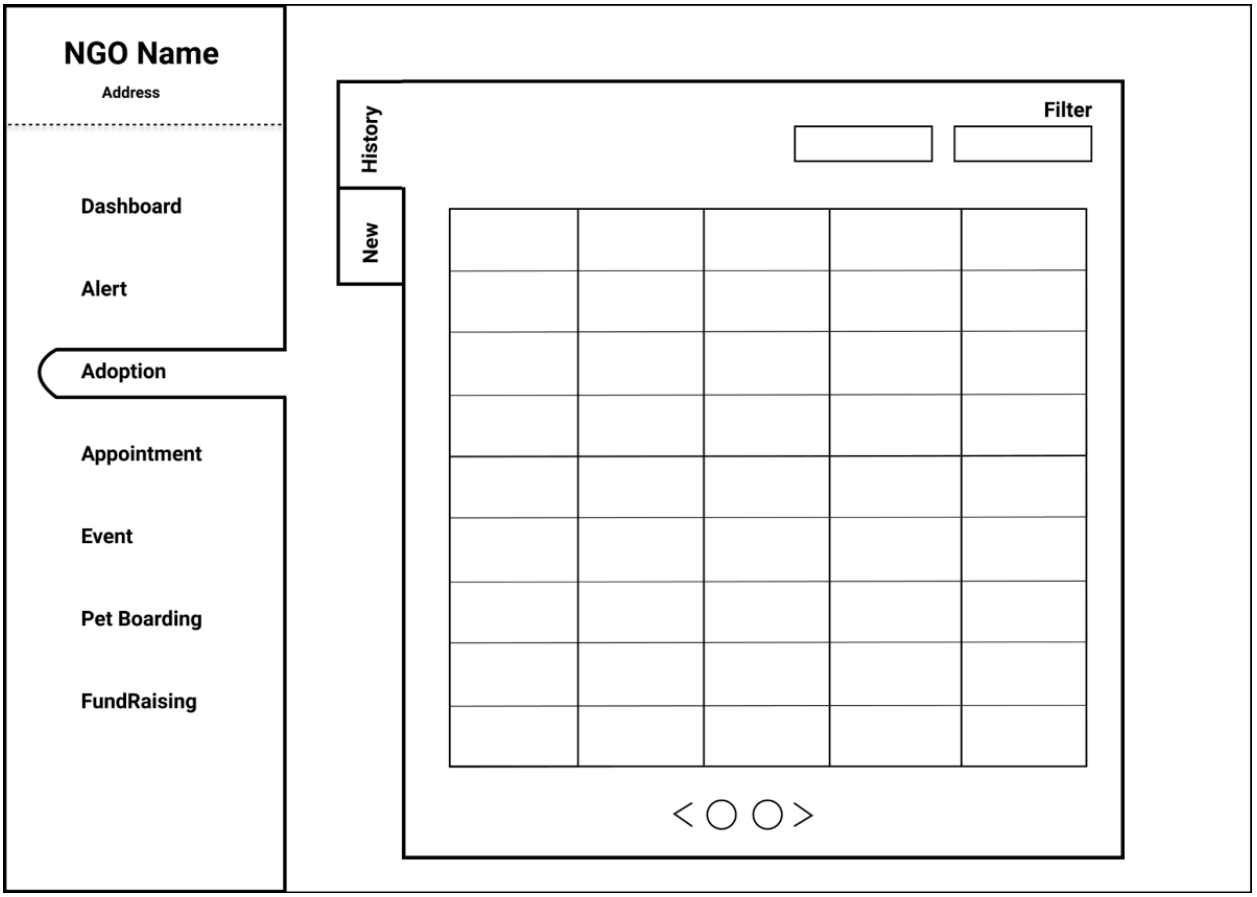


Figure 16. NGO History Page Layout

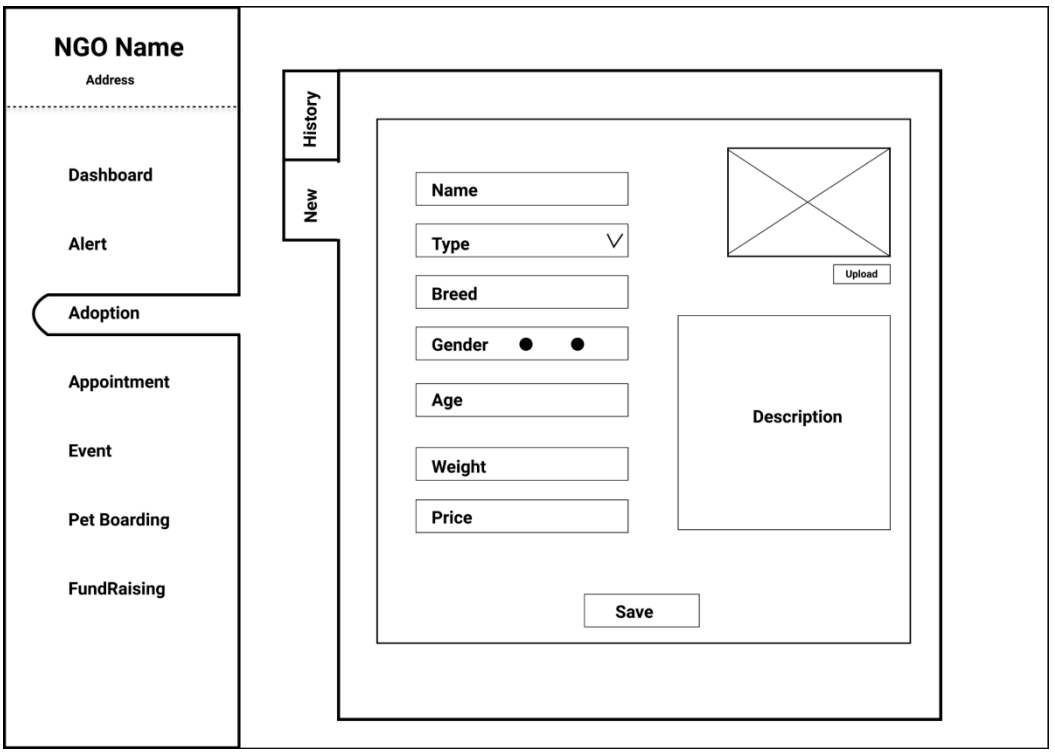


Figure 17. NGO Data Entry Page Layout

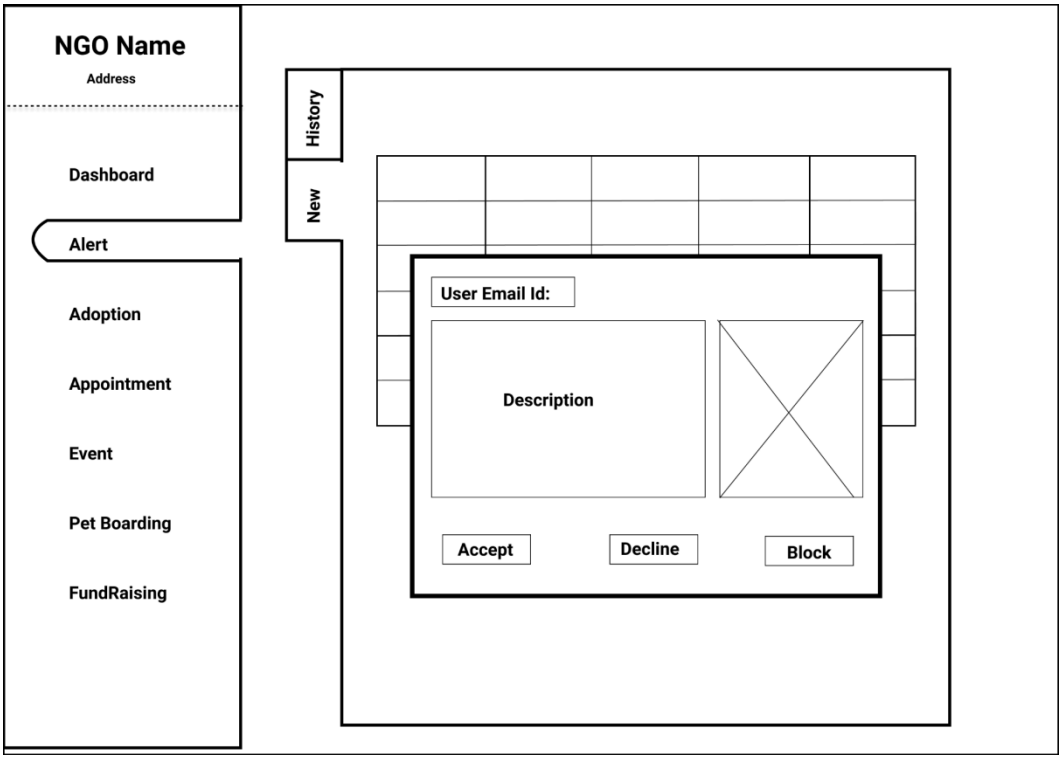


Figure 18. NGO User Requests Page Layout

PET ACCESSORIES

The pet accessories module is an e-commerce website that caters for any user who wishes to purchase food suppliers, grooming equipment and other pet-related accessories.

The main tasks performed by users on the website are; signing in, browsing through products on the website, adding products to their cart, checking out the products, reviewing the products, paying the required amount and placing their orders.

For users to perform the tasks mentioned above, roughly five user interfaces screens will be designed to handle a group of tasks. After a user signs in either through the website or through the other modules, the user will be presented with a home page containing some information about the categories or types of available pet products. A search box and filter sidebar will be provided to allow users to browse through the products quickly. The search box provides category wise searching. More so, products can be filtered according to price, brand, discount, and rating.
Furthermore, there will be separate UI screens of the same layout for each pet food and accessories category. Also, each product will be associated with an add to cart button. By clicking on add to cart button, users can add products to their cart.

To manage a user's cart, a separate UI screen will be designed. This screen will display a list of products added to the cart by the user and a summary of the total products and the total price. More so, a proceed to checkout; a button will be provided so that users can move on with their checkout from the cart page.

For the checkout page, the user will be required to enter relevant details related to shipping and payment. Also, the user, after reviewing the products, can place an order.

Moreover, it is worth noting that users can see a detailed description of a product by simply clicking on the product card, which will take them to a UI screen containing details about the product and some related and associated products.

The following are UI screens to help the programmer during implementation.

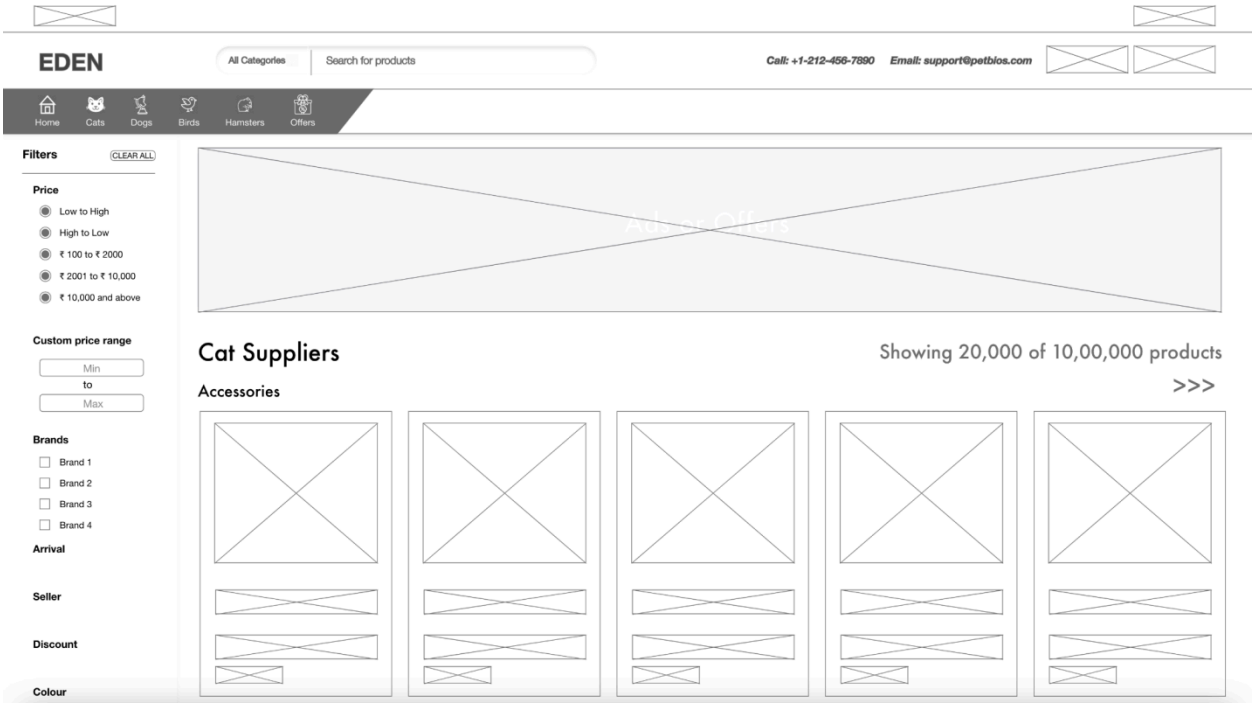


Figure 19. Pet Accessories Category Page

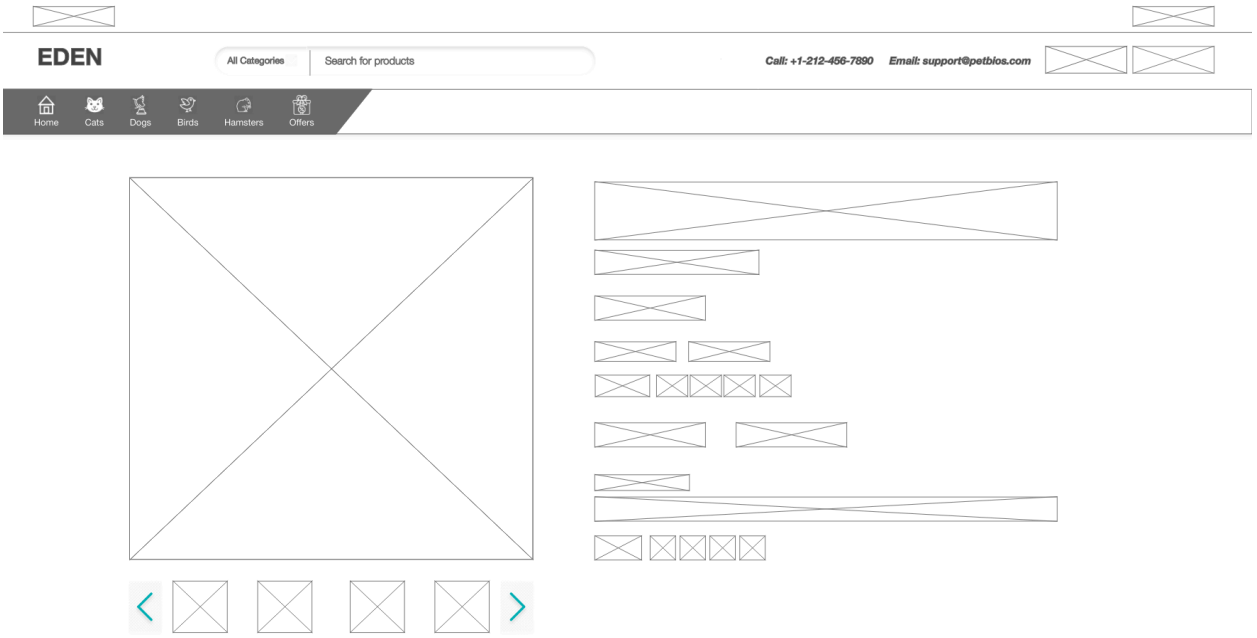


Figure 20. Pet Accessories Product Details

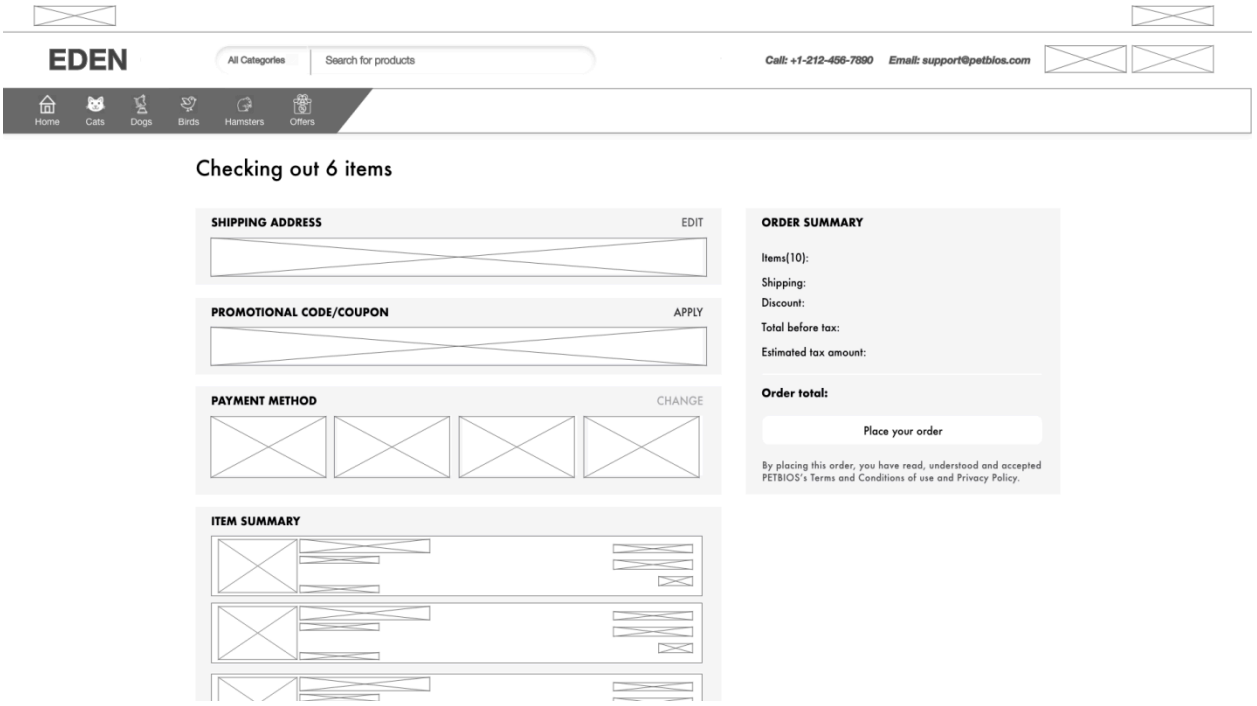


Figure 21. Pet Accessories Checkout Page

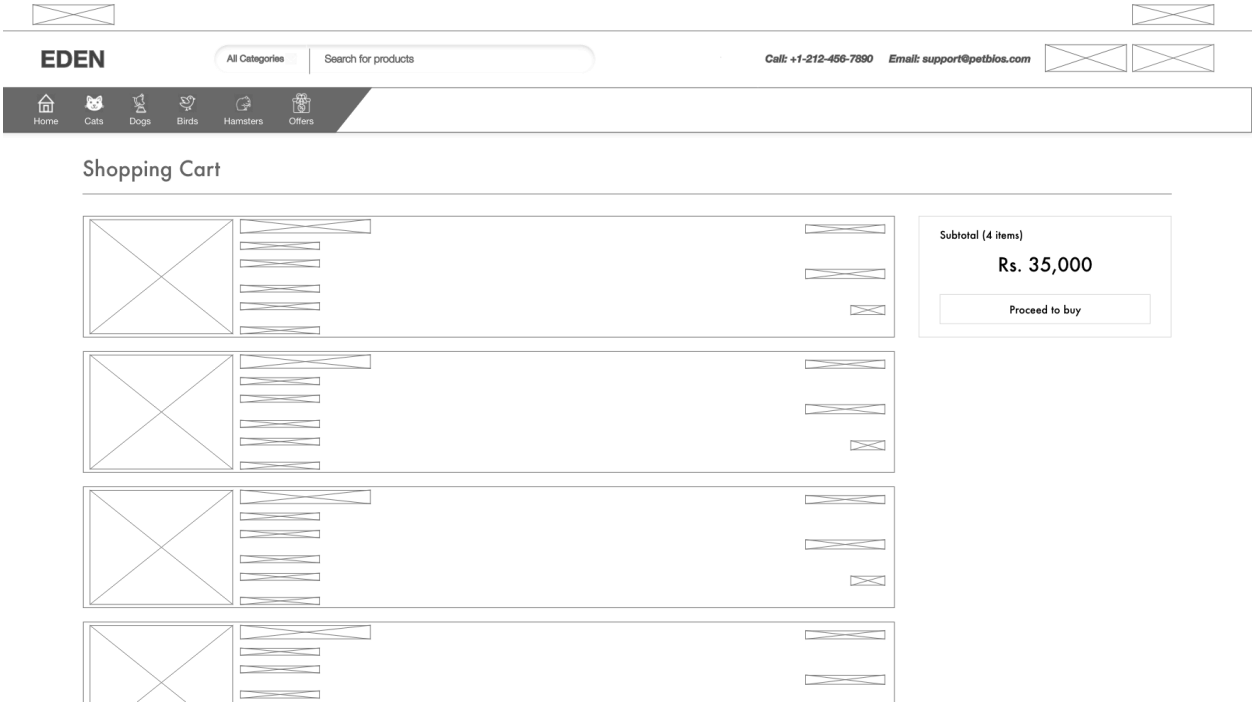


Figure 22. Pet Accessories Shopping Cart

3.6.2 PROCEDURAL DESIGN

NGO

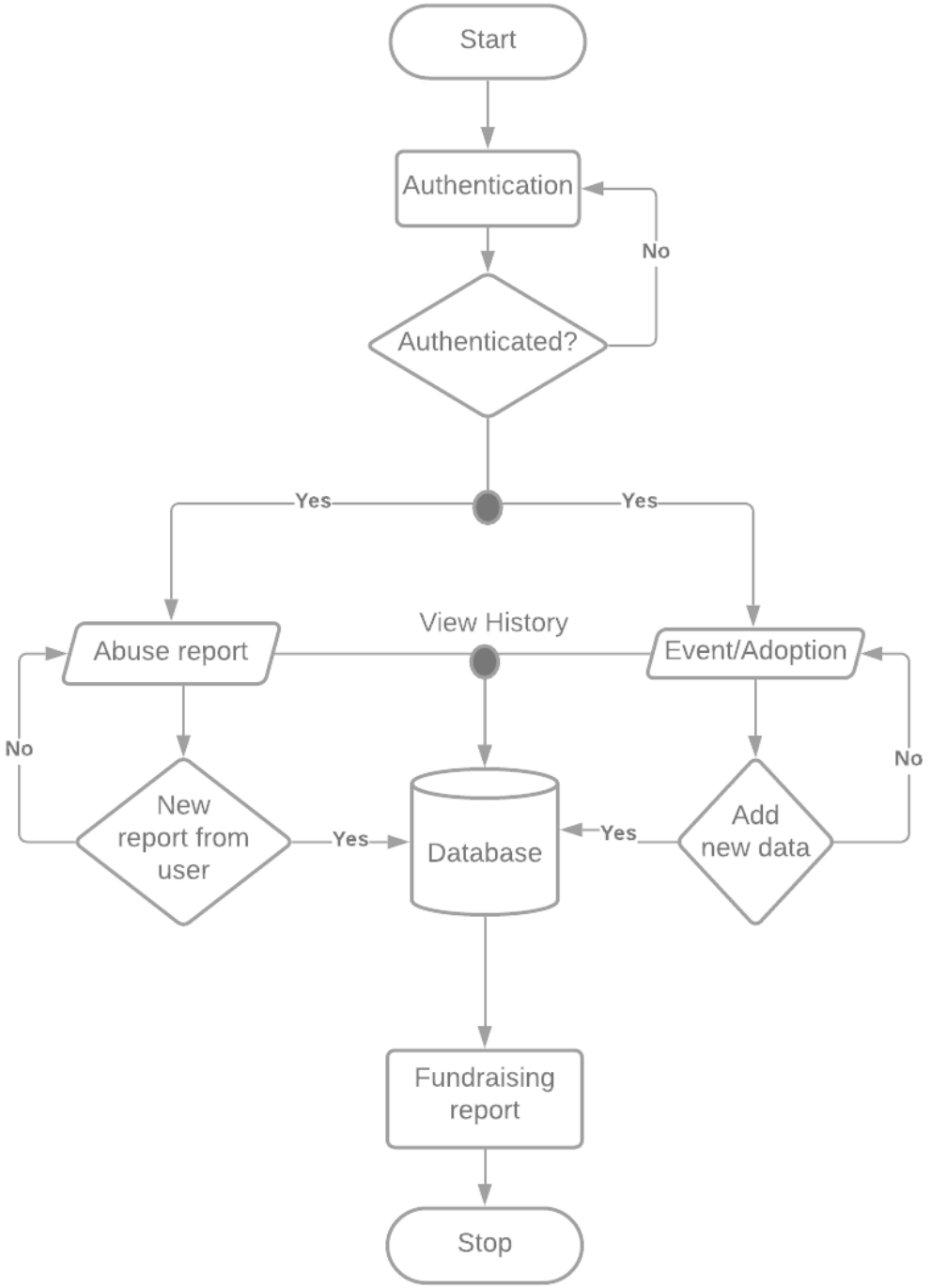


Figure 23. NGO Procedural Design

MOBILE CLIENT

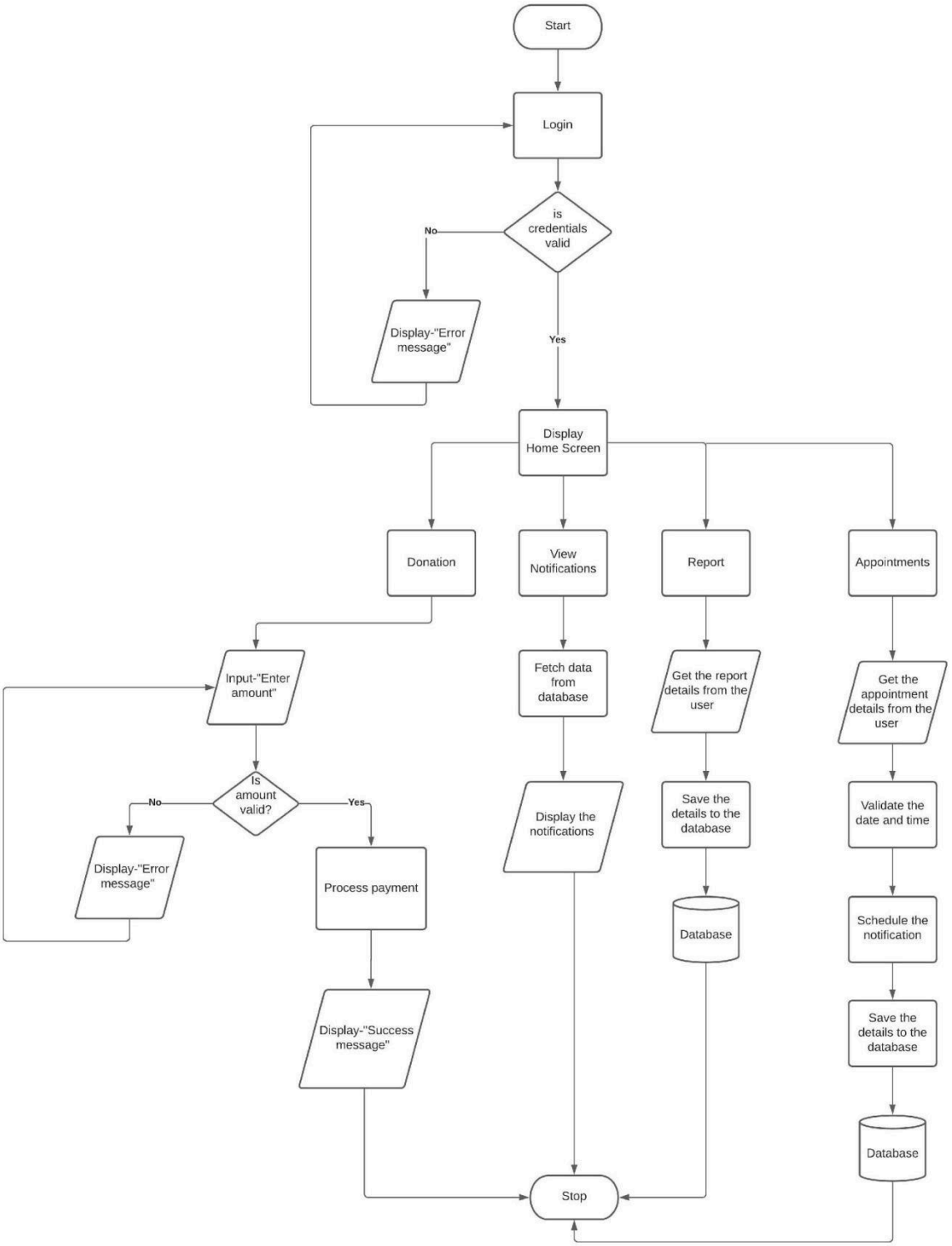


Figure 24. Mobile Client Procedural Design

ACCESSORIES

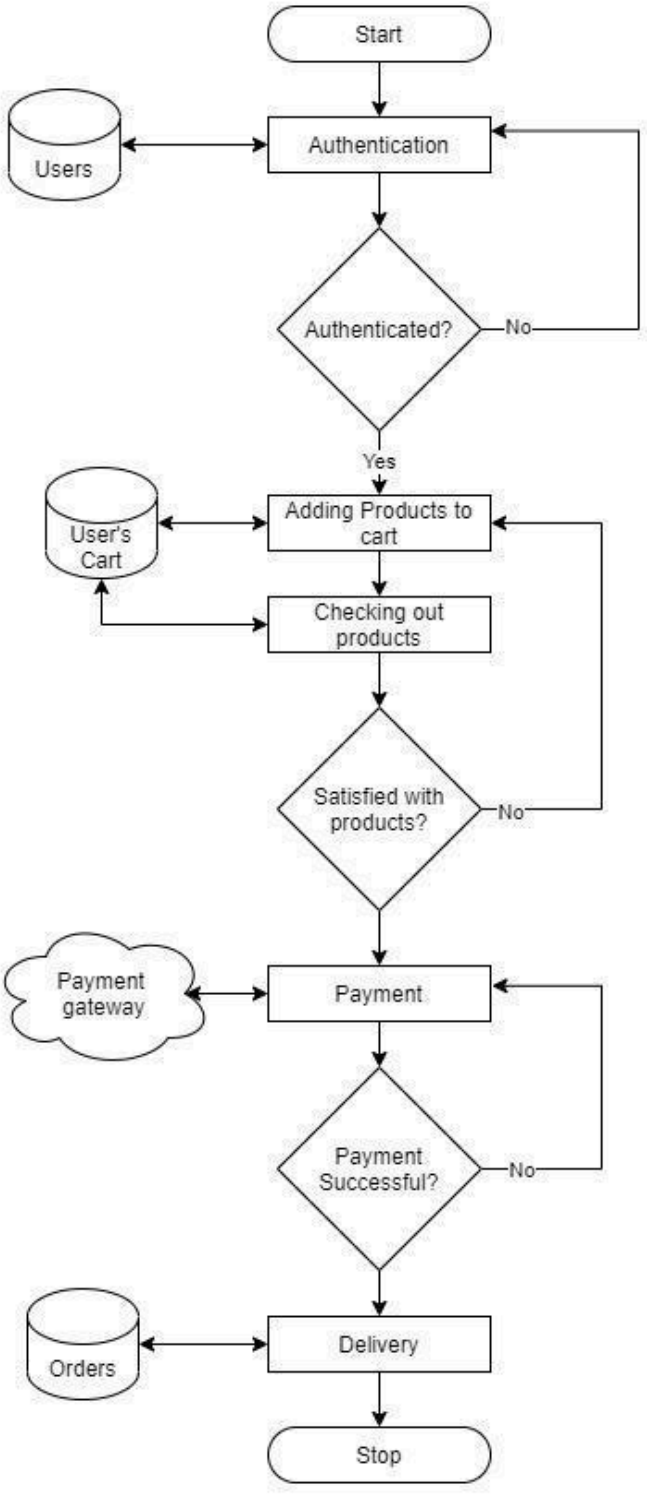


Figure 25. Pet Accessories Procedural Design

VETERINARIAN

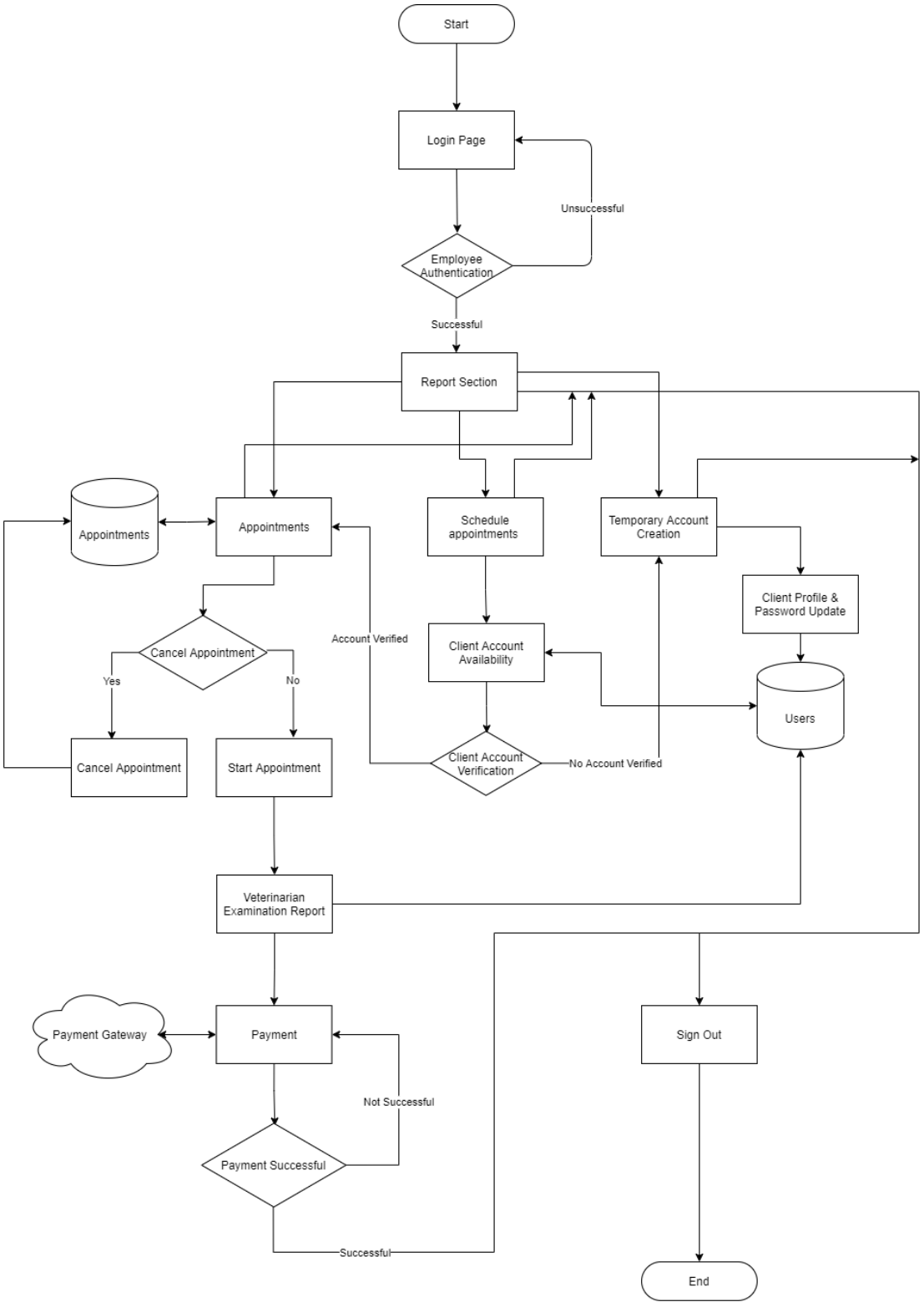


Figure 26. Veterinarian Procedural Design

GROOMER

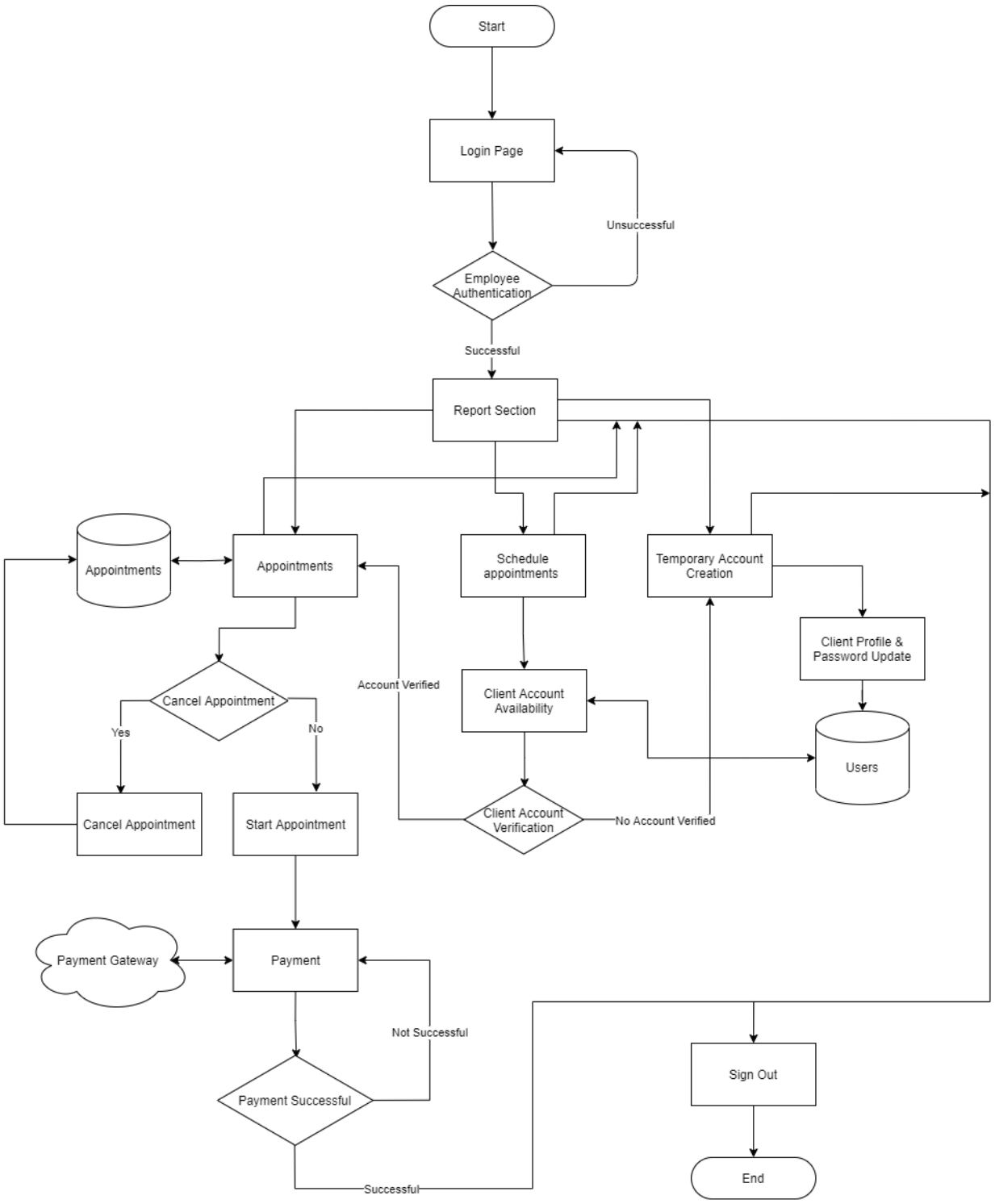


Figure 27. Groomer Procedural Design

3.7 REPORTS DESIGN

As EDEN comprises multiple independent modules, the report structure may vary as each module operates on a different set of users thus collecting different types of data. The following paragraphs describe the report structure for each module, inputs required to generate the reports and the output fields in the report.

MOBILE CLIENT

- Number of Non-Governmental Organizations situated at each location.
- Ability to list out all the reports that are associated with a particular NGO or a region.
- Ability to generate a list of all the different categories of products, each and every user purchases.
- A historical list of all confirmed appointments is kept in the remote data source. Based on these records, detailed medical reports can be generated.
- All NGO offices will be assigned a unique pin code. Based on the pin code, a list of all NGOs for a particular pin code/group of pin codes can be ascertained.

GROOMER & VETERINARIAN

The reports generated in the Groomers and Veterinarian modules will include graphical and tabular data representations to provide the employees a means to analyze the organization's performance. Certain Reports will be generated at the end of every day or month to present an overview of that month or day and thus determine whether there has been an increase or a decrease in client appointments, analysis of the reviews, etc.

- Reports may be differentiated based on weekly, monthly, and yearly reviews. Reports based on Sales Number of appointments, reviews and Groomer's sales will be available for analysis for all employee personnel of EDEN. The Groomers may identify the most preferred treatment by clients or future prospective treatments and open various features in hopes of increased revenue.

- The reports will provide the Groomers and Veterinarian modules to determine the areas of improvements and determine future outcome margins.

NGO

- The report in the NGO module will have graphical representations (charts and graphs). Reports will be generated in real-time so that the NGO has up-to-date information about the progress. The dashboard will give the NGO a quick glance at all the reports, adoptions, events and funds that have taken place. It will also show if any recent abuse has been reported.

- The fundraising will have a report about the fund raised from a particular module like adoption, events, and donations. It will have a graphical representation that can help

NGOs to understand the breakdown of total funds received and improve the strategies for raising funds for a particular module if needed.

The following are the Report Structure: -

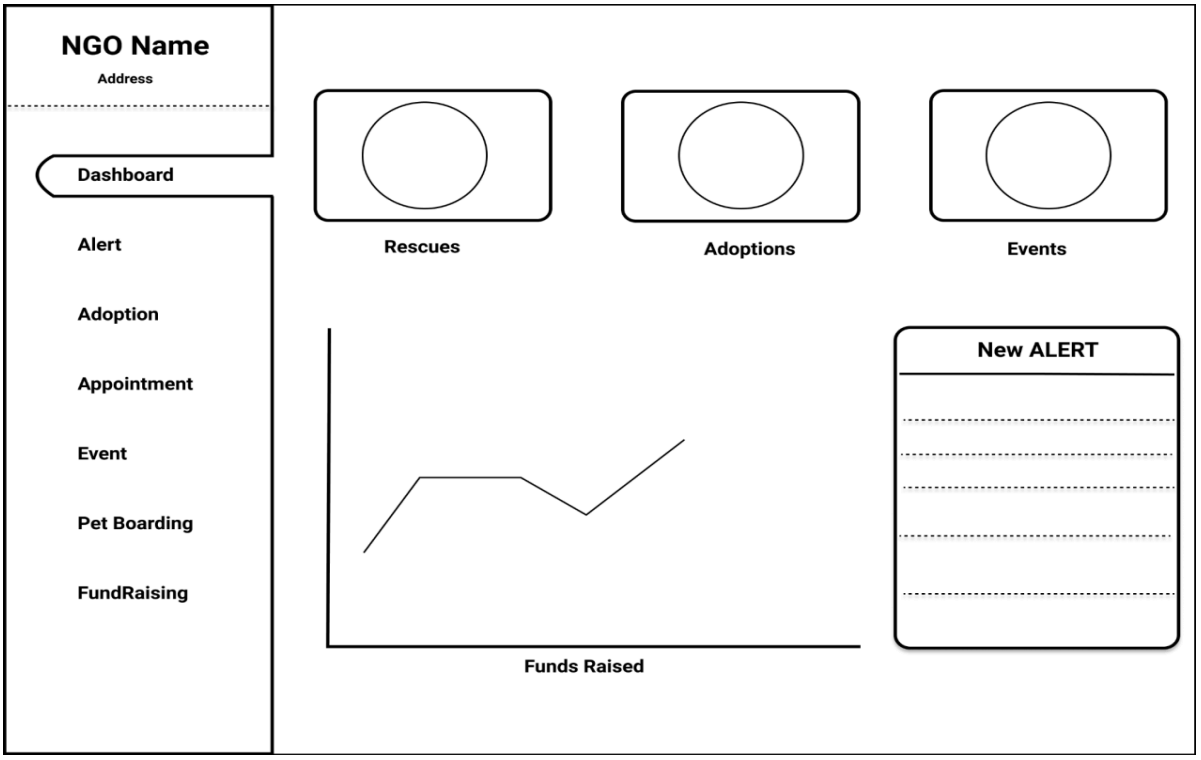


Figure 28. NGO Dashboard Report Structure

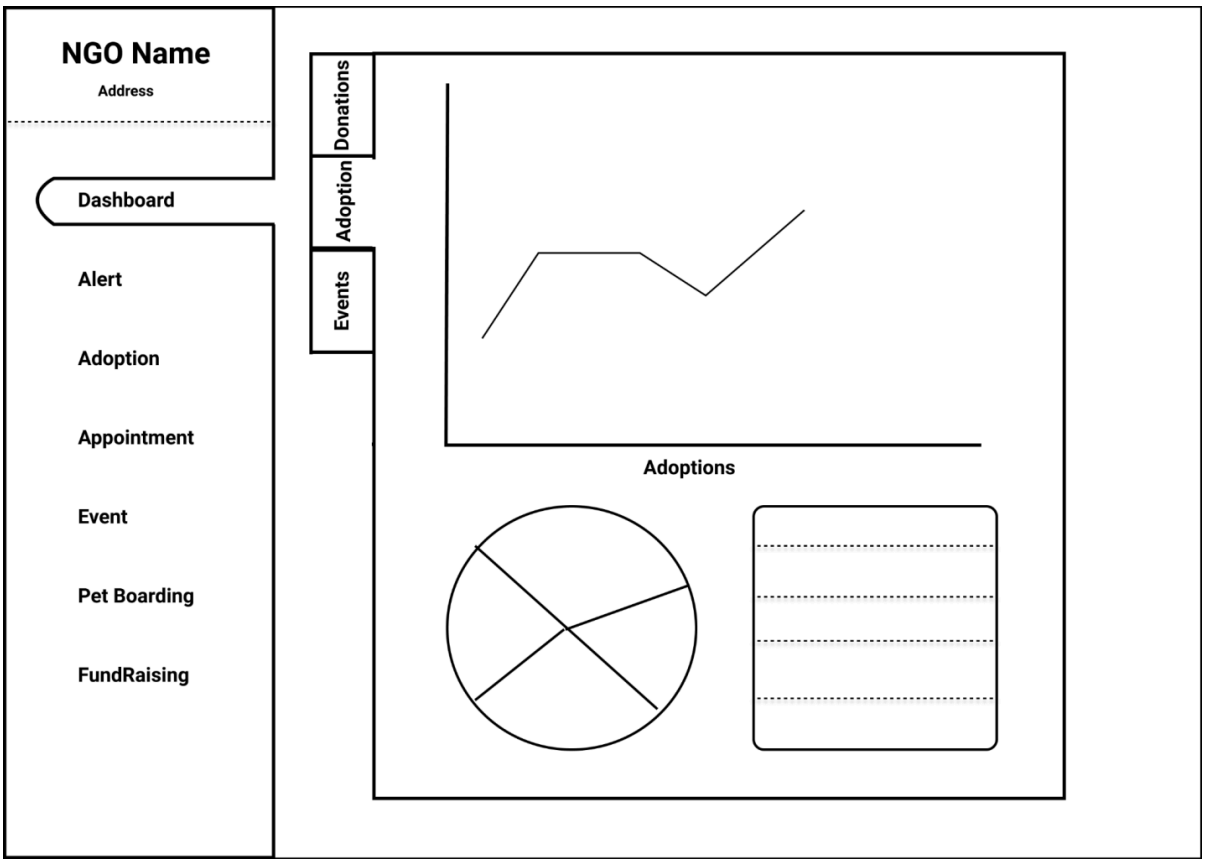


Figure 29. NGO Fundraising Report Structure

PET ACCESSORIES

- The reports generated in the pet accessories module will include tabular representations of data, some of which will be accompanied by graphical representations (charts and graphs) for quick and better understanding. Some reports will be generated in real-time to help the users (primarily sellers) better understand their progress on the website. On the other hand, some reports will be generated at the end of every day or month to present an overview of that month or day.

- An example of a real-time report would include an overview dashboard. This is meant to give sellers a quick snapshot of orders, traffic, visitor's data and sales. This overview dashboard would require inputs like order details (time, products, customer, and amount) immediately after. The output from these inputs would be a numerical and graphical representation of the total items sold, total orders, total sales, gross sales, total customers, total stock (total out of stock), total searches, recent orders and top-selling products.

- The monthly and daily reports will include sales reports, top searches, profit margins, and finances. Sales reports will take all sales of the day as input, and as output, it will present a summary of the sales by product, brand, and category displayed in a tabular format. Moving forward, top searches will take in as input all searches for the day and group the searches by product, brand and category. At the end of the month, all these

daily reports will be combined to form profit margins (profit by product, brand, and category) and finance reports (gross sales, discounts, returns, taxes, and shipping).

The following images show the report structures for the reports mentioned above;

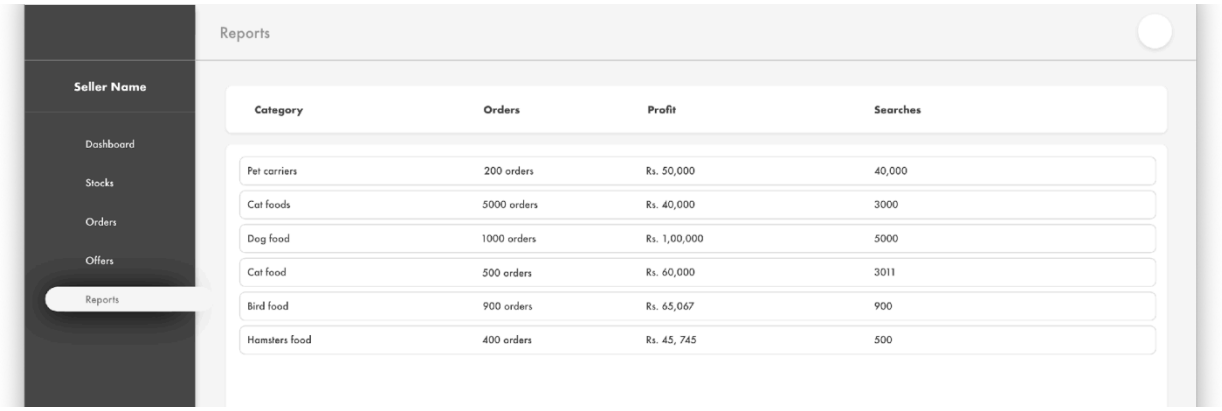


Figure 30. Pet Accessories Report Structure 1

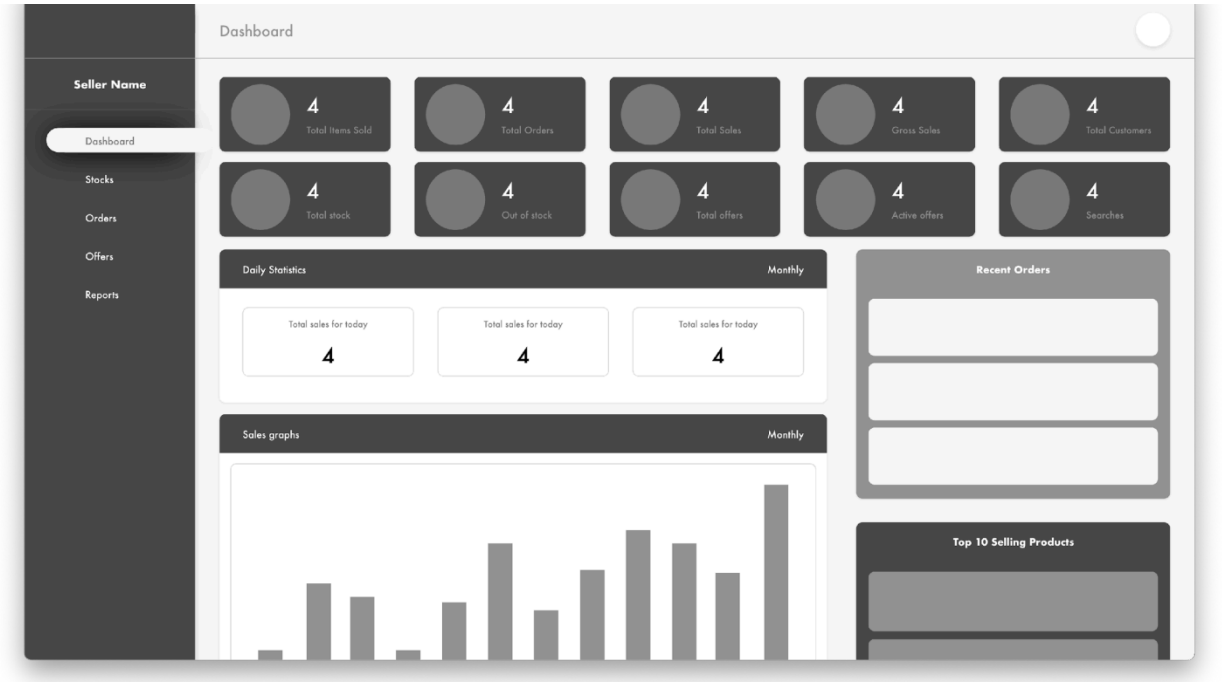


Figure 31. Pet Accessories Report Structure 2

4. IMPLEMENTATION

4.1 CODING STANDARDS

In the course of developing/coding this project, each member of the team made use of the coding standards/conventions based on the programming language used. The following points outline some of the coding standards used for the web client;

- For simplicity, the entire project and its modules were not written in one file. The entire project made use of componentization, that is, code was written in multiple files with each file holding code for one component.
- More so, files were grouped based on their type. For example, CSS files were kept in a folder separate from JavaScript files.
- Furthermore, to write clean and consistent code, we made use of a linting tool (eslint). This also made it easier to review our code.
- Moreover, for more understanding of and simplification, the code was split into multiple smaller functions, each with a single responsibility.
- In addition, comments were included wherever necessary. Comments were made meaning simple and easy to understand.
- As a good practice, all variables in JavaScript files were declared either at the beginning of the file or function definition, before being used in the code.
- Also, the use of 4 space indentation was followed to make the code clean and easy to read.
- Made use of semi columns in JavaScript files to signify the end of a statement.

The source code of the mobile client conforms fully to the Kotlin coding convention specification provided by JetBrains. It also conforms to the “Kotlin style guide” specification recommended by Google.

The following points are some of the coding standards as specified by Google and JetBrains, that have been incorporated:

- Encoded all files as UTF-8.
- The file name of files with single top-level classes matches the case and name of the class.
- Appropriate file names were chosen for files with more than one top-level class.
- Import statements for classes, functions, and properties were grouped together in a single list and were sorted by the respective ASCII values.
- The order of members within a class follows the same rules as the top-level declarations.
- Braces were not included for statements that could fit a single line.
- Followed the proper indentation guidelines (Each time a new block or block-like construct is opened, the indent increases by four spaces).
- Names of top-level or object properties which hold objects with behavior or mutable data use camel case variable naming convention.
- If a class has two properties which are conceptually the same but one is part of a public API and another is an implementation detail, an underscore is used as prefix for the name of the private property.
- Classes with a few primary constructor parameters are written in a single line.
- Classes with longer headers are formatted in such a way that each primary constructor parameter is in a separate line with indentation. Also, the closing parenthesis will be on a new line. If inheritance is used, the superclass constructor call or the list of implemented interfaces is located on the same line as the parenthesis.

4.2 CODING DETAILS

PET ACCESSORIES

- **Creating:**

```
router.post('/orders/create', async (req, res) => {
  console.log(req.body);


  var conditions;


  if(req.body.userType === "seller") conditions = firestore.collection("sellers").doc(req.body.uid)
  else conditions = firestore.collection("users").doc(req.body.uid)
  // checks whether user exist
  await conditions
  .get().then( response => {
     if (response.exists){
        console.log("creating order")
        // creating order
        razorpay.orders.create({
```

```
            amount: 100 * parseFloat(req.body.amount),
            currency: "INR",
            receipt: "order_rcptid_" + now()
          }, (error, order) => {
            if (error) throw error;
            console.log("order")
            console.log(order)
            res.json({ key_id: razorpayKeys.key_id, order: order, status: 200 });
          });
      }else res.json({"key_id": null, status: 404});
    }).catch( error => console.log(error));
});
```

- **Reading data:**

```
// fetching data
router.get('/:category/:type', async (req, res) => {
  console.log(req.params);
  var conditions;
  if(req.params.category == "all"){
     conditions = firestore.collection("products")
     .where("type", "==", req.params.type.toLowerCase());
  }else{
     conditions = firestore.collection("products")
     .where("category", "array-contains", req.params.category)
     .where("type" , "==", req.params.type.toLowerCase() );
  }
  // fetching data
  await conditions.limit(2).get()
  .then(docs => {
     var data = [];
     docs.docs.forEach(doc => {
        data.push(doc.data());
     });
      res.json({ products: data});
  }).catch( error => console.error(error));

});
```

- **Updating:**

```
// update cart item
router.put('/cart', async (req, res) => {
  console.log("updating item")
  console.log(req.body)
  await firestore.collection("users")
  .doc(req.body.userId).collection("cart")
  .where("id", "==", req.body.itemId)
  .get()
  .then(async docs => {
     await firestore.collection("users")
     .doc(req.body.userId).collection("cart")
     .doc(docs.docs[0].id)
     .update({ quantityNeeded: req.body.quantity })
     .then(async response => {
        console.log(docs.docs[0].data().name)
        res.json({ status: "quantity updated" });
     }).catch( error => console.error(error));

  }).catch( error => console.error(error));
});
```

- **Deleting:**

```
// deleting offer
router.delete('/', async (req, res) => {
  // deleting offer
```

```
  console.log(req.body.id);
  await firestore.collection("offers").where("id", "==", req.body.id).get()
  .then(response => {
     // console.log(response);
     //deleting files for the current offer item
     response.docs.forEach(async (doc) => {
        await firestore.collection("offers").doc(doc.id).delete()
        .then(response => {
           // console.log(response);
           // delete from algolia
           algoliaIndex.deleteObject(req.body.id)
           .then( response => console.log(response))
           .catch( error => console.log(error));
           res.json({"respond": "Stock successfully deleted offer"});
        }).catch(error => console.log(error));
     });
  }).catch(error => console.log(error));
});
```

- **Searching:**

```
// search
router.post('/search', (req, res) => {
  // for all categories
  if (req.body.category == "all"){
     algoliaIndex.search(req.body.searchText)
     .then( hits => {
        console.log(hits);
         res.json({ result: hits });
     }).catch( error => console.log(error));
  }else{
     algoliaIndex.search(req.body.searchText, { filters: `category:${  req.body.category.substring(0, 1).toUpperCase() +
req.body.category.slice(1) }`})
     .then( hits => {
        console.log(hits);
         res.json({ result: hits });
     }).catch( error => console.log(error));
  }
});
```

NGO

- **Adding new record of animal:**

```
router.post("/create",async (req,res)=>{
    console.log("post request received")
    //read data from client
    var data = req.body;
    console.log(data);
    await firestore.collection("mobile").doc("pet-adoption")
    .collection("availableForAdoption").add(data)
    .then((response)=>{
        //send response to the client
        res.json({status: 200, message: "Successful"});
    })
    .catch((error)=>{
        console.log(error);
    });
    //send res
});
```

- **Reading the data for a specified animal:**

```
router.get("/read/:id",async(req,res)=>{
    console.log("get request received")
    var id=req.params.id;
    //go to firebase and get data
    await firestore.collection("mobile")
    .doc("pet-adoption")
    .collection("availableForAdoption").doc(id).get()
    .then(response=>{
```

```
         console.log("data send")
       //send data as reposne to thr client
        //send response
        res.json({status: 200, message:"Successful", data: response.data()});
    }).catch( error => console.log(error));

});
```

- **Adding new Event and sending Notification to the user:**

```
router.post("/create",async (req,res)=>{
    console.log("post request received")

    //read data froms client
    var data = req.body;
    console.log(data);
    await firestore.collection("mobile").doc("Event")
    .collection("NewEvent").add(data)
    .then(async(response)=>{
        //send response to the client
        res.json({status: 200, message: "Successful"});
        //notification post
        await firestore.collection("mobile")
        .doc("notifications")
        .collection("recentlySent").add({
            content:`${data.Name} is coming up on ${data.Date} at ${data.city} in
${data.Area} organized by ${data.organizer}`,
            header:"Upcoming Event",
            type:'ngo',
            uid: 'ecf5VTQUjLVCJxTy0qygSkY9y9O2'
        }).then((response)=>{
            console.log("Notification send");
        }).catch((error)=>{
            console.log(error);
        })
    })
    .catch((error)=>{
        console.log(error);
    });
    //send res
});
```

- **Viewing adoption request:**

```
router.get("/read",async(req,res)=>{
    console.log("get request received")
    //go to firebase and get data
    await firestore.collection("mobile")
    .doc("pet-adoption")
    .collection("requestedForAdoption").orderBy("name","asc").get()
    .then(response=>{
      //  console.log(response.docs);
      var data=[];
       response.docs.forEach( (doc) => {
           // console.log(doc.data())
           var temp=doc.data()
           temp['docId'] = doc.id;
            data.push(temp)
        } );
         console.log("data send")
       //send data as response to the client
        //send response
        res.json({status: 200, message:"Successful", data: data});
    }).catch( error => console.log(error));

});
```

- **Viewing a specific alert request:**

```
router.get("/read/:id",async(req,res)=>{
    console.log("get request received")
    var id=req.params.id;
    //go to firebase and get data
```

```
    await firestore.collection("mobile")
    .doc("incident-reports")
    .collection("active-reports").doc(id).get()
    .then(response=>{
         console.log("data send")
       //send data as reposne to thr client
        //send response
        res.json({status: 200, message:"Successful", data: response.data()});
    }).catch( error => console.log(error));

});
```

- **Deleting an animal from adoption:**

```
router.delete("/remove/:id",async(req,res)=>{
    console.log("delete request received")

    //id of the doc
    var id=req.params.id;

    //delete document
    await firestore.collection("mobile").doc("pet-adoption")
    .collection("availableForAdoption").doc(id).delete()
    .then((response)=>{
        console.log("deleted doc: "+id);
        res.json({status:200,message:"Deleted"});
    })
    .catch((error)=>{
        console.log(error);
    })

});
```

GROOMERS AND VETERINARIANS

- **Code for Displaying appointments: (Similar for both modules)**

```
importReact, { useEffect } from"react";
import {useState} from"react";
import {db} from"./firebase-config";
import {collection, getDocs, updateDoc, doc} from"firebase/firestore";

import {
  Badge,
  Button,
  Card,
  Navbar,
  Nav,
  Table,
  Container,
  Row,
  Col,
} from"react-bootstrap";
import{ getDefaultFormatCodeSettings } from"typescript";

functionTableList() {

  const [users, setUsers] = useState([]);
  constusersCollectionRef = collection(db,"appointmentVet");

  constupdateUser = async (id, Status)=>{
    constuserDoc = doc(db,"appointmentVet",id);
    constnewFields = {Status:"Completed"};
    awaitupdateDoc(userDoc, newFields);
  };

  useEffect(()=>{
    constgetUsers = async()=>{
      constdata = awaitgetDocs(usersCollectionRef);
      setUsers(data.docs.map((doc)=>({...doc.data(), id:doc.id})));
    };
```

```
    getUsers();
  }, []);

  return (
    <>
      <Containerfluid>
        <Row>
          <Colmd="12">
            <CardclassName="strpied-tabled-with-hover">
              <Card.Header>
                <Card.Titleas="h4">Appointments</Card.Title>
                <pclassName="card-category">
                  All Veterinarian appointments are as follows
                </p>
              </Card.Header>
              <Card.BodyclassName="table-full-width table-responsive px-0">
                <TableclassName="table-hover table-striped">
                  <thead>
                    <tr>
                      <thclassName="border-0">Client Name</th>
                      <thclassName="border-0">Pet</th>
                      <thclassName="border-0">Breed</th>
                      <thclassName="border-0">Fee</th>
                      <thclassName="border-0">Contact</th>
                      <thclassName="border-0">Status</th>
                      <thclassName="border-0"></th>
                    </tr>
                  </thead>
                  <tbody>

                       {users.map((user)=>{
                         return(
                           <tr>
                             {" "}
                             <td>{user.clientName}</td>
                             <td>{user.petType}</td>
                             <td>{user.animalBreed}</td>
                             <td>{user.AppFee}</td>
                             <td>{user.contact}</td>
                             <td>{user.Status}</td>
                             <td><buttononClick={()=>{updateUser(user.id, user.Status);}}>Update
Status</button></td>
                           </tr>
                         );
                       })}

                  </tbody>
                </Table>
              </Card.Body>
            </Card>
          </Col>
        </Row>
      </Container>
    </>
  );
}

exportdefaultTableList;
```

- **Code for Scheduling appointment:**

```
importReactfrom"react";
importChartistGraphfrom"react-chartist";

import {useState, useEffect} from"react";
import {db} from"./firebase-config";
import {collection, getDocs, addDoc} from"firebase/firestore";

// react-bootstrap components
import {
  Badge,
  Button,
  Card,
```

```
  Navbar,
  Nav,
  Table,
  Container,
  Row,
  Col,
  Form,
  OverlayTrigger,
  Tooltip,
} from"react-bootstrap";

functionDashboard() {

  const [newClientName, setNewClientName] = useState("");
  const [newPetType, setNewPetType] = useState("");
  const [newAppFee, setNewAppFee] = useState("");
  const [newStatus, setNewstatus] = useState("");
  const [newBreed, setNewBreed] = useState("");
  const [newContact, setNewContact] = useState(0);

  //For display
  const [users, setUsers] = useState([]);
  constusersCollectionRef = collection(db,"appointmentVet");

  constcreateApp = async() => {
    awaitaddDoc(usersCollectionRef, {clientName:newClientName, petType:newPetType, AppFee:newAppFee,
Status:newStatus, animalBreed:newBreed, contact:newContact});
  };

  useEffect(()=>{
    constgetUsers = async()=>{
      constdata = awaitgetDocs(usersCollectionRef);
      setUsers(data.docs.map((doc)=>({...doc.data(), id:doc.id})));
    };

    getUsers();
  }, []);

  return (
    <>
<Table>
                    <center>
                      <tr>
                      <inputplaceholder="Client
Name"onChange={(event)=>{setNewClientName(event.target.value);}}/>
                    </tr><tr>
                      <inputplaceholder="Pet
Type"onChange={(event)=>{setNewPetType(event.target.value);}}/>
                      </tr><tr>
                      <inputplaceholder="Pet
Breed"onChange={(event)=>{setNewBreed(event.target.value);}}/>
                      </tr><tr>
                      <inputplaceholder="Fees
Received"onChange={(event)=>{setNewAppFee(event.target.value);}}/>
                      </tr><tr>
                      <inputtype =
"number"placeholder="Contact"onChange={(event)=>{setNewContact(event.target.value);}}/>
                      </tr><tr>

<inputplaceholder="Status"onChange={(event)=>{setNewstatus(event.target.value);}}/>
            </tr><tr>
                          <tr></tr>
                      <buttononClick={createApp}>Create Appointment</button>
                      </tr>
                    </center>
                  </Table>
  );
}

exportdefaultDashboard;
```

- **Account Creation Form:**

```
import React, { useState } from "react";

const Contact = () => {
 const [userData, setUserData] = useState({
  firstName: "",
  lastName: "",
  phone: "",
  email: "",
  address: "",
  message: "",
 });

 let name, value;
 const postUserData = (event) => {
  name = event.target.name;
  value = event.target.value;

  setUserData({ ...userData, [name]: value });
 };

 // connect with firebase
 const submitData = async (event) => {
  event.preventDefault();
  const { firstName, lastName, phone, email, address, message } = userData;

  if (firstName && lastName && phone && email && address && message) {
   const res = fetch(
    "https://reactfirebasewebsite-default-rtdb.firebaseio.com/userDataRecords.json",
    {
     method: "POST",
     headers: {
      "Content-Type": "application/json",
     },
     body: JSON.stringify({
      firstName,
      lastName,
      phone,
      email,
      address,
      message,
     }),
    }
   );

   if (res) {
    setUserData({
     firstName: "",
     lastName: "",
     phone: "",
     email: "",
     address: "",
     message: "",
    });
    alert("Data Stored");
   } else {
    alert("Please enter all the fields before proceeding");
   }
  } else {
   alert("Please enter all the fields before proceeding");
  }
 };

 return (
```

```
<>
  <section className="contactus-section">
    <div className="container">
      <div className="row">
        <div className="col-12 col-lg-10 mx-auto">
          <div className="row">
            <div className="contact-leftside col-12 col-lg-5">
              <h1 className="main-heading fw-bold">
                Temporary Account Creation<br /> _____
              </h1>
              <p className="main-hero-para">
                At EDEN, your data is confidential and kept secure at all times.
                To create an appointment with us, we need you to set up an EDEN Account with us.
              </p>
              <figure>
                <img
                  src="./images/hero1.jpg"
                  alt="contatUsImg"
                  className="img-fluid"
                />
              </figure>
            </div>

            {/* right side contact form  */}
            <div className="contact-rightside col-12 col-lg-7">
              <form method="POST">
                <div className="row">
                  <div className="col-12 col-lg-6 contact-input-feild">
                    <input
                      type="text"
                      name="firstName"
                      id=""
                      className="form-control"
                      placeholder="First Name"
                      value={userData.firstName}
                      onChange={postUserData}
                    />
                  </div>
                  <div className="col-12 col-lg-6 contact-input-feild">
                    <input
                      type="text"
                      name="lastName"
                      id=""
                      className="form-control"
                      placeholder="Last Name"
                      value={userData.lastName}
                      onChange={postUserData}
                    />
                  </div>
                </div>
                <div className="row">
                  <div className="col-12 col-lg-6 contact-input-feild">
                    <input
                      type="text"
                      name="phone"
                      id=""
                      className="form-control"
                      placeholder="Phone Number "
                      value={userData.phone}
                      onChange={postUserData}
                    />
                  </div>
                  <div className="col-12 col-lg-6 contact-input-feild">
                    <input
                      type="text"
                      name="email"
```

```
              id=""
              className="form-control"
              placeholder="Email ID"
              value={userData.email}
              onChange={postUserData}
             />
            </div>
           </div>
           <div className="row">
            <div className="col-12 contact-input-feild">
             <input
              type="text"
              name="address"
              id=""
              className="form-control"
              placeholder="Add Address"
              value={userData.address}
              onChange={postUserData}
             />
            </div>
           </div>

           <div className="row">
            <div className="col-12 ">
             <input
              type="text"
              name="message"
              id=""
              className="form-control"
              placeholder="Enter Pet Description"
              value={userData.message}
              onChange={postUserData}
             />
            </div>
           </div>
           <div class="form-check form-checkbox-style">
            <input
             class="form-check-input"
             type="checkbox"
             value=""
             id="flexCheckChecked"
            />
            <label
             class="form-check-label"
             className="main-hero-para">
             I agree EDEN and its associated services may contact me at the
             email address or phone number above
            </label>
           </div>

           <button
```

MOBILE CLIENT

- **Authentication**

```
/**
* A sealed class that encapsulates the status of initiating the
* authentication process.
*
* This sealed class consist of two data classes representing
* success and failure states.
* The [Success] class contains an [EdenUser] object, which
* represents the user who was successfully authenticated.
```

```
* The [Failure] class contains the [FailureType] which can
* be used to infer the type of failure.
*/
sealed class AuthenticationResult {
   data class Success(val user: EdenUser) : AuthenticationResult()
   data class Failure(val failureType: FailureType) : AuthenticationResult()

    /**
     * An enum consisting of all the different types of failures
     * related to [AuthenticationService]
     */
    enum class FailureType {
        /**
         * Indicates that an authentication failure occurred due to
         * an invalid email address.
         */
        InvalidEmail,

        /**
         * Indicates that an authentication failure occurred due to
         * an invalid password.
         */
        InvalidPassword,

        /**
         * Indicates that an authentication failure occurred as a
         * result of one or more of the credentials being invalid.
         */
        InvalidCredentials,

        /**
         * Indicates that an authentication failure occurred
         * as a result of an attempt to create an already existing
         * user with the same credentials.
         */
        UserCollision,

        /**
         * Indicates that an authentication failure occurred
         * during the creation of a new user account.
         */
        AccountCreation,

        /**
         * Indicates that an authentication failure occurred as
         * a result of an attempt made to fetch the details
         * of a non-existent user.
         */
        InvalidUser,

        /**
         * Indicates that a failure occurred due to a network
         * error.
         */
        NetworkFailure
    }
}
```

- **Signing in a user using Firebase:**

```
/**
* Used to signIn an existing user with the specified [email] and
* [password].
* @return an instance of [AuthenticationResult.Failure] if an error
* occurred, or, an instance of [AuthenticationResult.Success] if
* an account was created successfully.
*/
```

```
override suspend fun signIn(
   email: String,
   password: String
): AuthenticationResult = runCatching {
   firebaseAuth.signInWithEmailAndPassword(email, password).await()
   AuthenticationResult.Success(firebaseAuth.currentUser!!.toEdenUser())
}.getOrElse {
   AuthenticationResult.Failure(
        when (it) {
            is FirebaseAuthInvalidUserException ->
AuthenticationResult.FailureType.InvalidEmail
            is FirebaseAuthInvalidCredentialsException ->
AuthenticationResult.FailureType.InvalidPassword
            else -> AuthenticationResult.FailureType.NetworkFailure
        }
   )
}
```

- **Creating a new user account using Firebase:**

```
/**
* Used to create a new user account with the specified [username],
* [email],[password] and [profilePhotoUri].
* @return an instance of [AuthenticationResult.Failure] if an error
* occurred, or, an instance of [AuthenticationResult.Success] if
* an account was created successfully.
*/
override suspend fun createAccount(
   username: String,
   email: String,
   password: String,
   profilePhotoUri: Uri?
): AuthenticationResult = runCatching {
   val firebaseUser = firebaseAuth.createUser(username, email, password, profilePhotoUri)
   AuthenticationResult.Success(firebaseUser.toEdenUser())
}.getOrElse {
   AuthenticationResult.Failure(
        when (it) {
            is FirebaseAuthWeakPasswordException ->
AuthenticationResult.FailureType.InvalidPassword
            is FirebaseAuthInvalidCredentialsException ->
AuthenticationResult.FailureType.InvalidCredentials
            is FirebaseAuthUserCollisionException ->
AuthenticationResult.FailureType.UserCollision
            is FirebaseAuthInvalidUserException -> AuthenticationResult.FailureType.InvalidUser
            else -> AuthenticationResult.FailureType.NetworkFailure
        }
   )
}

/**
* This extension method is used for creating a [FirebaseUser] with
* [name],[email],[password] and profile picture ([profilePhotoUri]).
*
* Firebase doesn't provide a default method to create a user along with a
* display name and profile photo.In order to perform such a task we need to
* chain two methods - [FirebaseAuth.createUserWithEmailAndPassword] and
* [FirebaseAuth.updateCurrentUser].

* @throws  FirebaseAuthWeakPasswordException  thrown if the password is not strong enough
* @throws FirebaseAuthInvalidCredentialsException thrown if the email address is malformed
* @throws FirebaseAuthUserCollisionException thrown if there already exists an account with
the given email address
* @throws FirebaseAuthInvalidUserException thrown if the current user's account has been
disabled, deleted, or its credentials are no longer valid.
*/
suspend fun FirebaseAuth.createUser(
   name: String,
   email: String,
   password: String,
   profilePhotoUri: Uri?
```

```
): FirebaseUser = runCatching {
    createUserWithEmailAndPassword(email, password).await()
    //if user is created successfully, set the display name and profile picture
    val userProfileChangeRequest = UserProfileChangeRequest.Builder()
        .setDisplayName(name)
        .setPhotoUri(profilePhotoUri)
        .build()
    currentUser!!.updateProfile(userProfileChangeRequest).await()
    currentUser!!
}.getOrThrow()
```

- **Fetching all the pets available for adoption from Cloud Firestore:**

```
package com.example.eden.auth
class FirebaseRemoteDatabase : RemoteDatabase {
    private val fireStore = Firebase.firestore

    /**
     * Used to fetch all pets that are available for adoption.
     */
    override suspend fun fetchAllPetsAvailableForAdoption(): List<PetInfo> =
        fireStore.collection("mobile/pet-adoption/availableForAdoption")
            .get()
            .await()
            .documents
            .map(DocumentSnapshot::toPetInfo)
```

- **Email and password validation before creating a new user account:**

```
/**
* The method is used to check whether the [email] is valid .An email is valid
* if, and only if, it is not blank(ie. is not empty and doesn't contain whitespace characters)
* and matches the [Patterns.EMAIL_ADDRESS] regex.
*/
private fun isValidEmail(email: String) =
    email.isNotBlank() && Patterns.EMAIL_ADDRESS.matcher(email).matches()

/**
* The method is used to check whether the [password] is valid.A password is valid if, and only
if,
* It is of length 8 , contains at least one uppercase and lowercase letter and contains at
least one digit.
*/
private fun isValidPassword(
    password: String
) = password.length >= 8 && password.containsUppercase() && password.containsLowercase() &&
password.containsDigit()

override fun createNewAccount(
    name: String,
    email: String,
    password: String,
    profilePhotoUri: Uri?
) {
    if (!isValidEmail(email) || !isValidPassword(password)) _uiState.value =
        SignUpUiState.Failed(SignUpUiFailureType.INVALID_CREDENTIALS)
    else viewModelScope.launch(defaultDispatcher) {
        _uiState.value = SignUpUiState.Loading
        val authenticationResult =
            authenticationService.createAccount(name, email.trim(), password, profilePhotoUri)
        _uiState.value = when (authenticationResult) {
            is AuthenticationResult.Success -> SignUpUiState.Success
            is AuthenticationResult.Failure ->
getUiStateForFailureType(authenticationResult.failureType)
        }
    }
}
```

- **Listening for real time updates from Firebase Cloud Firestore**

```
private fun listenForRealtimeUpdates() {
    fireStore.collection(PETS_AVAILABLE_FOR_ADOPTION_COLLECTION_PATH)
        .addSnapshotListener { snapshot, exception ->
            if (exception == null) {
                val list = snapshot!!.documents.map(DocumentSnapshot::toPetInfo)
                _petsAvailableForAdoption.value = list
            }
        }
}
```

- **Send a request to adopt a pet**

```
/**
* Used when a user with [userId] wants to send a request for
* adopting a pet with [petId].
*
* Calling this function will move the pet document located
* in the [PETS_AVAILABLE_FOR_ADOPTION_COLLECTION_PATH] collection
* to the [PETS_REQUESTED_FOR_ADOPTION_COLLECTION_PATH] collection.
*/
override suspend fun sendRequestForAdoption(userId: String, petId: String) {
    runCatching {
        // get the pet document from "petsAvailableForAdoption" collection
        val petDocument = fireStore.collection(PETS_AVAILABLE_FOR_ADOPTION_COLLECTION_PATH)
            .whereEqualTo("id", petId)
            .get()
            .await()
            .documents
            .first()
        // add the document to the "requestedForAdoption" collection
        petDocument.data?.let {
            fireStore.collection(PETS_REQUESTED_FOR_ADOPTION_COLLECTION_PATH)
                .add(it)
                .await()
        }
        // delete the document from "petsAvailableForAdoption" collection
        fireStore.document("$PETS_AVAILABLE_FOR_ADOPTION_COLLECTION_PATH/${petDocument.id}")
            .delete()
            .await()
    }.getOrElse { Timber.w("${it.cause}: Failed to send adoption request.") }
}
```

- **Saving an incident report**

```
override suspend fun saveIncidentReport(
    userId: String,
    email: String,
    reportInfo: IncidentReportInfo
) {
    runCatching {
        val imageByteArray = reportInfo.image.toByteArray()
        // generate unique identifier for image name
        val uuidString = UUID.nameUUIDFromBytes(imageByteArray).toString()
        // upload image to cloud storage
        val uploadTaskSnapShot = firebaseStorage.reference
            .child("$INCIDENT_REPORT_IMAGE_STORAGE_LOCATION_PATH/$uuidString")
            .putBytes(imageByteArray)
            .await()
        // get the download URL of uploaded image
        val downloadUrlForImage = uploadTaskSnapShot.metadata?.reference?.downloadUrl?.await()
        // save to fireStore
        fireStore.collection(INCIDENT_REPORT_COLLECTION_PATH)
            .add(
                reportInfo.toIncidentReportInfoDTO(
                    userID = userId,
                    email = email,
                    imageURLString = downloadUrlForImage.toString()
                )
            )
```

```
            .await()
    }.getOrElse { Timber.w("${it.message}: Failed to save incident request.") }
}
```

- **Unit tests for firebase authentication**

```
class FirebaseAuthenticationServiceTest {
    private val firebaseAuthService = FirebaseAuthenticationService()
    private val firebaseAuth = FirebaseAuth.getInstance()

    // test user' details
    private val testUserName = "testUserName"
    private val testEmail = "testEmail@testEmail.com"
    private val testPassword = "testPassword"
    private var isSignInTestExecuted = false

    @Before
    fun setUp() {
        if (firebaseAuthService.currentUser != null) {
            // If a user is already signed in, sign the user off.
            firebaseAuthService.signOut()
        }
    }

    @Test
    fun createAccountTest_validUserDetails_isSuccessfullyCreated() {
        // given a set of valid user details
        runBlocking {
            // when creating an account with the details
            firebaseAuthService.createAccount(
                username = testUserName,
                email = testEmail,
                password = testPassword
            )
        }
        // the account must be successfully created and be assigned to the
        // currentUser variable.
        assertNotNull(firebaseAuthService.currentUser)
    }

    @Test
    fun signInTest_existingUser_isSuccessful() {
        isSignInTestExecuted = true
        // give a set of valid credentials
        val authenticationResult = runBlocking {
            // when signIn is called with the credentials
            firebaseAuthService.signIn(testEmail, testPassword)
        }
        // the result must be an instance of AuthenticationResult.Success
        assert(authenticationResult is AuthenticationResult.Success)

    }

    @After
    fun tearDown() {
        // signInTest must be the last test to be executed.
        // If it has executed, delete the user from firebase.
        if (isSignInTestExecuted) firebaseAuth.currentUser!!.delete()
        else firebaseAuth.signOut()
    }
```

4.3 SCREENSHOTS

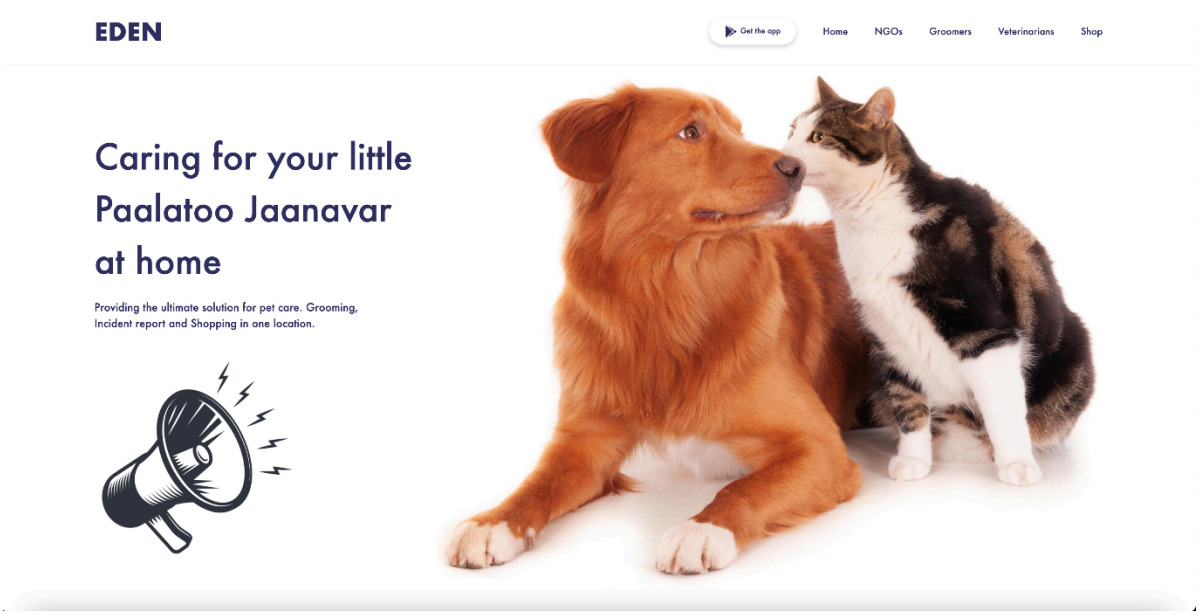


Figure 32. EDEN Home Page

NGO:

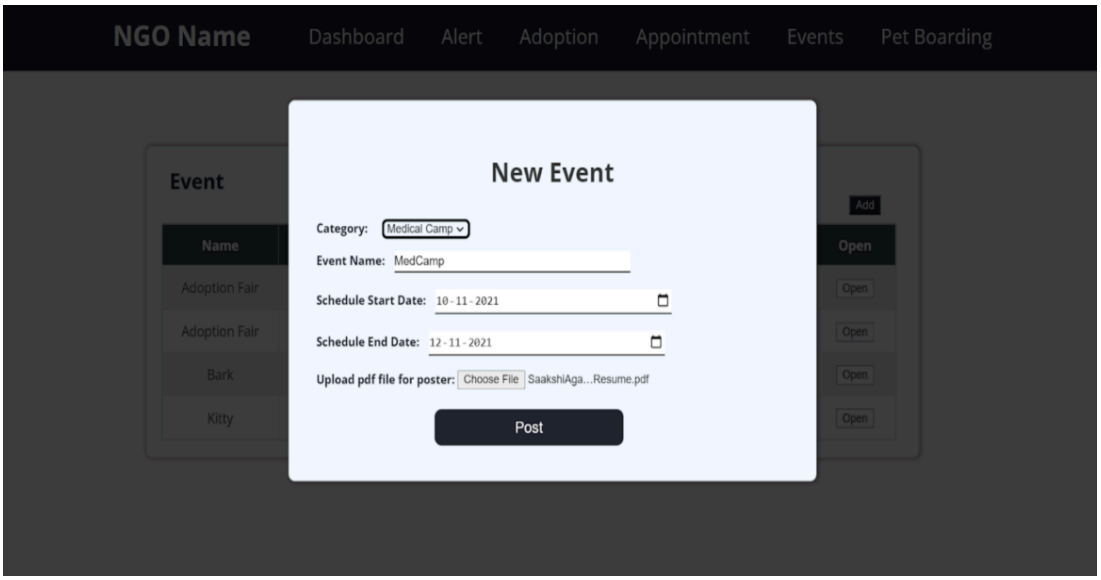


Figure 33. NGO: Adding new Event

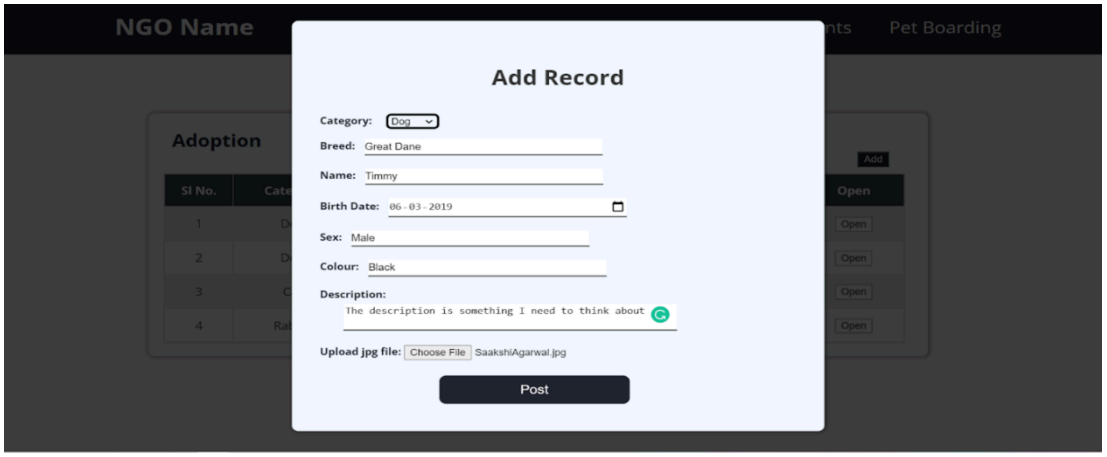


Figure 34. NGO: Adding new animal up for adoption

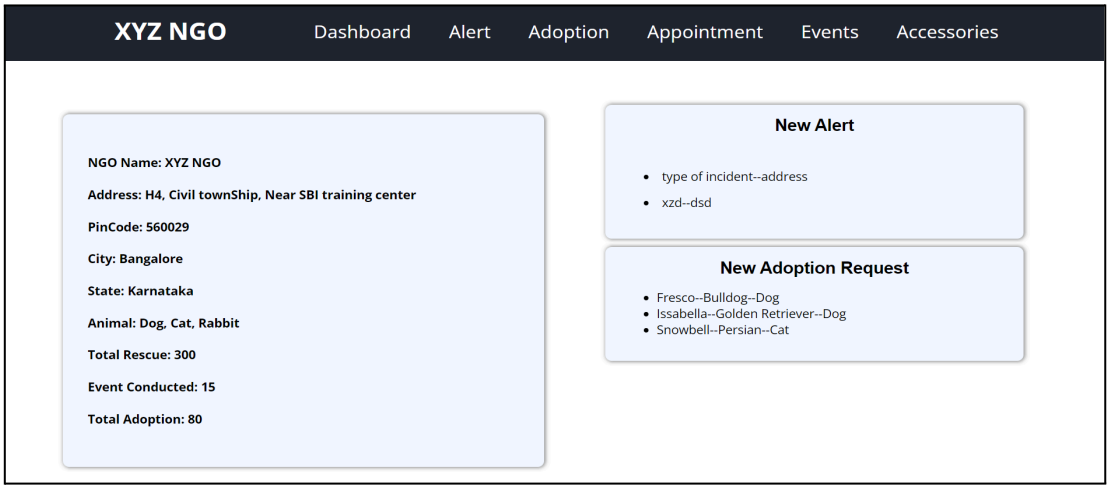


Figure 35. NGO: Dashboard

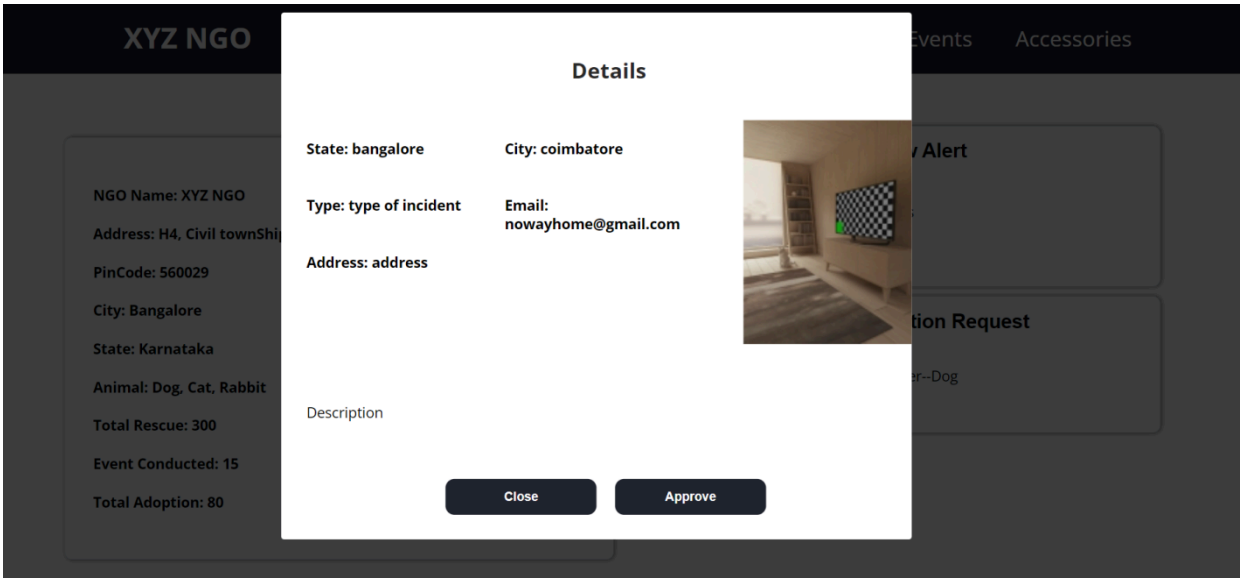


Figure 36. NGO: Viewing alert request made by user

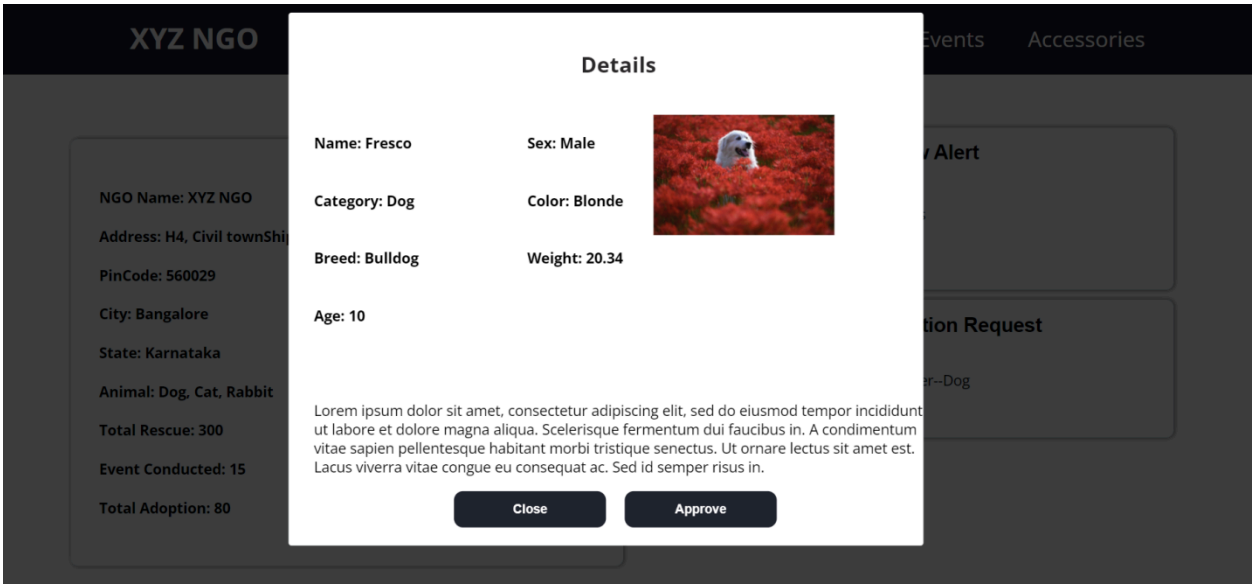


Figure 37. NGO: Viewing request for adoption

ACCESSORIES:

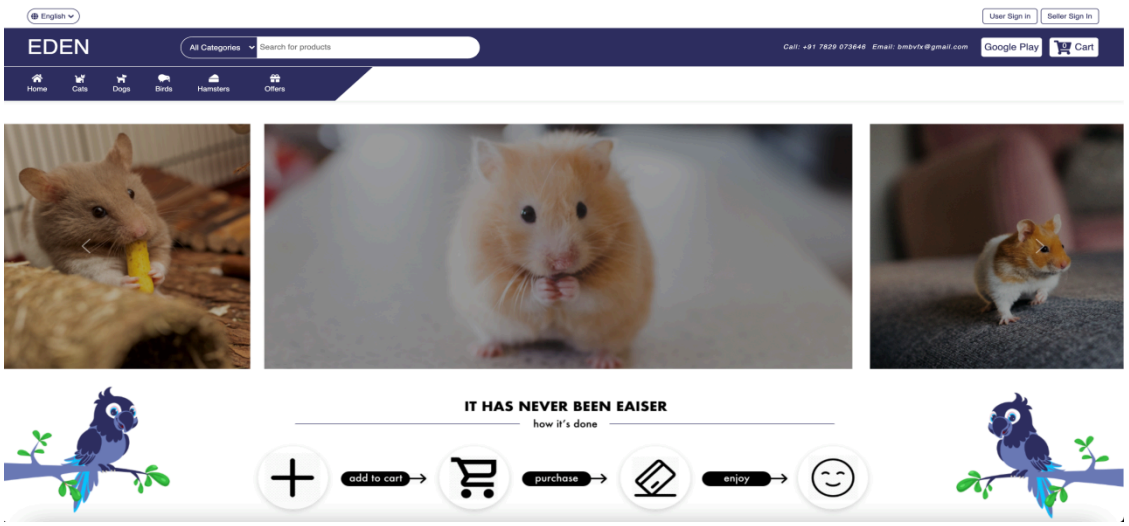


Figure 38. – Pet Accessories: Home Page

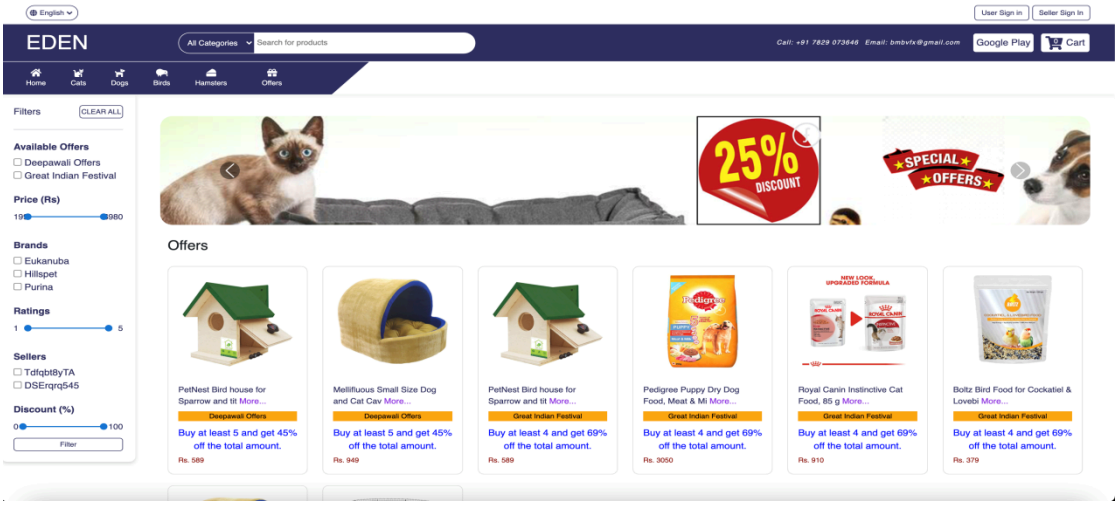


Figure 39. – Pet Accessories: Category and Offer Page

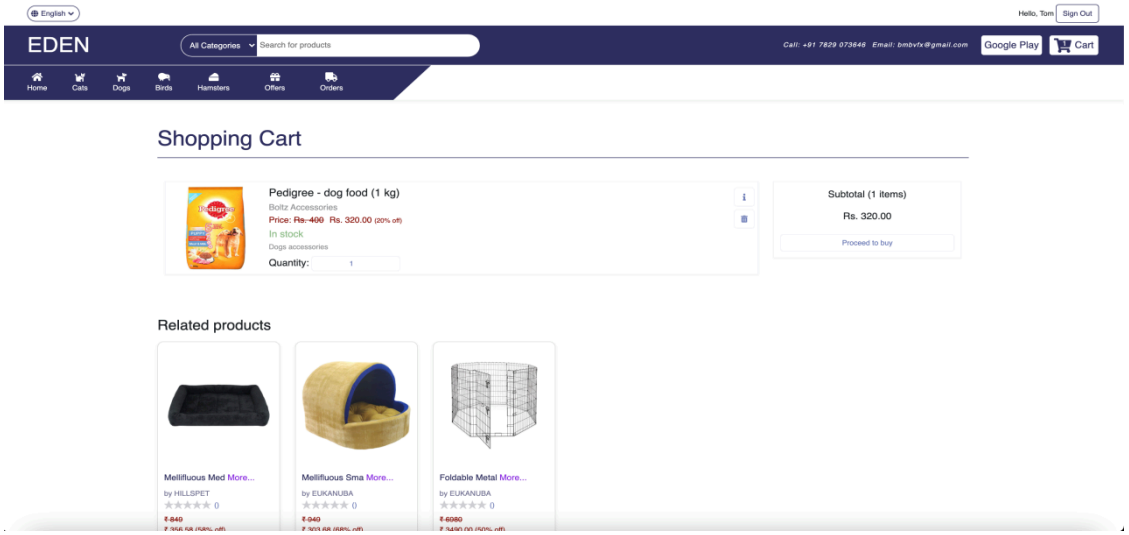


Figure 40. – Pet Accessories: Shopping Cart Page

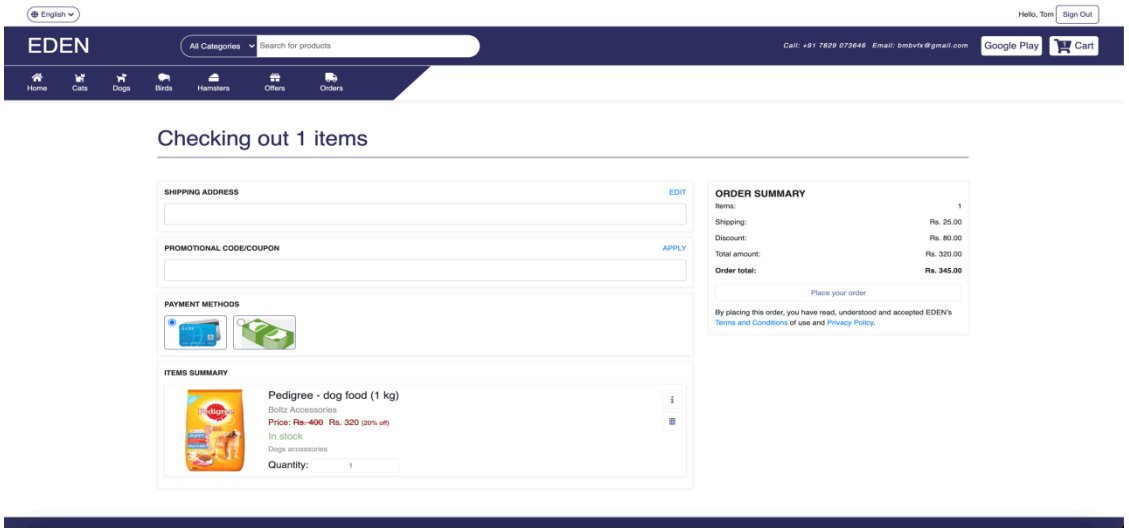


Figure 41. – Pet Accessories: Checkout Page

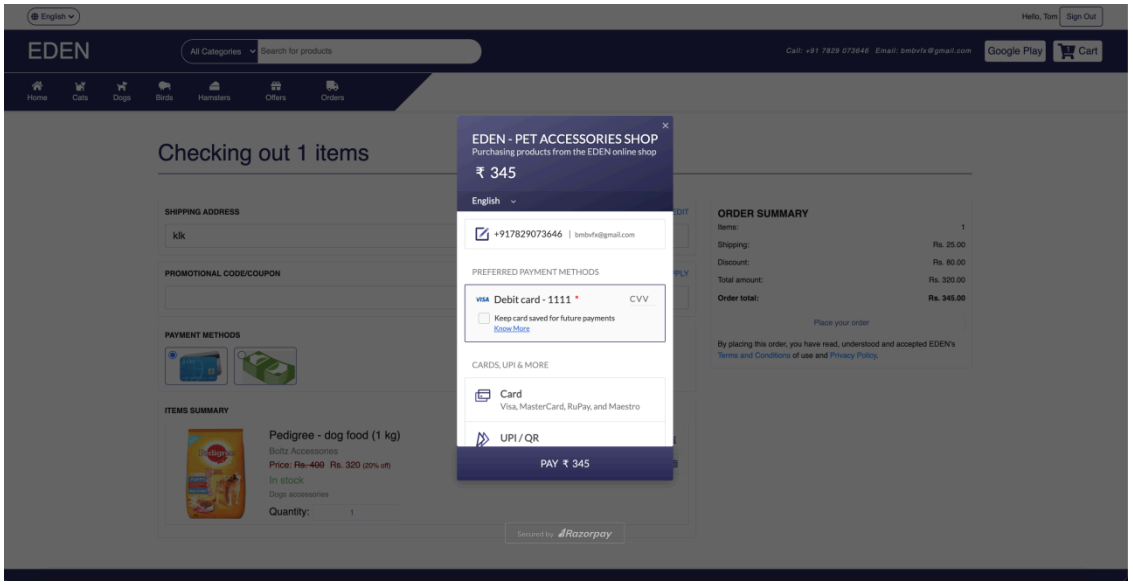


Figure 42. – Pet Accessories: Payment

GROOMERS AND VETERINARIANS:

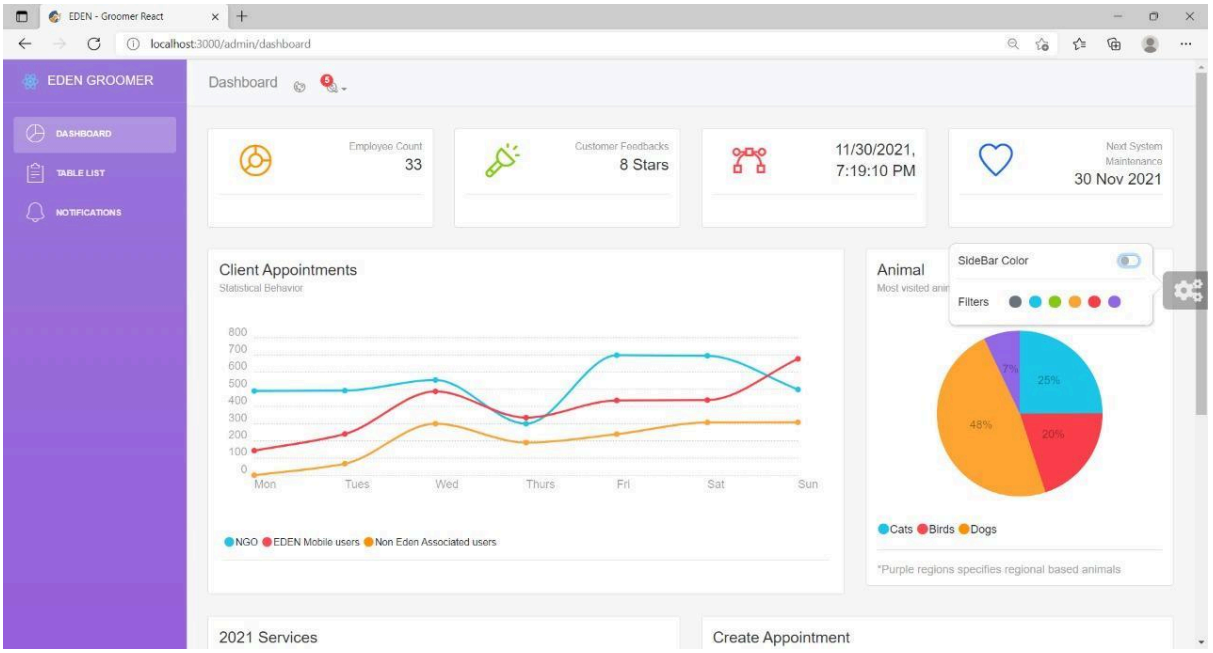


Figure 43. – Groomers: Dashboard

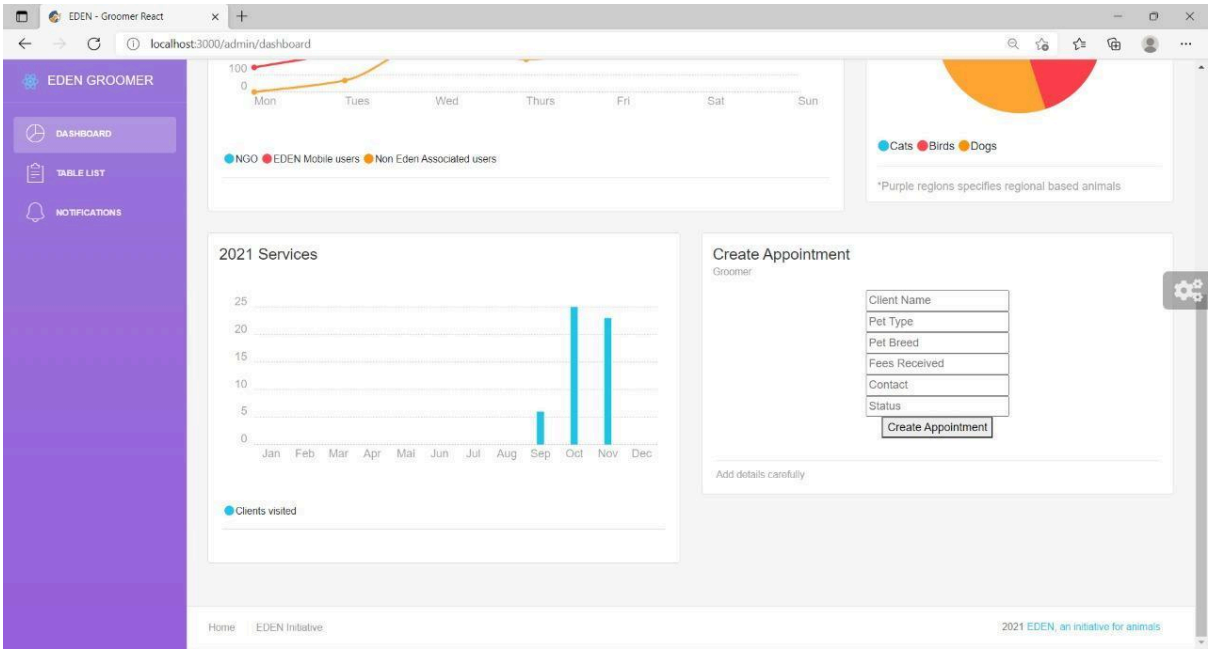


Figure 44. – Groomers: Appointment Entry

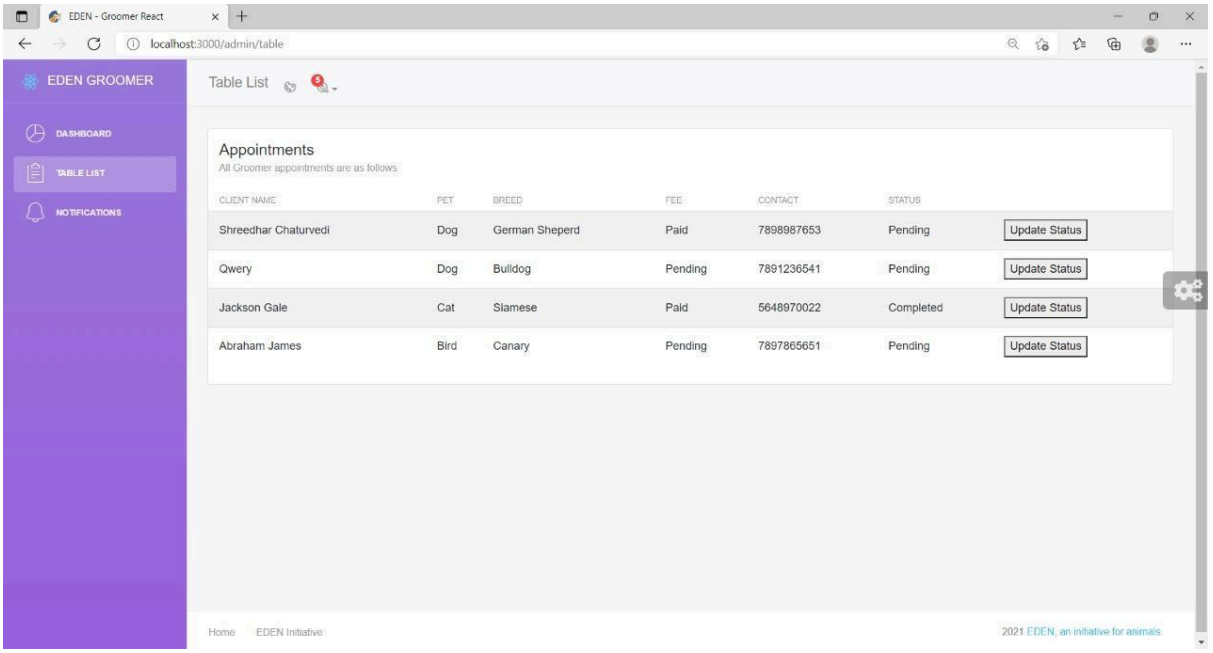


Figure 45. – Groomers: Appointments List

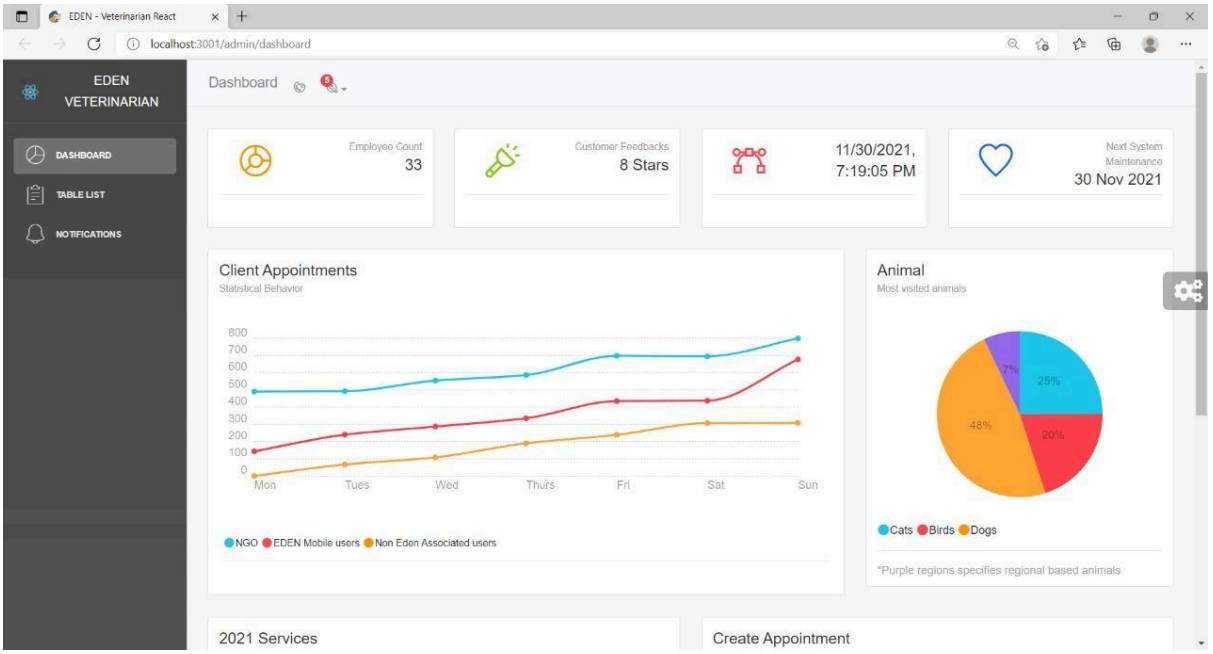


Figure 46. – Veterinarian: Dashboard

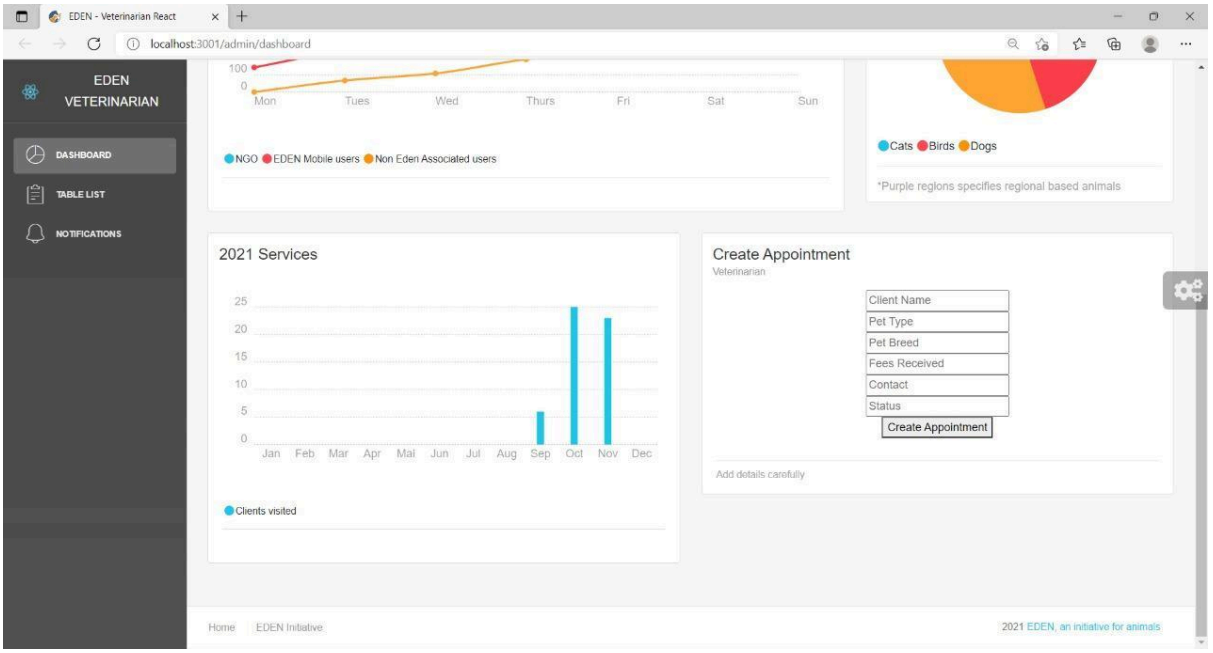


Figure 47. – Veterinarian: Appointment Entry

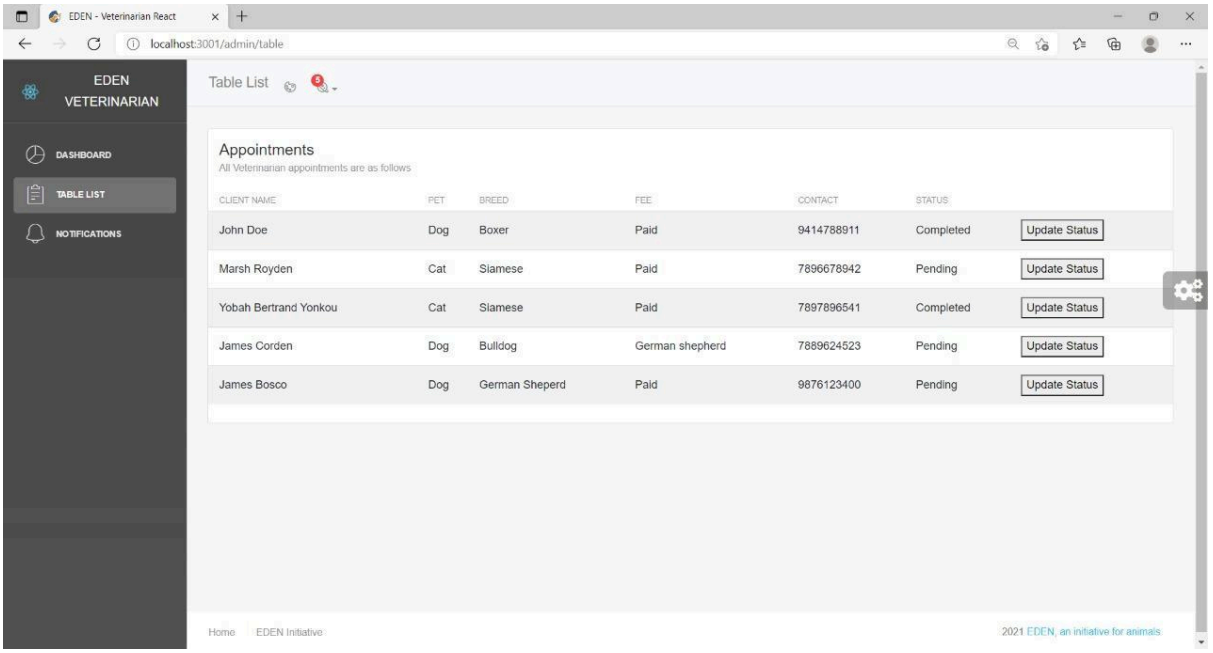


Figure 48. – Veterinarian: Appointments List

MOBILE CLIENT:

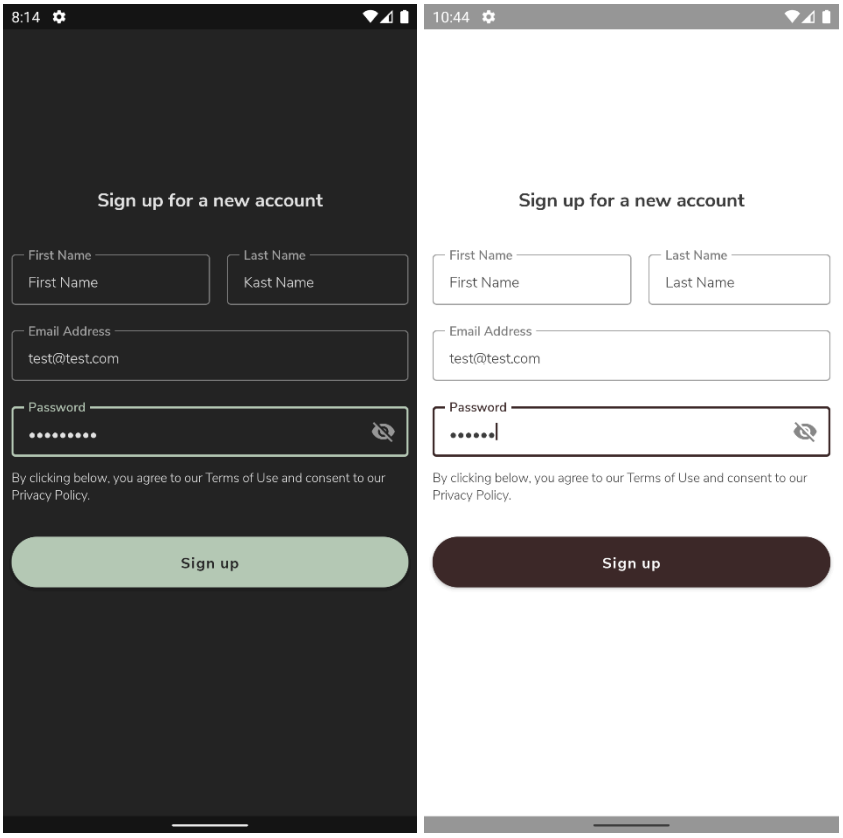


Figure 49. - Mobile Client: Sign Up (Dark and Light mode)

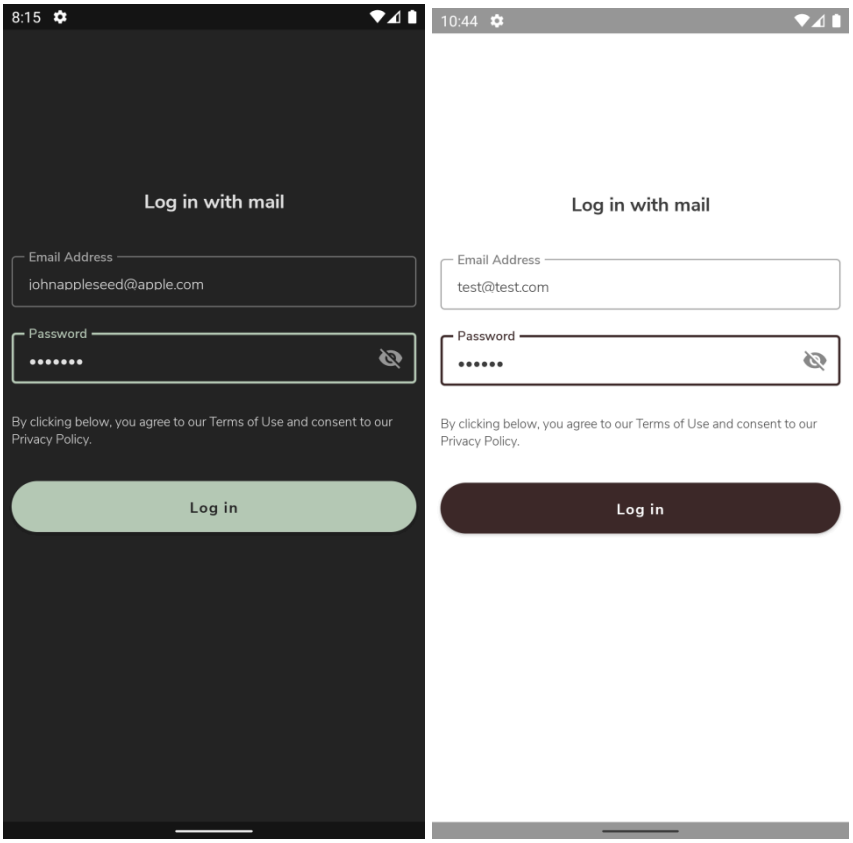


Figure 50. – Mobile Client: Login (Dark and Light mode)

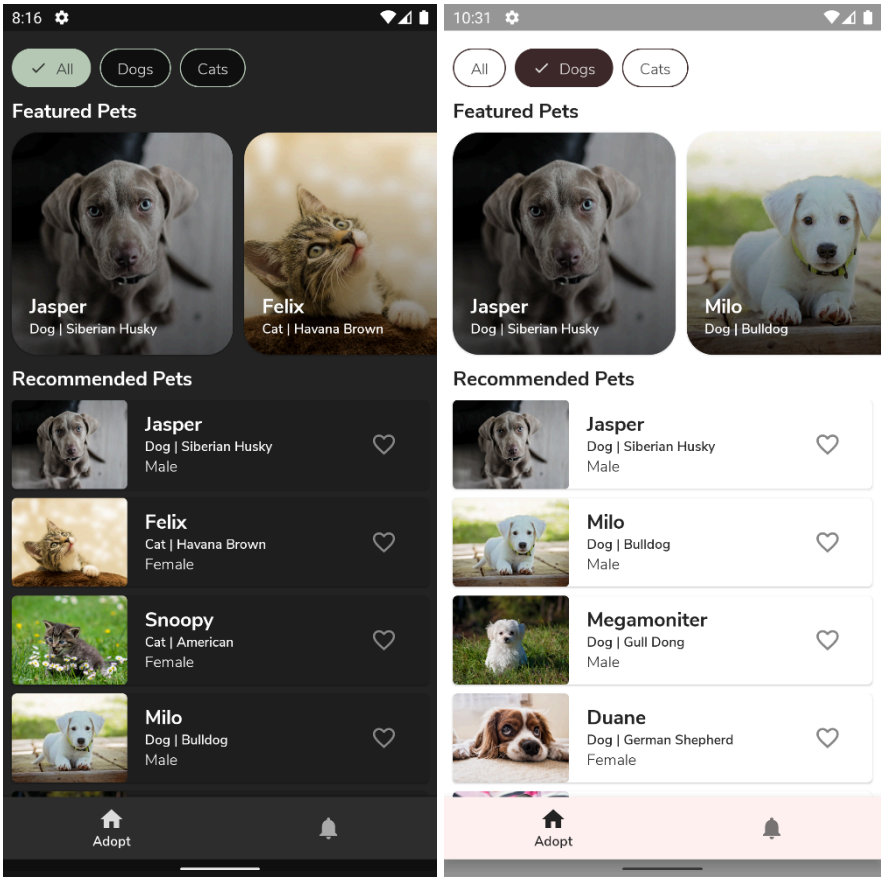


Figure 51. - Mobile Client: Pet Adoption (Dark and Light mode)

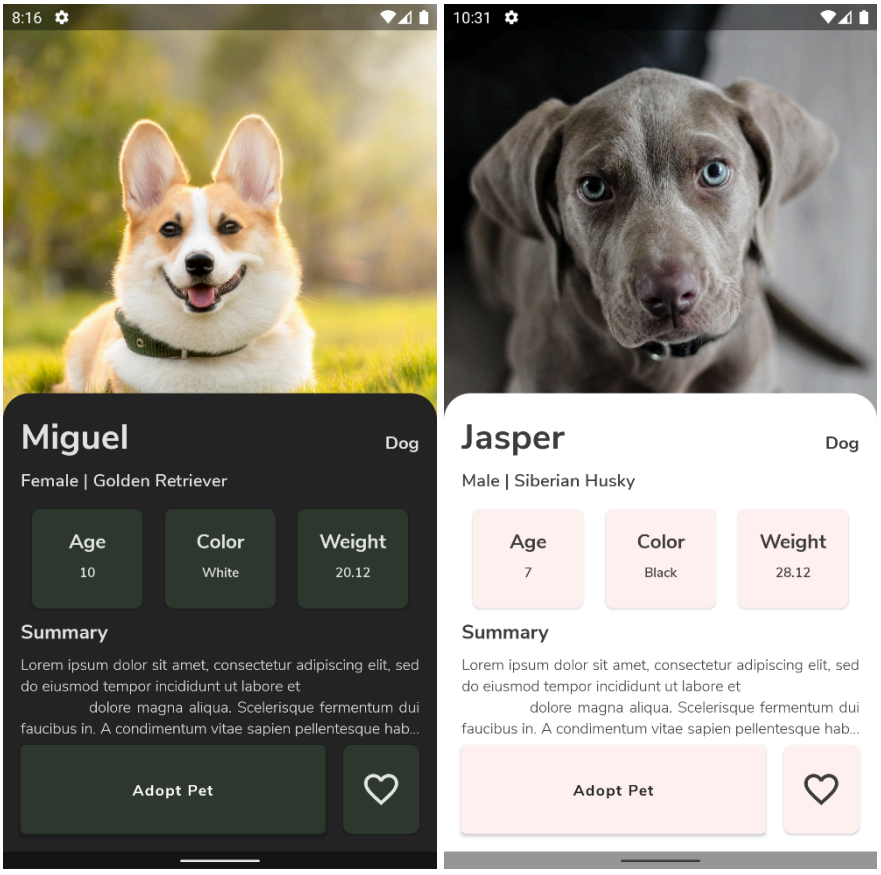


Figure 52. - Mobile Client: Pet Details (Dark and Light mode)

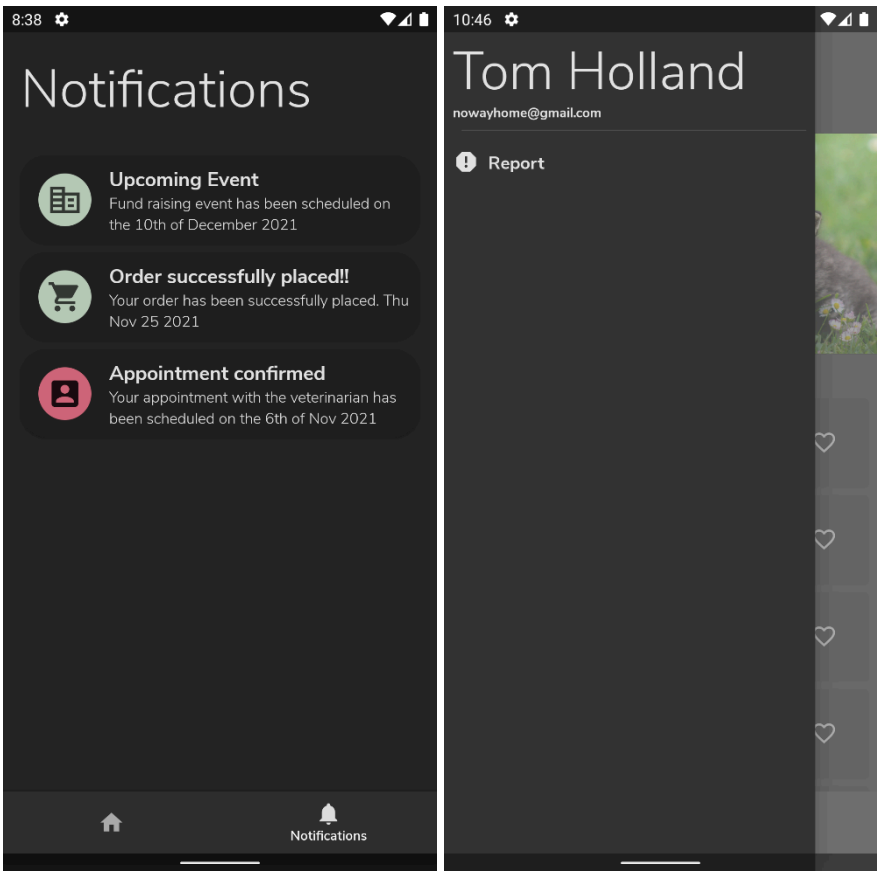


Figure 53. - Mobile Client: Notifications and Navigation Drawer (Dark mode)

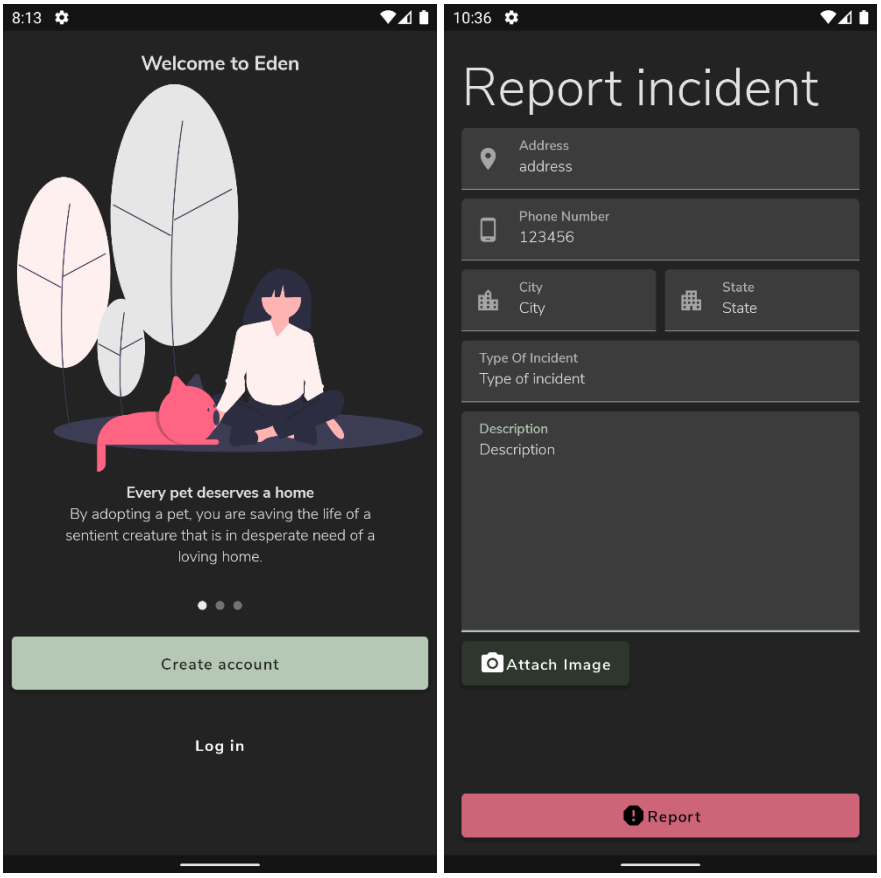


Figure 54. - Mobile Client: On boarding and Report Incident (Dark mode)

5. TESTING

EDEN has a number of modules which are required to complete aim of the project namely, the NGO, Pet Accessories, Veterinarian and Groomer modules. Each of the modules have been developed while looking into possible test cases and bugs which might occur in the execution of the applications.

5.1 TESTING APPROACHES

This project is a combination of android app and website used by users, NGO, Groomers, Sellers and Veterinarians to manage different activities. The following testing strategies have been applied on the processes on the website: -

Unit Testing: This testing strategy focuses on the functioning of individual processes in the system. That is, it evaluates how the individual processes work in the system.
Each process is fed a number of test cases and checks are made for how the process is interacting with the backend and whether the data supplied by the test case is appropriate.

Integration Testing: It is the phase in which individual modules are combined and tested as a group. It is conducted to evaluate the compliance of components with specified functional requirements.

5.2 TEST CASES

PET ACCESSORIES

Table 24 – Test Cases - Pet Accessories

S. No	Module Name	Test Case No	Test Case Description	Expected Result
1	ACCESSORIES MODULE - Searching	TC1	Search based on brand, category, name and description.	Products where the search term is found in their name or category or brand or description.
2	ACCESSORIES MODULE - Product page	TC2	Verify whether product details and related products are displayed correctly.	Display correct product details like description, price, brand, name, related products and offers (if any).
3	ACCESSORIES MODULE - Add to cart	TC3	Verify whether products can be added to cart from product page, and product cards.	To successfully add products to cart from product page and product card.
4	ACCESSORIES MODULE - Increase product quantity	TC4	Verify whether product quantity can be changed from product page, product card and shopping cart page.	To successfully change product quantity in the product page, product card and shopping cart.
5	ACCESSORIES MODULE - Cart page (checkout button)	TC5	Verify that the checkout button is enabled only when the user has items in the shopping cart.	Checkout button will be enabled only when the shopping cart has at least one item.
6	ACCESSORIES MODULE - Checkout	TC6	Verify that all payment methods are enabled and working. Also verify that checkout should be possible only when the shipping address is available.	Working payment options and checkout only when shipping address is provided.
7	ACCESSORIES MODULE - Previous orders	TC7	Verify that the orders page is displayed only when a user is signed in.	To display previous orders option when a user is signed in.

MOBILE CLIENT

Table 25 – Test Cases - Mobile Client

S. No	Module Name	Test Case No	Test Case Description	Expected Result
1	Mobile Client - New account credentials validation	TC8	An invalid pair of user credentials shouldn't allow a new user to create an account.	An error message must be displayed when the user enters an invalid pair of credentials.
2	Mobile Client - New account user collision	TC9	Two user accounts cannot have the same email-id.	An error message must be displayed If the email of the newly created account matches with an already existing account in the database.
3	Mobile Client- Existing account credentials validation	TC10	A user with invalid credentials must not be able to log into a valid user account.	An error message must be displayed if the user's credentials are invalid.
4	Mobile Client - User specific notifications	TC11	Verify that only those notifications that are relevant to a particular user are being displayed. Together with user specific notifications, the user must also be able to receive notifications that are relevant to all users of the platform.	Notifications relevant to all users must be delivered to all users. User-specific notifications must be delivered only to those respective users.
5	Mobile Client - Favorite pets filtering	TC12	Verify that only those pets that are liked by the user, are listed in the favorites screen.	Favorites screen must be filtered in such a way, that it displays only those pets that are liked by the user.

NGO

Table 26 – Test Cases - NGO

S. No	Module Name	Test Case No	Test Case Description	Expected Result
1	NGO-Enter new event data	TC13	All the fields should be filled and scheduled start date should be less than the schedule end date	A new record for event gets inserted if all the inputs are valid.
2	NGO-Add new animal up for adoption	TC14	All the fields should be filled with appropriate data.	A new record for event gets inserted if all the inputs are valid.
3	NGO-Accept/Decline Alert request	TC15	It can accept/reject pet boarding request of user after going through the data	If it is accepted/rejected the backend will get updated accordingly and will remove it from the table
4	NGO-Request for an appointment for vet/groomer	TC16	All the fields should be filled with appropriate date	A new record for event gets inserted if all the inputs are valid.

GROOMERS AND VETERINARIANS

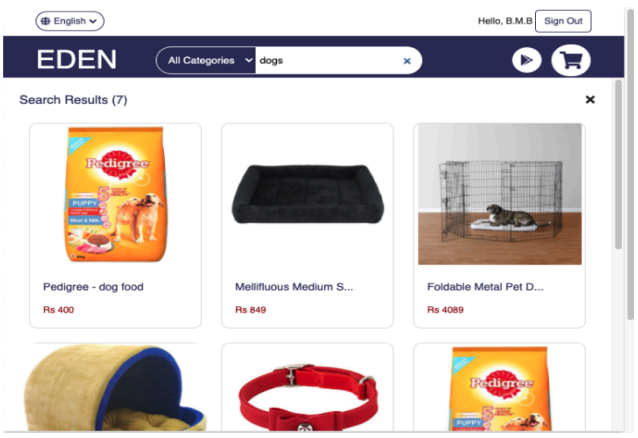
Table 27 – Test Cases - Groomer and Veterinarian

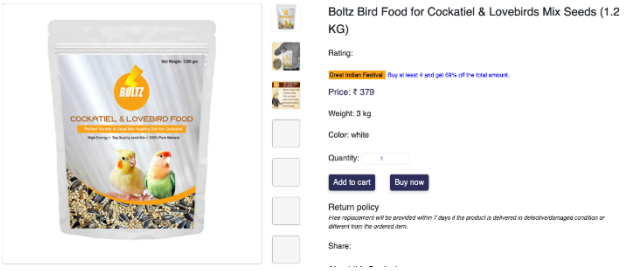
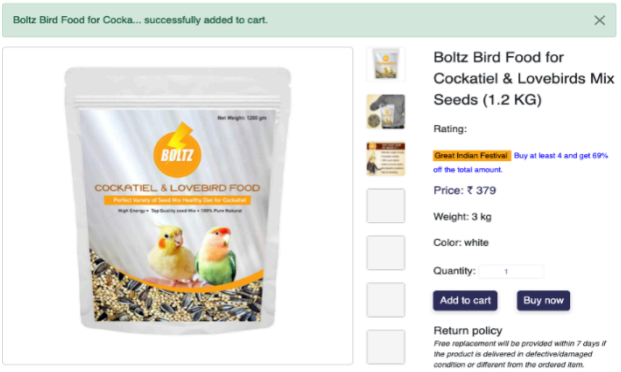
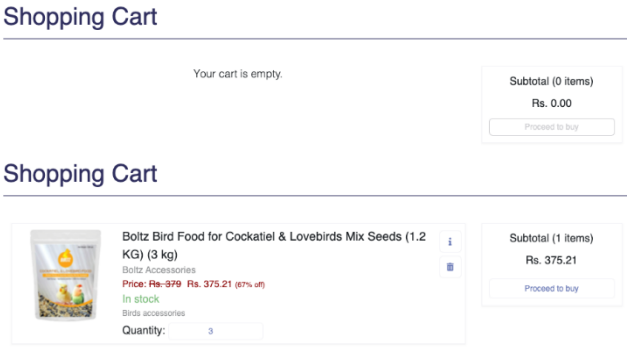
S. No	Module Name	Test Case No	Test Case Description	Expected Result
1	Vet_Groomer-Enter new appointment	TC18	All the fields should be filled and scheduled start date should be less than the schedule end date	A new record for event gets inserted if all the inputs are valid.
2	Vet_Groomer-Accessing records	TC19	Employees can access pet records if they are assigned the appointment	If valid employee, then record is fetched, else denied access
3	Vet_Groomer-Open Schedule	TC20	Only time slots available can be selected for appointments	If time slot is open, then appointment is made, else no appointment is scheduled

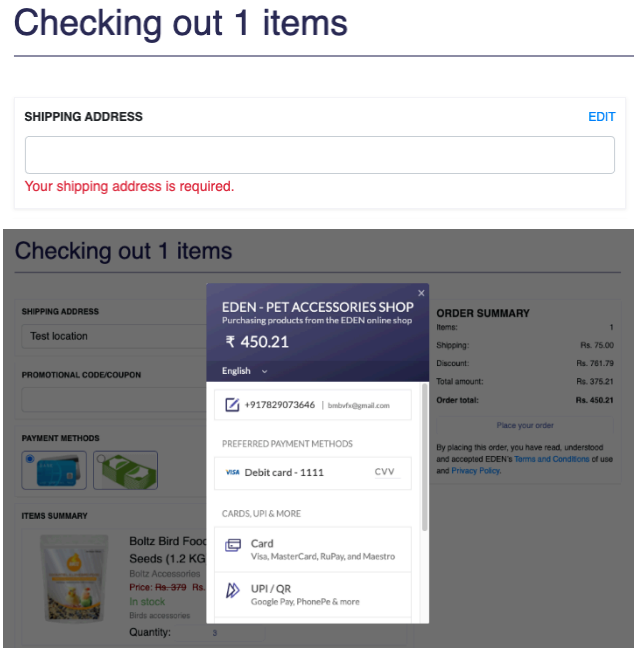
5.3 TEST REPORTS

PET ACCESSORIES

Table 28 – Test Reports - Accessories

S. No	Test	Test Status	Test Report
1	TC1	Successful	

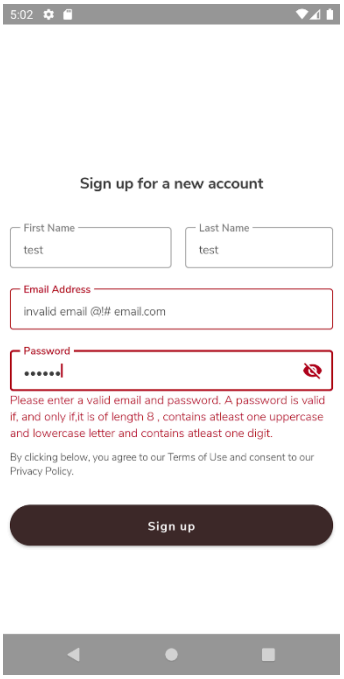
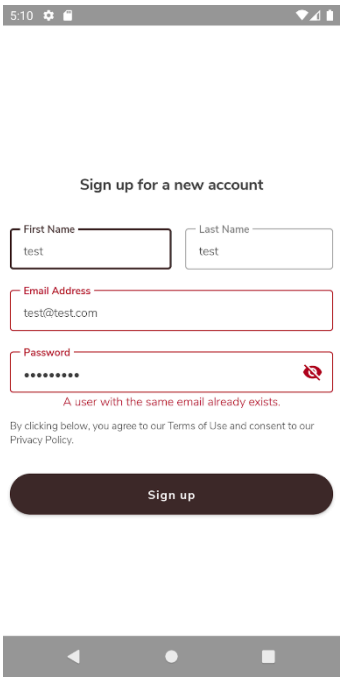
2	TC2	Successful	
3	TC3	Successful	
4	TC4	Successful	
5	TC5	Successful	

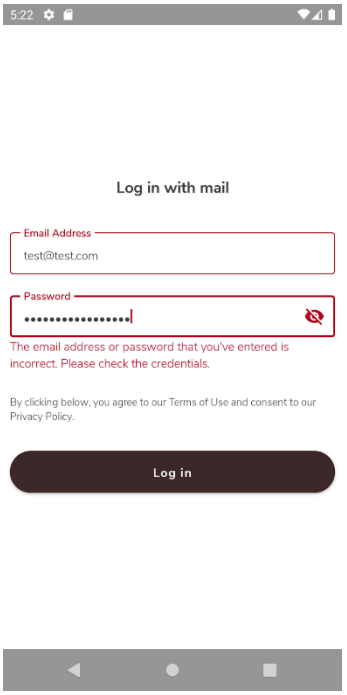
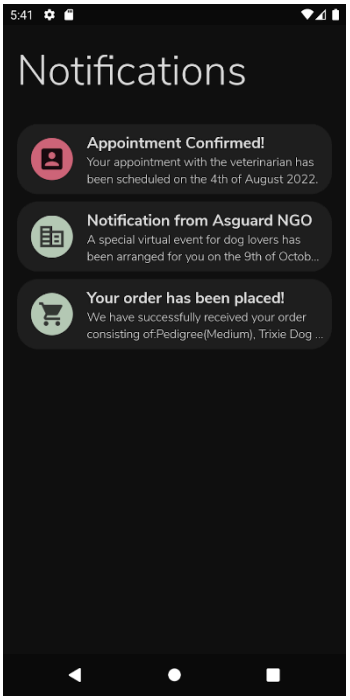
6	TC6	Successful	
7	TC7	Successful	

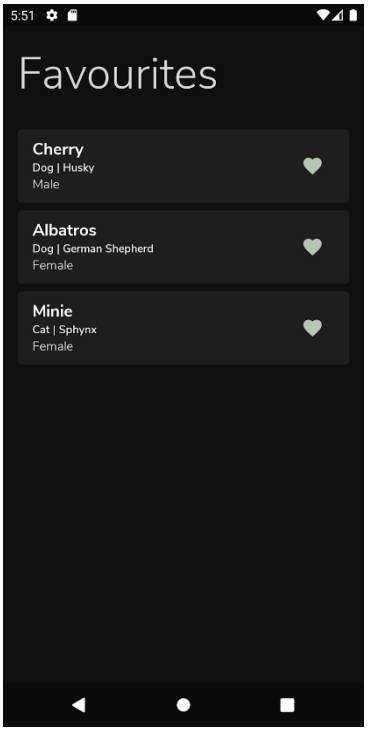
MOBILE CLIENT

Table 29 – Test Reports – Mobile Client

S. No	Test	Test Status	Test Report

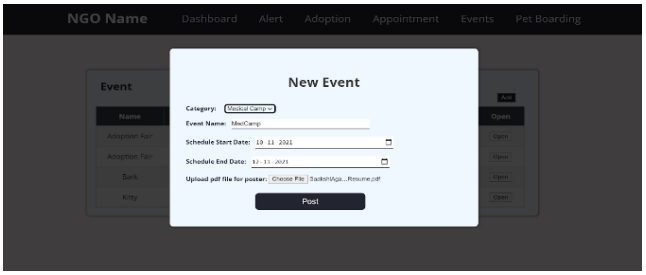
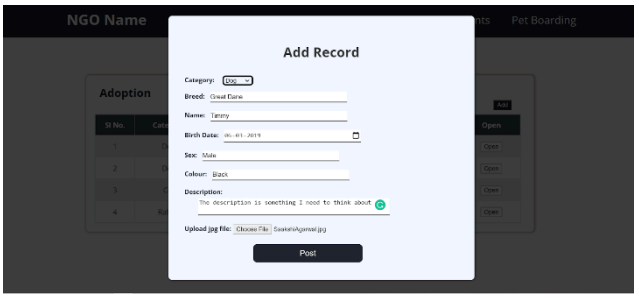
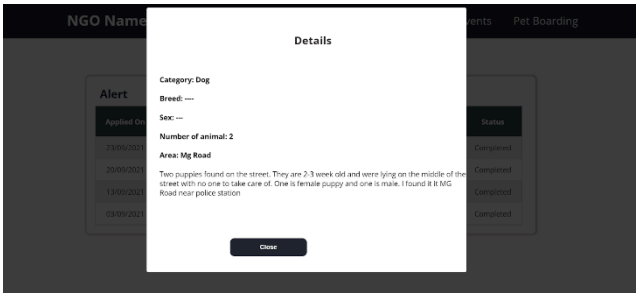
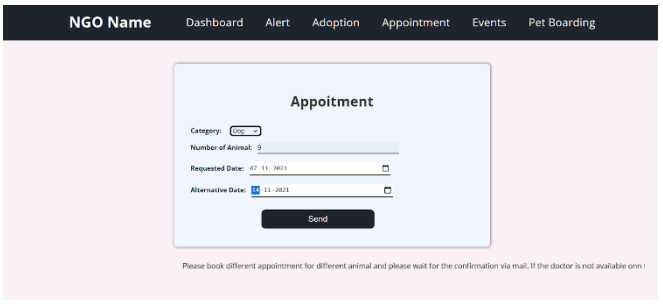
1	TC8	Successful	 **(Light mode)**
2	TC9	Successful	 **(Light mode)**

3	TC10	Successful	 **(Light mode)**
4	TC11	Successful	 **(Dark mode)**

5	TC12	Successful	 **(Dark mode)**

NGO

Table 30 – Test Reports - NGO test

S. No	Test	Test Status	Test Report
1	TC13	Successful	
2	TC14	Successful	
3	TC15	Successful	
4	TC16	Successful	

6. CONCLUSION

The EDEN Project has been developed over the span of 4 months, however certain technical issues or still exist in the system which can be solved if provided time.

6.1 DESIGN AND IMPLEMENTATION ISSUES

6.1.1 DESIGN ISSUES

The EDEN Project aims to provide its users and its target employees with the required facilities and operations to help streamline the daily operations of the NGO, Veterinarian, Groomer, Accessories and Mobile Client. However, with efforts being made into developing the websites and mobile client, limitations and development issues have and will rise from time to time during the life of this project.

With focus on the designs or the look and appearance of the websites and mobile client, chances are that, with the passing of time, newer technologies and styling techniques will be introduced which will cast a shadow on the project. This will inevitably require attention in the due course of time and efforts will need to be taken to refresh the project with respect to the current design standards.

If this project is deployed and is used in the industry, performance and speed might be areas where problems may occur. This will mainly be due to the large number of users required to access data and along with the daily database queries to be performed i.e., Create, Read, Update and Delete. However, Firestore does not create a slow query, but transferring the data to the user's device from Firestore can produce delays. Addressing this issue and allocating

enough resources to the project preferably assigning Amazon Web Services to fulfil the requirements of the EDEN project. Maintaining EC2 instances with preferred workloads will cater to periodic surges in traffic.

A major concern regarding the project is the lack of possible cyber security standards required to protect user data during sensitive data entries via forms, etc. Securing the websites with a SSL certificate which is a global security standard ensuring encrypted communication between the user's browser and server would be a high priority if the EDEN project is to be deployed for an industrial based usage.

6.1.2 IMPLEMENTATION ISSUES

The major issues regarding the implementation are with regards to developing the EDEN project modules and ensuring features and operations don't hamper with other modules during normal execution of the overall application project. The modules under observation are:

GROOMERS AND VETERINARIANS

- **Redundant Data:** The data given about animals that are for common for a single household can have duplicate data if each member in the family creates a personal ID and enters the same Pet information.
- **Data Updates:** The data in the animal's profile needs to be updated every year by the person in-charge of handling the data.
- **Dummy accounts:** Ensure dummy accounts are not created by personnel to be misused and provide fake feedbacks regarding the employees.

NGO

- **Repetition of data:** The data given about animals that are for adoption and event can have duplicate data if the person enters it twice as both data will be treated as unique data.
- **Changing data:** The data about animals needs to be updated every year by the person in-charge of handling the data.
- **PET ACCESSORIES**

- **Inadequate data:** Given that the modules were built from scratch, some areas that require data to build models could not be implemented. For example, in the pet accessories module, areas like product recommendations and relations required lots of data to train a model based on the products we could provide. This was not possible as the data requirements (at least 10,000 records for recommendation system) could not be accomplished within the project time frame.

- **Change over:** Moving from development environment to the production environment was tedious as a lot of operating system dependencies had to be dealt with.

6.2 ADVANTAGES AND LIMITATIONS

ADVANTAGES

One drawback of existing systems with respect to pet related applications is that all these apps provide a subset of the facilities required for animals i.e., grooming, adoption, medical support, etc. The development of the EDEN Project is providing users with a one stop solution wherein users and register their pet's details and avail the facilities all pets require, all from a single application. The application will cater to the target audience's need for pet accessories, Veterinarian, NGO and Groomer services. Thus, ensuring users need not search online for pet related outlets or applications.

LIMITATIONS

- Could not implement an AI powered recommendation and relation system because of inadequate data to train the machine learning model.
- Could not provide 24/7 medical help for domestic animals.
- Displaying appropriate recommendations in the product and shopping cart pages because of no machine learning model.
- Users cannot request NGO to take care of their animal when they are unavailable to do it.

6.3 FUTURE SCOPE OF THE PROJECT

- Providing or participating in online marketplaces, which process third-party business-to-consumer (B2C) or consumer-to-consumer (C2C) sales.
- Exposing certain sections of the accessories shop through APIs. This would allow other applications to extend or include certain parts of your application in their own application.
- Package tracking for the accessories shop.
- Recommendation system (no time to collect enough data to train ML model).
- Implementing emergency contact response for a veterinarian.
- Dynamically updating the various charts based on statistical information gathered for the groomer and veterinarian module.
- Implementing code segments to send the concerned veterinarian/groomer, list of appointments assigned to them.

- Add the ability to make appointments and access to pet boarding directly from the mobile client.
- Add pet boarding.
- Implement a fining system for multiple false alerts from a single user.
- Ranking veterinarians, groomers and NGOs.

REFERENCES

1. TailsLife: Complete Pet Care App
https://www.tailslife.com/

2. PetFirstAid:
https://www.petfirstaid.org/

3. Amazon – Pet Supplies
https://www.amazon.in/Pet-Supplies/b?ie=UTF8&node=2454181031

4. HeadsUpForTails – Pet Accessories
https://headsupfortails.com/?gclid=CjwKCAjw95yJBhAgEiwAmRrutKmJb9wAG8kriYgF_dtpUkHJvab8fJyV5GeuhJrZdRi_lCCsqrftohoCGd4QAvD_BwE

5. JustDial – Pet Adoption
https://www.justdial.com/Bangalore/Dog-Adoption-Centres/nct-10168823

6. Bangalore Pet Hospital
http://bangalorepethospital.in/

7. NGOs for pets
https://www.scoopwhoop.com/ngos-for-animals/