

Sri Lanka Institute of Information Technology



IT2080 - Information Technology Project

Year 2 Semester 2 - 2025

Assignment 02

[Progress Review]

ITP25_B7.2_168

**Project Title: Veterinary Management System
(PetIQ.lk)**

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Group Details

	IT Number	Student Name	Student E-mail Address	Contact Number
1	IT23640948	Gamlath K. G. V. K. D	it23640948@my.sliit.lk	0702943487
2	IT23631106	Kavindu J. M. R	it23631106@my.sliit.lk	0703274142
3	IT23594722	Wanasinghe W. M. D. T	it23594722@my.sliit.lk	0774376338
4	IT23413474	Rathnayake W. P. D. D. W	it23413474@my.sliit.lk	0702979383
5	IT23631274	Abeyasinghe A. M. B. N	it23631274@my.sliit.lk	0761260885

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Contents

1.1	Introduction.....	4
1.2	User Stories.....	6
1.3	Completed Features	7
1.4	In Progress	8
1.5	To-Do List	9
1.6	Repository Link.....	9
2.1	ER Diagram	10
2.2	Normalized schema – (3NF).....	11
2.5	High-Level System Design Diagram.....	17
2.6	Innovative Parts of the Project	18
2.7	Commercialization	19
3.0	Individual contribution	20
	3.1 - Gamlath K.G.V.K.D (IT23640948).....	20
	3.2 - Kavindu J M R (IT23631106)	25
	3.3 - Wanasinghe W.M.D.T (IT23594722).....	32
	3.4 - Rathnayake W P D D W (IT23413474)	38
	3.5 - Abeysinghe A M B N (IT23631274)	42

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

1.1 Introduction

The Veterinary Management System (PetIQ) is designed to automate and modernize the operations of the PetIQ Animal Hospital in Athurugiriya, Sri Lanka. At present, the hospital relies on manual, paper-based methods for managing appointments, medical records, product sales, and payments. These outdated processes are inefficient, prone to errors, and lack transparency resulting in service delays and subpar customer experience.

Problem Statement:

- Manual record-keeping leads to inconsistent data and makes it difficult to retrieve pet histories.
- Scheduling appointments is time-consuming and often results in double-bookings.
- Payments are handled offline, leading to inaccurate financial records and delayed reporting.
- There is no integration between hospital services and pet-mart purchases, resulting in disjointed workflows

Objectives of the VMS:

- Provide secure, role-based access for veterinarians, receptionists, pet owners, and administrators.
- Enable real-time online appointment booking and scheduling.
- Maintain up-to-date and easily accessible medical records.
- Manage pet-mart inventory and collect delivery information for online purchases.
- Integrate a secure Stripe-based payment gateway that supports card CRUD operations, real-time validation, and transaction history.
- Generate reports and analytics to assist administrators in decision-making.

Scope of the System:

- In Scope: Appointment scheduling, pet medical records, pet-mart sales and deliveries, payment processing, and reporting dashboards.
- Out of Scope: Pharmacy inventory outside the pet-mart, HR payroll systems, and external lab integrations.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Technology Stack:

- Frontend: React with Tailwind CSS (responsive UI design)
- Backend: Node.js with Express.js and RESTful APIs
- Database: MongoDB Atlas (cloud-based, schema designed in 3NF)
- Payment Gateway: Stripe API (PCI-DSS compliant)
- Deployment: Vercel or Netlify for the front end, Render or Railway for the backend

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

1.2 User Stories

- **US 001:** As a pet owner, I want to register and log in securely so that I can manage my appointments and orders.
- **US 002:** As a pet owner, I want to book veterinary appointments online so that I don't have to visit the hospital physically to book veterinary appointments.
- **US 003:** As a veterinarian, I want to record treatments and prescriptions so that each pet's medical history is accurately maintained.
- **US 004:** As a receptionist, I want to approve or reschedule appointments so that hospital resources are managed efficiently.
- **US 005:** As a pet owner, I want to browse and purchase products from the pet-mart online so that I can have them delivered conveniently.
- **US 006:** As a customer, I want to provide my delivery details so that my orders are delivered to the correct address on time.
- **US 007:** As an admin, I want to manage order dispatch and deliveries so that customers are kept informed.
- **US 008:** As a pet owner, I want to securely save up to three payment cards so that I can make quick and easy payments.
- **US 009:** As a pet owner, I want to pay for hospital services and pet-mart products through a secure gateway so that I have a valid transaction record.
- **US 010:** As an admin, I want to view unalterable payment transaction logs so that I can perform audits effectively.
- **US 011:** As a pet owner, I want to view my past appointments, so that I can track my pet's healthcare.
- **US 012:** As an admin, I want to generate reports on appointments, sales, and revenue so that I can make informed strategic decisions.
- **US 013:** As an admin, I want to monitor delivery performance so that I can enhance operational efficiency.

1.3 Completed Features

Authentication & Security

- Secure registration and login with role-based access control.
- Passwords are hashed and validated using JWT authentication.

Appointment Management

- Users can book, update, and cancel appointments.
- Real-time availability of time slots is displayed.

Medical Records

- Full CRUD (Create, Read, Update, Delete) functionality for pet medical history, prescriptions, and treatments.
- Full access is restricted to administrators.

Pet Product Management

- Completed Backend CRUD (Create, Read, Update, Delete) functions
- Completed Frontend Admin Dashboard

Payment Gateway

- Integrated with Stripe Elements for secure card entry.
- Users can add, edit, or delete up to three cards per account.
- Field-level validations include Luhn checks, expiry date, and CVV verification.
- All transaction logs are securely stored in MongoDB Atlas.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Appendix

High level user story/ function	Completed (Y/N)	Responsible member (Name)
Appointment Management	Y	Gamlath K. G. V. K. D
User Management	Y	Kavindu J. M. R
Medical Records Management	Y	Wanasinghe W. M. D. T
Pet Product Management	Y	Rathnayake W. P. D. D. W
Payment Management	Y	Abeyasinghe A. M. B. N

1.4 In Progress

UI/UX Improvements

- Tailwind CSS-based responsive layouts for appointment, pet-mart, and payment pages.
- Enhanced review page with error messages, colored status icons, and confirmation pop-ups.

Delivery Management

- Delivery details are integrated directly into the payment process.
- Order tracking features are being developed for customer use.

Admin Dashboards

- A read-only payments record part is available in admin dashboard with filters such as payment source and reference.
- Initial setup of admin dashboard allows viewing saved cards (edit/delete not available).

1.5 To-Do List

Critical Tasks

- Generate payment receipts in PDF format (using ReportLab or jsPDF).
- Integrate Stripe Webhooks to handle real-time payment success/failure notifications.

Major Tasks

- Enable CSV export of transaction data.
- Build analytics dashboards with visual charts (using Recharts or Chart.js).

Normal Tasks

- Implement a notification system for email/SMS alerts about bookings, orders, and payments.
- Improve validation in the delivery form (e.g., address and phone number accuracy).

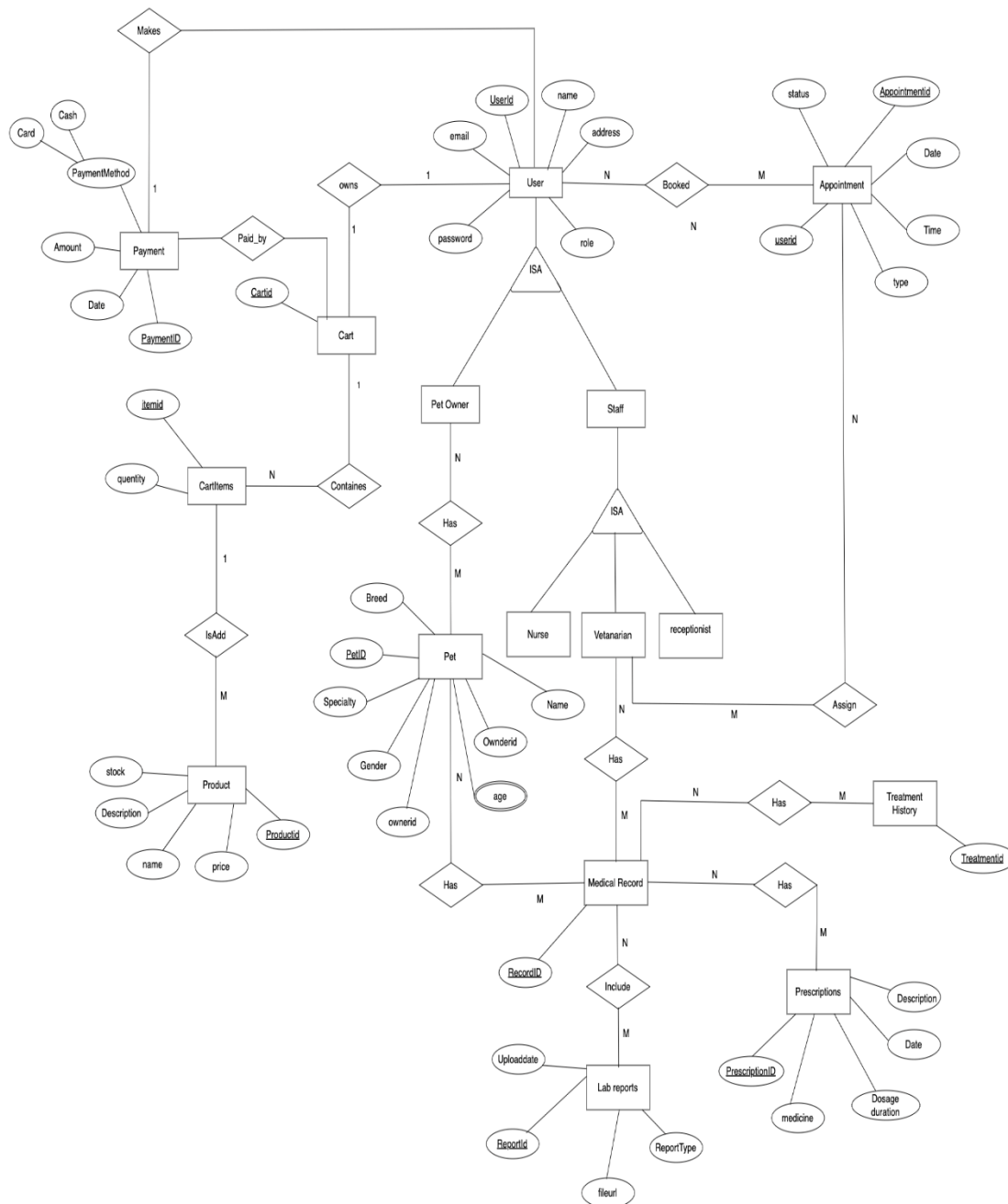
Minor Tasks

- UI enhancements such as centering the success page and polishing icon designs.
- Perform mobile-first testing for both hospital and pet-mart workflows.

1.6 Repository Link

- mernpetiq : <https://github.com/VishwaKeshara/mernpetiq.git>

2.1 ER Diagram



2.2 Normalized schema – (3NF)

2.2.1 Appointment Schema

```
1  const mongoose = require("mongoose");
2
3  const appointmentSchema = new mongoose.Schema({
4
5    ownerName: { type: String, required: true },
6    petName: { type: String, required: true },
7    petType: { type: String, required: true },
8    service: { type: String, required: true },
9    price: { type: Number, required: true },
10   vet: { type: String, required: true },
11   date: { type: String, required: true },
12   time: { type: String, required: true },
13   createdAt: { type: Date, default: Date.now },
14
15  });
16
17  module.exports = mongoose.model("Appointment", appointmentSchema);
```

2.2.2 Employee Schema

```
import mongoose from "mongoose";
import bcrypt from "bcryptjs";

const employeeSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: {
    type: String,
    required: true,
    enum: ['admin', 'veterinarian', 'nurse', 'receptionist'],
    default: 'admin'
  },
  avatarUrl: { type: String },
  isActive: { type: Boolean, default: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

B.Sc. (Hons) in Information Technology

IT2080 - IT Project

Activity 02 - Requirements Engineering Activity

2.2.3 Register Schema

```
import mongoose from "mongoose";
import bcrypt from "bcryptjs";

const registerSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  avatarUrl: { type: String },
  phone: { type: String },
  addressLine1: { type: String },
  addressLine2: { type: String },
  city: { type: String },
  state: { type: String },
  postalCode: { type: String },
  country: { type: String },
  dateOfBirth: { type: Date },
  isActive: { type: Boolean, default: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

2.2.4 Medical Record Schema

```
1  const mongoose = require('mongoose');
2
3  const medicalRecordSchema = new mongoose.Schema({
4    petName: {type: String,required: [true, 'Pet name is required'],trim: true,maxlength: [50, 'Pet name cannot exceed 50 characters']},
5    owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: [true, 'Owner is required']},
6    ownerName: {type: String,required: [true, 'Owner name is required'],trim: true},
7    petType: { type: String, required: [true, 'Pet type is required'],enum: ['Dog', 'Cat', 'Bird', 'Rabbit'],
8    breed: { type: String, trim: true, maxlength: [50, 'Breed cannot exceed 50 characters']},
9    age: { type: String,trim: true, maxlength: [20, 'Age cannot exceed 20 characters']},
10   weight: { type: String,trim: true,maxlength: [20, 'Weight cannot exceed 20 characters']},
11   symptoms: {type: String,trim: true,maxlength: [1000, 'Symptoms cannot exceed 1000 characters']},
12   diagnosis: { type: String, trim: true, maxlength: [1000, 'Diagnosis cannot exceed 1000 characters']},
13   treatment: {type: String,trim: true,maxlength: [1000, 'Treatment cannot exceed 1000 characters']},
14   prescription: {type: String,trim: true,maxlength: [1000, 'Prescription cannot exceed 1000 characters']},
15   notes: {type: String, trim: true, maxlength: [2000, 'Notes cannot exceed 2000 characters']},
16   visitDate: {type: Date,required: [true, 'Visit date is required'],default: Date.now},
17   veterinarian: {type: mongoose.Schema.Types.ObjectId,ref: 'User',required: [true, 'Veterinarian is required']},
18   veterinarianName: {type: String,required: [true, 'Veterinarian name is required']},
19   attachments: [{filename: String,originalName: String,mimetype: String,size: Number,
20   uploadDate: { type: Date, default: Date.now}}],
21   status: { type: String,enum: ['active', 'completed', 'archived'],default: 'active'}
22 }, {
23   timestamps: true
24 });
25
26 // TODO: For better performance
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

2.2.5 Pet Product Schema

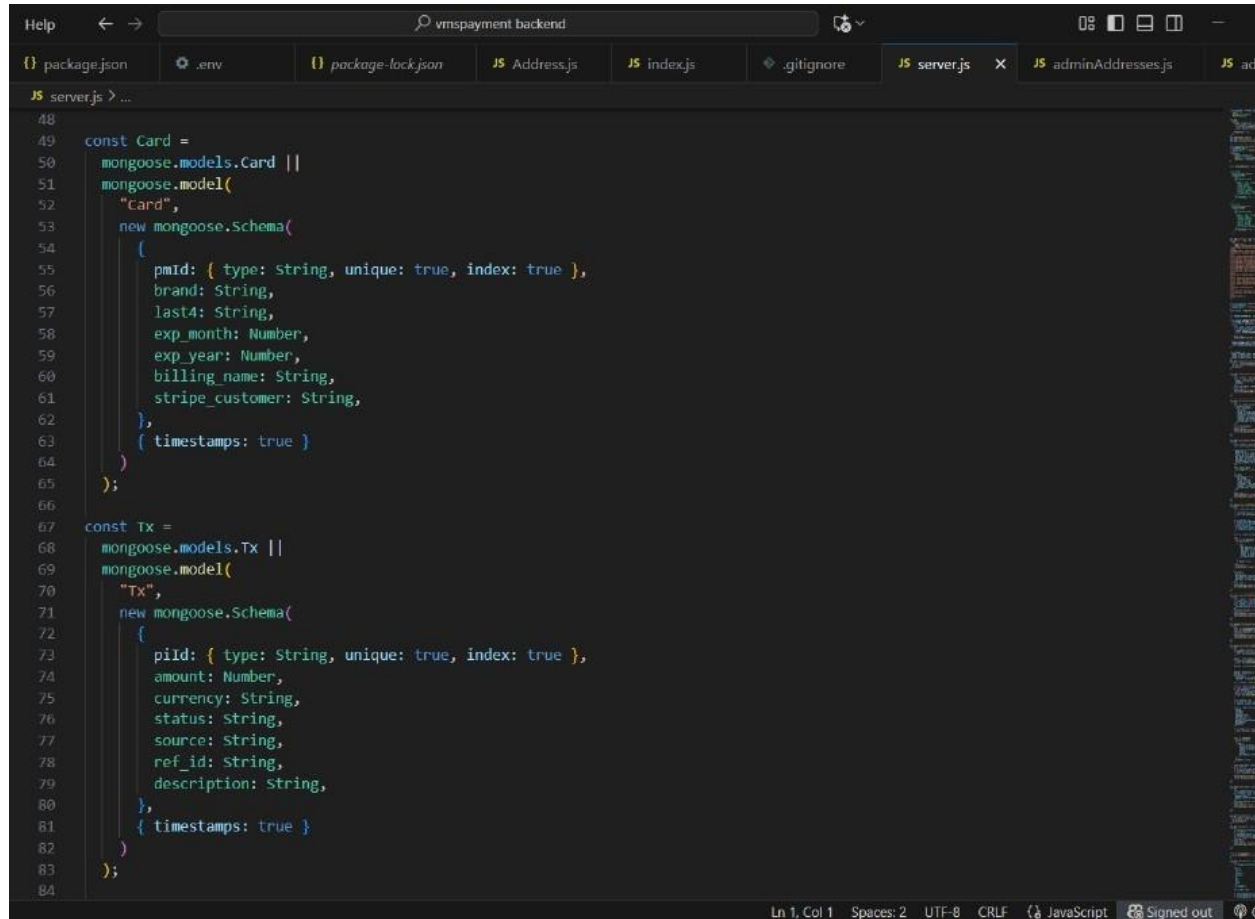
```
server > Model > JS ProductModel.js > default
You, 6 seconds ago | 1 author (You)
1  import mongoose from "mongoose";
2
3  const productSchema = new mongoose.Schema({
4
5      name :{ type: String, required: true },
6      price:{ type: Number, required: true },
7      description: { type: String, required: true },
8      category: { type: String, required: false, default: "Other"},
9      stock: { type: Number, required: true, min: 0 },
10     image: { type: String, required: true },
11
12 }, {
13     timestamps: true // createdAt, updatedAt
14 });
15
16 const Product = mongoose.model('Product', productSchema);
17
18 export default Product;
```

2.2.6 Address Schema

```
Help  ← →  vmispayment backend  [Icons]
[package.json] [env] [package-lock.json] JS Address.js X JS index.js [gitignore] JS server.js JS adminAddresses.js JS addri ...
models > JS Address.js > ...
1
2  const mongoose = require("mongoose");
3
4  const AddressSchema = new mongoose.Schema(
5  {
6      userId: { type: String, required: true, index: true, default: "guest" },
7
8      firstName: { type: String, required: true, trim: true },
9      lastName: { type: String, required: true, trim: true },
10     phone: { type: String, required: true, trim: true },
11
12     line1: { type: String, required: true, trim: true },
13     line2: { type: String, trim: true, default: "" },
14     city: { type: String, required: true, trim: true },
15
16     state: { type: String, required: true, trim: true },
17     postalCode:{ type: String, trim: true, default: "" },
18     country: { type: String, trim: true, default: "Sri Lanka" },
19 },
20 { timestamps: true }
21 );
22
23 module.exports = mongoose.model("Address", AddressSchema);
24
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

2.2.8 Card Details Schema



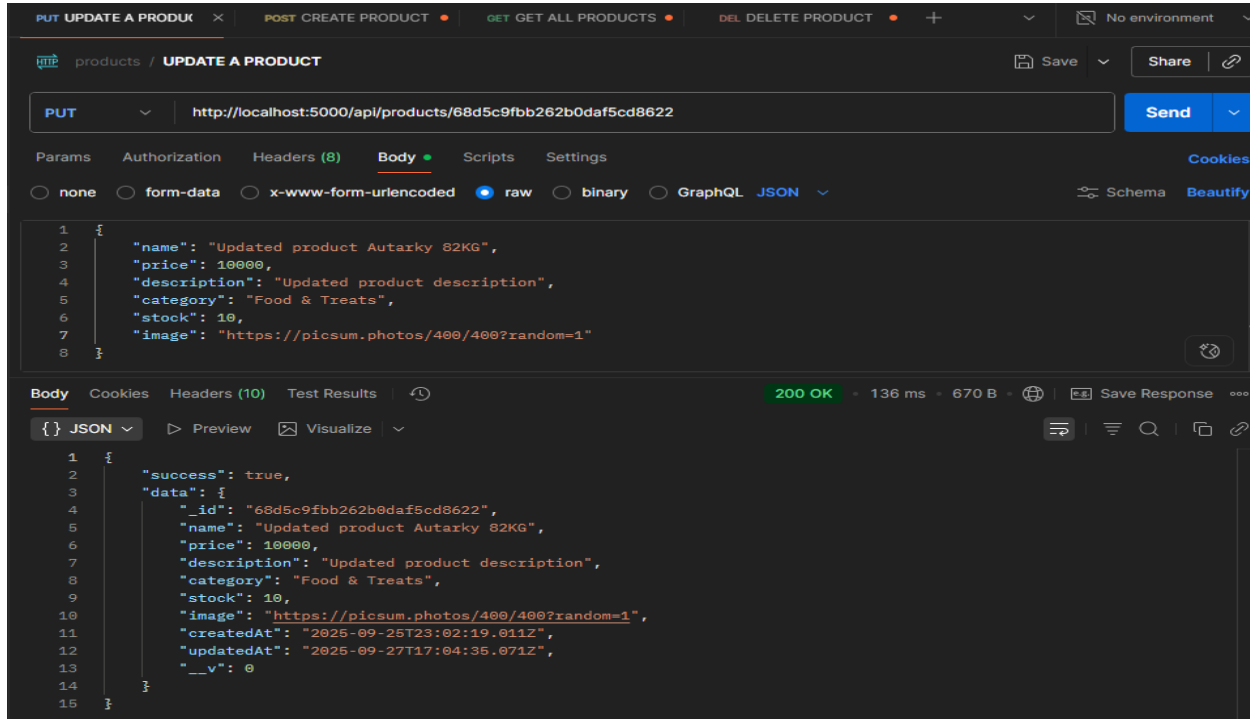
```
48
49 const Card =
50   mongoose.models.Card ||
51   mongoose.model(
52     "card",
53     new mongoose.Schema(
54       {
55         pmId: { type: String, unique: true, index: true },
56         brand: String,
57         last4: String,
58         exp_month: Number,
59         exp_year: Number,
60         billing_name: String,
61         stripe_customer: String,
62       },
63       { timestamps: true }
64     )
65   );
66
67 const Tx =
68   mongoose.models.Tx ||
69   mongoose.model(
70     "Tx",
71     new mongoose.Schema(
72       {
73         piId: { type: String, unique: true, index: true },
74         amount: Number,
75         currency: String,
76         status: String,
77         source: String,
78         ref_id: String,
79         description: String,
80       },
81       { timestamps: true }
82     )
83   );
84
```

B.Sc. (Hons) in Information Technology

IT2080 - IT Project

Activity 02 - Requirements Engineering Activity

2.3 Test Cases



The screenshot shows a REST client interface with a PUT request to `http://localhost:5000/api/products/68d5c9fbb262b0daf5cd8622`. The request body is a JSON object with the following fields:

```

1 {
2   "name": "Updated product Autarky 82KG",
3   "price": 10000,
4   "description": "Updated product description",
5   "category": "Food & Treats",
6   "stock": 10,
7   "image": "https://picsum.photos/400/400?random=1"
8 }

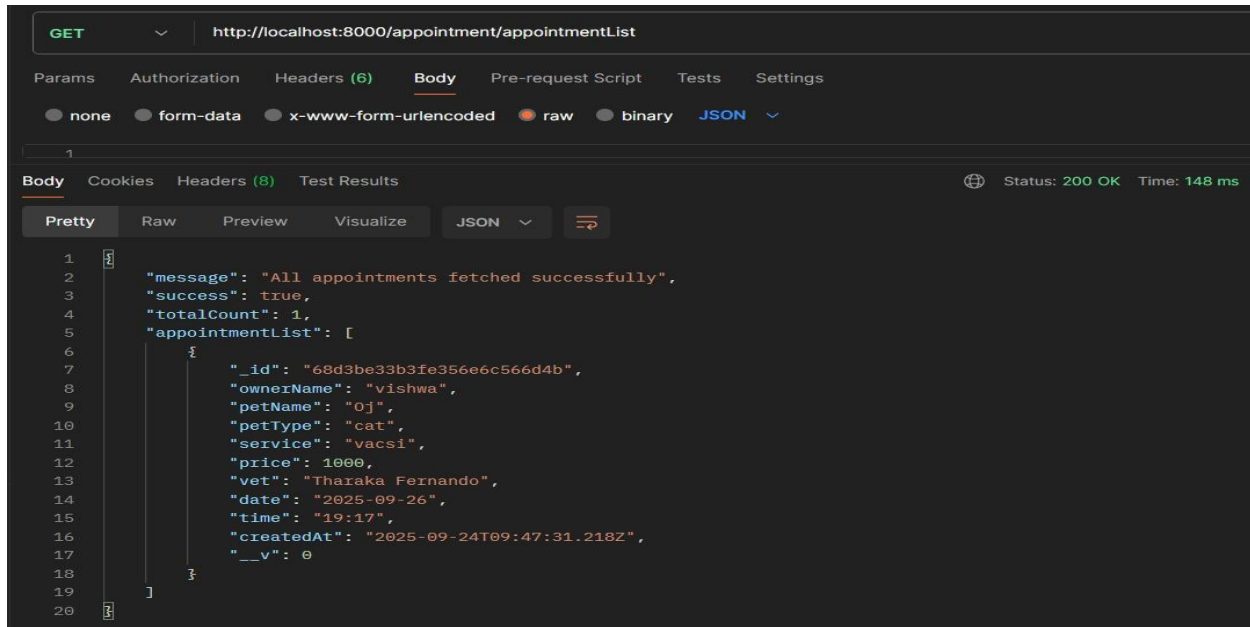
```

The response is a 200 OK status with a response time of 136 ms and a body size of 670 B. The response body is a JSON object:

```

1 {
2   "success": true,
3   "data": {
4     "_id": "68d5c9fbb262b0daf5cd8622",
5     "name": "Updated product Autarky 82KG",
6     "price": 10000,
7     "description": "Updated product description",
8     "category": "Food & Treats",
9     "stock": 10,
10    "image": "https://picsum.photos/400/400?random=1",
11    "createdAt": "2025-09-26T23:02:19.011Z",
12    "updatedAt": "2025-09-27T17:04:35.071Z",
13    "__v": 0
14  }
15 }

```



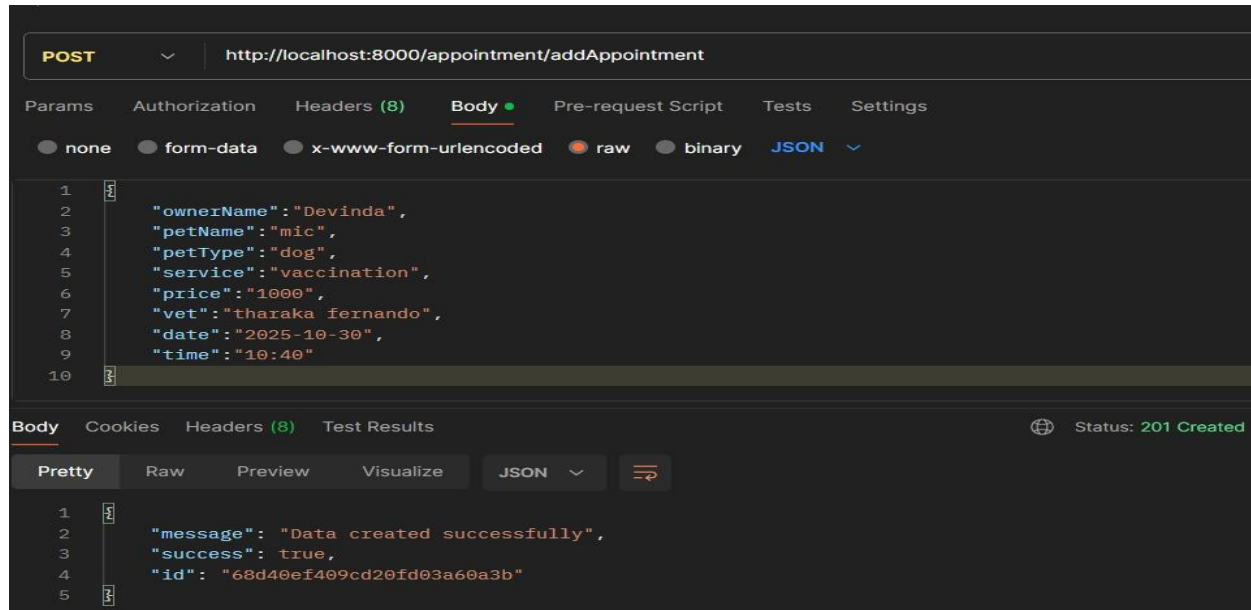
The screenshot shows a REST client interface with a GET request to `http://localhost:8000/appointment/appointmentList`. The response is a 200 OK status with a response time of 148 ms. The response body is a JSON object:

```

1 {
2   "message": "All appointments fetched successfully",
3   "success": true,
4   "totalCount": 1,
5   "appointmentList": [
6     {
7       "_id": "68d3be33b3fe356e6c566d4b",
8       "ownerName": "vishwa",
9       "petName": "Oj",
10      "petType": "cat",
11      "service": "vacsi",
12      "price": 1000,
13      "vet": "Tharaka Fernando",
14      "date": "2025-09-26",
15      "time": "19:17",
16      "createdAt": "2025-09-24T09:47:31.218Z",
17      "__v": 0
18    }
19  ]
20 }

```


B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity



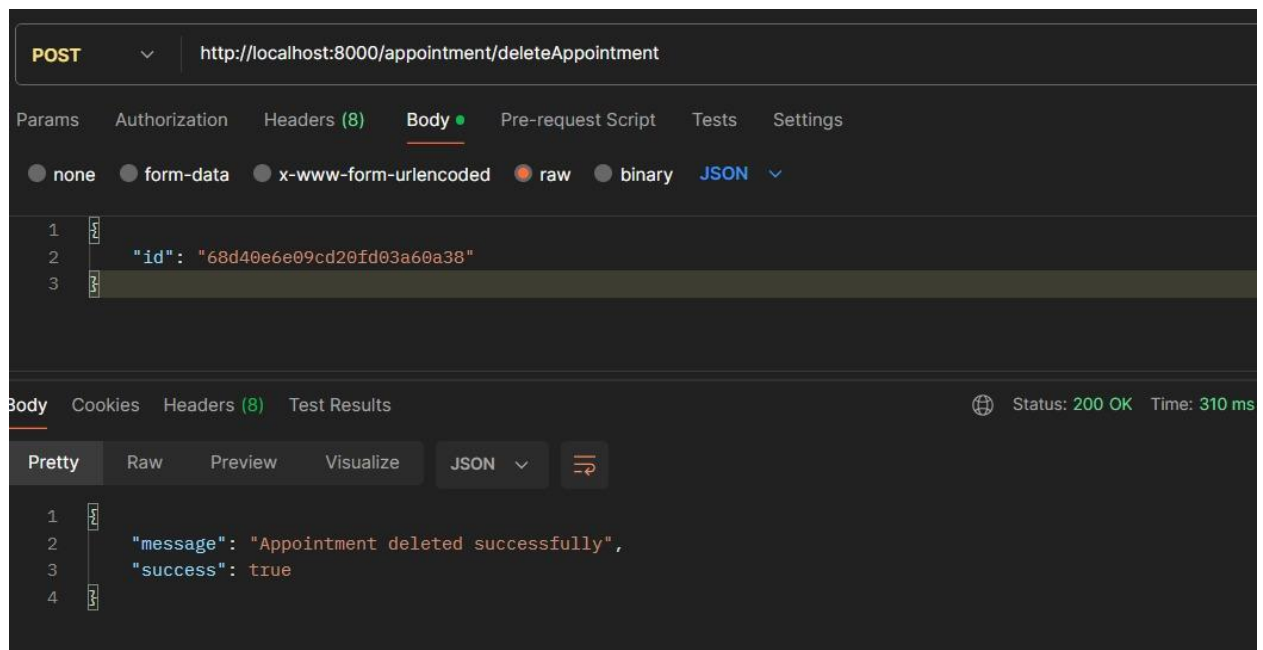
The screenshot shows a REST client interface with a POST request to `http://localhost:8000/appointment/addAppointment`. The request body is a JSON object with the following fields: `ownerName`, `petName`, `petType`, `service`, `price`, `vet`, `date`, and `time`. The response status is 201 Created, and the response body is a JSON object with `message`, `success`, and `id` fields.

```
POST http://localhost:8000/appointment/addAppointment

{
  "ownerName": "Devinda",
  "petName": "mic",
  "petType": "dog",
  "service": "vaccination",
  "price": "1000",
  "vet": "tharaka fernando",
  "date": "2025-10-30",
  "time": "10:40"
}
```

Status: 201 Created

```
{
  "message": "Data created successfully",
  "success": true,
  "id": "68d40ef409cd20fd03a60a3b"
}
```



The screenshot shows a REST client interface with a POST request to `http://localhost:8000/appointment/deleteAppointment`. The request body is a JSON object with the `id` field. The response status is 200 OK, and the response body is a JSON object with `message` and `success` fields.

```
POST http://localhost:8000/appointment/deleteAppointment

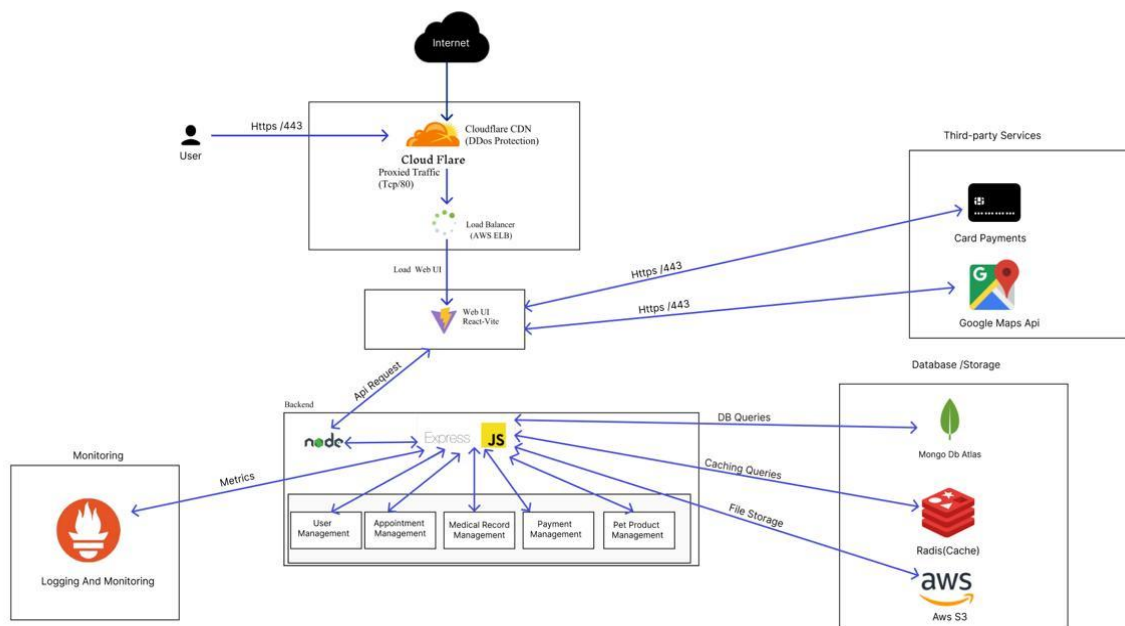
{
  "id": "68d40e6e09cd20fd03a60a38"
}
```

Status: 200 OK Time: 310 ms

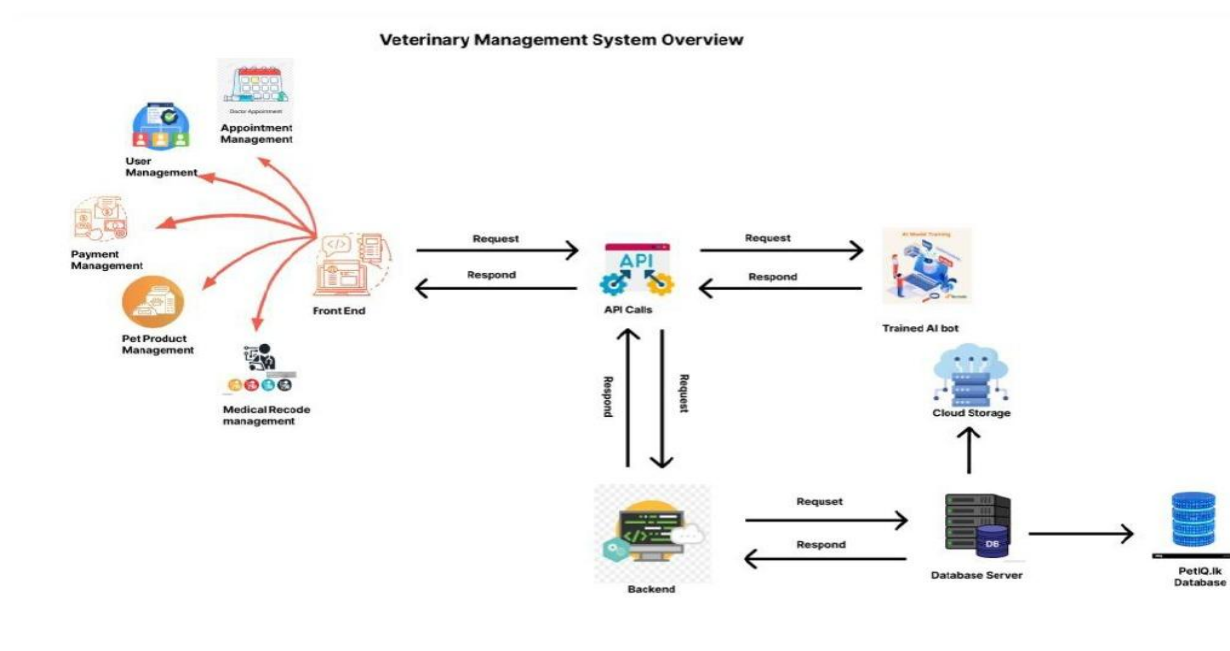
```
{
  "message": "Appointment deleted successfully",
  "success": true
}
```


B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

2.4 Network Designing



2.5 High-Level System Design Diagram



2.6 Innovative Parts of the Project

Smart Appointment and Queue Management:

The system offers an online appointment booking feature with automated scheduling. Real-time updates and notifications are sent to pet owners to minimize waiting times and improve overall clinic efficiency. Queue management ensures a smoother workflow for veterinarians while enhancing client satisfaction.

Digital Medical Records and Prescription Management:

All medical records, including vaccination history, treatments, and prescriptions, are securely stored in the system. Veterinarians can instantly retrieve a pet's complete health history, while owners receive digital prescriptions and dosage reminders, ensuring continuity of care and reducing the risk of errors.

Automated Vaccination and Health Reminders:

The platform provides automated reminders for vaccinations, deworming, and routine check-ups via SMS and email. This proactive feature supports preventive healthcare and contributes to the long-term well-being of pets.

Integrated Payment and Invoice Generation:

PetIQ.LK integrates secure online payment gateways for consultations, treatments, and medication purchases. Upon transaction completion, digital invoices and downloadable PDF records are automatically generated, improving transparency and simplifying financial record-keeping for clinics and clients alike.

Telemedicine and Virtual Consultation:

The system enables remote veterinary consultations through video conferencing for minor health concerns or follow-ups. This feature improves accessibility, particularly in rural areas where veterinary services are limited, while reducing unnecessary in-person visits.

Pet Health Analytics and Report Generation:

Comprehensive health analytics and periodic reports are generated for each pet, including growth tracking, vaccination progress, and treatment outcomes. These reports empower pet owners with valuable insights and assist veterinarians in data-driven clinical decision-making.

2.7 Commercialization

Market Research:

Market research conducted within Sri Lanka indicates a growing trend in pet ownership and an increasing demand for digitized veterinary services. Current challenges such as inefficient appointment systems, lack of centralized medical history, and limited remote consultation options highlight the market potential for PetIQ.LK. By addressing these pain points, the system aims to fulfill unmet needs within the veterinary sector.

Competitive Analysis:

Most veterinary clinics in Sri Lanka rely on traditional management practices with limited digital integration. PetIQ.LK differentiates itself by offering AI-powered health assessments, automated reminders, telemedicine services, and comprehensive digital record-keeping. These innovations provide a superior, technology-driven alternative compared to existing systems.

Customer Support and Feedback Mechanism:

The platform includes a dedicated customer support module to resolve issues efficiently. Users can rate services and provide feedback, ensuring continuous improvement in service quality. Furthermore, AI-powered chatbots handle frequently asked questions related to appointments, clinic timings, and vaccination schedules, thereby enhancing user experience.

Legal Compliance and Veterinary Standards:

PetIQ.LK ensures full compliance with national veterinary regulations and data protection policies. The system also adheres to industry-standard practices for medical record confidentiality and secure financial transactions. Collaborations with veterinary associations and government agencies will further validate system credibility and sustainability.

3.0 Individual contribution

3.1 - Gamlath K.G.V.K.D (IT23640948)

(Appointment Management)

3.1.1 Completion Level:

Current Status: **80%** Completed

3.1.2 Completed

- Developed appointment creation, updating, and deletion functionalities.
- Implemented appointment listing with search and filtering features.
- Added validation for required fields, including date and time restrictions.
- Restricted past date/time selections to ensure valid bookings.
- Integrated backend APIs for appointment CRUD operations.
- Secured storage of appointment details in the database.
- Designed and implemented a responsive, user-friendly vertical form for appointment entry.
- Navigation is enabled from appointment list to update form.
- Designed a detailed flowchart to represent the appointment management process.

3.1.3 To Be Completed

- Integration with the payment module for appointment fee handling.
- Visualization of appointment statistics using charts and graphs.
- Implementation of automated email/SMS notifications for confirmed appointments.
- Enhancement of UI/UX for improved usability and accessibility.
- Role-based access control for different user types (admin, vet, receptionist).

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.1.4 Queries

- Create Appointment

```
const appointmentAdd = await Appointment.create(body);
```

- Read All Appointments

```
const appointmentList = await Appointment.find({});
```

- Delete Appointment

```
const deleted = await Appointment.deleteOne({ _id: id });
```

- Update Appointment

```
const updating = await Appointment.updateOne({ _id }, { $set: body });
```

```
const { data } = await appointmentBaseUrl.post("/deleteAppointment", { id });
```

3.1.5 Algorithm

- Time Conflict Detection Algorithm

```
const checkForConflict = (vet, date, time, excludeId = null) => {  
  if (!vet || !date || !time) return false;  
  const timeToMinutes = (timeStr) => {  
    const [hours, minutes] = timeStr.split(":").map(Number);  
    return hours * 60 + minutes;  
  };  
  const newStart = timeToMinutes(time);  
  const newEnd = newStart + 60;  
  return existingAppointments.some((a) => {  
    if (a.vet !== vet || a.date !== date || a._id === excludeId) return false;  
    const existingStart = timeToMinutes(a.time);  
    const existingEnd = existingStart + 60;  
    return newStart < existingEnd && newEnd > existingStart;  
  });  
};
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

- Time Calculation Algorithm

```
const getEndTime = (startTime) => {  
  const [hours, minutes] = startTime.split(":").map(Number);  
  const endHours = hours + 1;  
  return `${endHours.toString().padStart(2, "0")}:${minutes  
    .toString()  
    .padStart(2, "0")}`;  
};
```

- 24-Hour to 12-Hour Time Conversion Algorithm

```
const toAmPm = (time24) => {  
  if (!time24) return "";  
  const [h, m] = time24.split(":").map(Number);  
  const period = h >= 12 ? "PM" : "AM";  
  const hour12 = h % 12 === 0 ? 12 : h % 12;  
  return `${hour12.toString().padStart(2, "0")}:${m.toString().padStart(2, "0")} ${period}`;  
};
```

- Multi-Field Search/Filter Algorithm

```
const filteredAppointments = appointmentList?.filter((appointment) => {  
  const query = searchQuery.trim().toLowerCase();  
  if (!query) return true;  
  const fieldsToSearch = [  
    appointment?.ownerName,  
    appointment?.petName,  
    appointment?.petType,  
    appointment?.service,  
    appointment?.vet,  
    appointment?.date,  
    toAmPm(appointment?.time),  
    String(appointment?.price),  
  ];  
  return fieldsToSearch.some((value) => String(value || "").toLowerCase().includes(query));  
});
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

- URL Parameter Parsing Algorithm

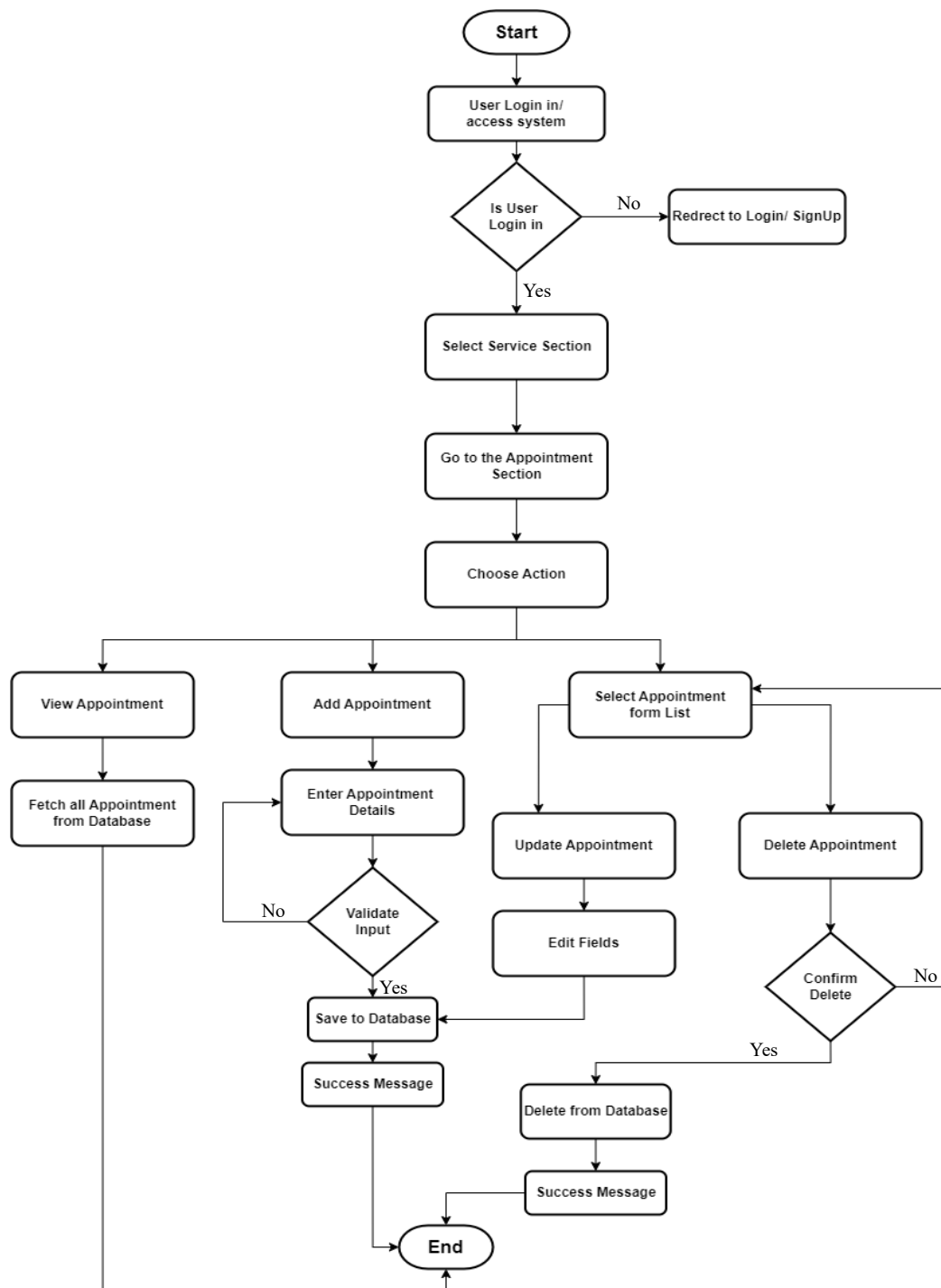
```
useEffect(() => {  
  if (!isUpdating && location.search) {  
    const params = new URLSearchParams(location.search);  
    const service = params.get("service") || "";  
    const price = params.get("price") || "";  
    if (service || price) {  
      setAppointmentForm((prev) => ({  
        ...prev,  
        service,  
        price: price ? String(price) : prev.price,  
      }));  
    }  
  }  
}, [isUpdating, location.search]);
```

- Data Validation Algorithm

```
if (Object.values(appointmentForm).some((f) => !f)) {  
  alert("All fields are required!");  
  return;  
}  
if (  
  appointmentForm.time < OPEN_TIME ||  
  appointmentForm.time > LAST_START_TIME  
) {  
  alert("Please select a time between 08:00 and 23:00.");  
  return;  
}
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

- Flow Chart



B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.2 - Kavindu J M R (IT23631106)

(User Management)

3.2.1 Completion Level:

Current Status: **80%** Complete

3.2.2 Completed (Done)

- Users (e.g., pet owners, veterinarians, staff) can register and create profiles with role-specific details.
- Admins can create, edit, or deactivate any user profile.
- Veterinarians can update their availability and specializations.
- Pet owners can add multiple pets to their profiles, with details such as species, breed, age, and medical history.
- Role-based access control implemented:
- Veterinarians can access assigned patient records.
- Pet owners can view their own pet data and appointment history.
- Admins have full access.
- Login/Logout system with session tracking and user authentication.
- Email verification during sign-up.
- Admin dashboard includes user statistics (active users, roles distribution, etc.).
- Search, filter, and sort users by name, role, registration date, or status.
- Activity logs to track major user actions (e.g., login, profile update, deletion).

3.2.3 Work Not Completed (Doing)

- **Advanced User Features in Progress:**
 - User profile picture upload with validation and cropping tool.
 - Notification settings per user type (e.g., appointment reminders, follow-ups).
 - Admin bulk user import via CSV/XLSX.
 - Role delegation: allow admins to assign limited admin rights to other staff.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.2.4 To Be Completed (To-Do)

- **Veterinarian Rating & Performance Tracking:**
 - Implement a system to collect ratings from pet owners after appointments.
 - Log completed appointments and use them to calculate experience/performance indicators.
- **Dynamic Appointment Updates:**
 - Enable real-time updates for appointment rescheduling or cancellations.
 - Adjust vet availability automatically if appointment slots are canceled or filled.
- **Veterinarian Leave Request & Approval System:**
 - Allow veterinarians to request leave via their dashboard with date, reason, and optional notes.
 - Approved leave dates make the vet unavailable for bookings automatically.

3.2.5 Queries used

Pet Owner Section

Check if user already exists (email search)

```
const existingUser = await RegisterModel.findOne({ email });
```

Save new users

```
const savedPetOwner = await newPetOwnerData.save();
```

Find user by email (for login)

```
const user = await RegisterModel.findOne({ email });
```

Get all register data (excluding password)

```
const allAdmins = await RegisterModel.find({}).select('-password');
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Get register data by ID (exclude password)

```
const admin = await RegisterModel.findById(id).select('-password');
```

Update register data by ID

```
const updatedUser = await RegisterModel.findByIdAndUpdate(  
  id,  
  filteredUpdate,  
  { new: true, runValidators: true }  
) .select('-password');
```

Delete register data by ID

```
const deletedAdmin = await RegisterModel.findByIdAndDelete(id);
```

Reset password (update password by userId)

```
const updated = await RegisterModel.findByIdAndUpdate(  
  userId,  
  { password: hashed },  
  { new: true }  
) .select('-password');
```

Delete self account (delete by userId)

```
const deleted = await RegisterModel.findByIdAndDelete(userId);
```

B.Sc. (Hons) in Information Technology

IT2080 - IT Project

Activity 02 - Requirements Engineering Activity

Change password (find user by ID, then save after hashing new password)

```
const user = await RegisterModel.findById(userId);
if (!user) return res.status(404).json({ success: false, message: 'User not found' });

const isValid = await user.comparePassword(currentPassword);
if (!isValid) return res.status(401).json({ success: false, message: 'Current password is incorrect' });

const salt = await bcrypt.genSalt(10);
user.password = await bcrypt.hash(newPassword, salt);
await user.save();
```

Staff (Admin) Section

Check if user already exists (for register)

```
const existingUser = await EmployeeModel.findOne({ email });
```

Save new employee/admin

```
const savedAdmin = await newAdminData.save();
```

Get all admins (excluding password)

```
const allAdmins = await EmployeeModel.find({}).select('-password');
```

Get admin by ID

```
const admin = await EmployeeModel.findById(id).select('-password');
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Update admin by ID

```
const updatedAdmin = await EmployeeModel.findByIdAndUpdate(  
  id,  
  filteredUpdate,  
  { new: true, runValidators: true }  
)  
.select('-password');
```

Delete admin by ID

```
const deletedAdmin = await EmployeeModel.findByIdAndDelete(id);
```

Search admins with filters

```
const searchResults = await EmployeeModel.find(searchCriteria).select('-password');
```

Login – find admin by email

```
const user = await EmployeeModel.findOne({ email });
```

Reset password (update by userId)

```
const updated = await EmployeeModel.findByIdAndUpdate(  
  userId,  
  { password: hashed },  
  { new: true }  
)  
.select('-password');
```

Change password (find user by ID)

```
const user = await EmployeeModel.findById(userId);  
if (!user) return res.status(404).json({ success: false, message: 'User not found' });
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Delete self account

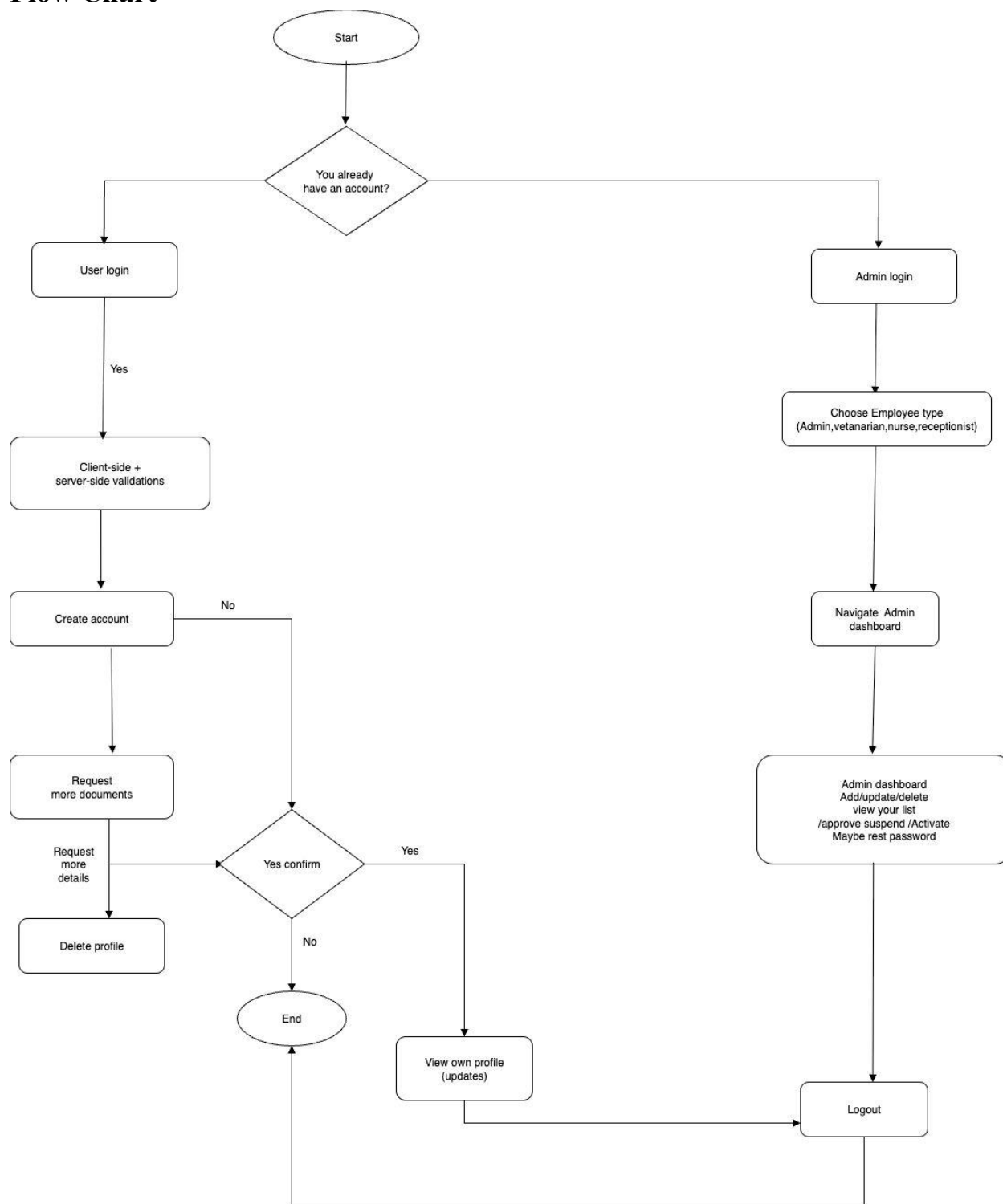
```
const deleted = await EmployeeModel.findByIdAndDelete(userId);  
if (!deleted) return res.status(404).json({ success: false, message: 'User not found' });
```

3.2.6 Algorithm (Mern Stack)

```
const getAdminDataById = async (req, res) => {  
  try {  
    const { id } = req.params;  
  
    const admin = await EmployeeModel.findById(id).select('-password');  
  
    if (!admin) {  
      return res.status(404).json({  
        success: false,  
        message: 'Admin data not found in MongoDB'  
      });  
    }  
  
    res.status(200).json({  
      success: true,  
      message: 'Admin data retrieved successfully',  
      data: {  
        admin: admin  
      }  
    });  
  } catch (error) {  
    console.error('Get admin data by ID error:', error);  
    res.status(500).json({  
      success: false,  
      message: 'Server error while retrieving admin data from MongoDB'  
    });  
  }  
};
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Flow Chart



B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.3 - Wanasinghe W.M.D.T (IT23594722)

(Medical Record Management)

3.3.1 - Completion Level

Current Status: 80% Complete

3.3.2 Completed Work (Done)

The foundational elements for managing basic animal patient records have been successfully implemented and tested. The following functionalities are complete:

Patient Registration and Profile Creation:

- A system to register new animal patients, capturing details such as pet name, species, breed, date of birth, owner information, and primary medical history.
- Vets and admins can view and search the complete list of patient profiles.

Core Medical Record Entry:

- Authorized veterinary staff can create new medical records for a patient, including details like visit date, reason for visit, symptoms observed, and initial diagnosis.
- Basic clinical notes can be added and saved to the patient's permanent history.

Medical History Tracking:

- The system maintains a chronological log of all medical records for each patient, providing a clear overview of past visits and treatments.
- Users can click on a patient to view their full medical history in a single, scrollable interface.

3.3.3 Work in Progress (Doing)

Development is currently focused on enhancing the depth and clinical utility of the medical records. The following features are under active development:

Treatment and Prescription Management:

- The functionality to add specific treatments administered and medications prescribed during a visit is being integrated into the medical record form.
- This includes fields for medication name, dosage, frequency, and duration.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Digital Document Attachment:

- Implementation is underway to allow vets to attach digital files (such as lab reports, X-ray images, or vaccination certificates) directly to a medical record.
- The system will store and display links to these documents within the patient's history.

Advanced Search & Filtering:

- Development of an advanced search feature is ongoing. This will allow staff to filter patients or records based on criteria like diagnosis, medication prescribed, or date range, moving beyond simple name-based search.

3.3.4 Work to be Completed (To-Do)

The following advanced features are planned for future development cycles to create a comprehensive medical management system:

Billing Integration:

- Link medical procedures and prescriptions directly to the billing module to automatically generate invoices for consultations, treatments, and medications.

Vaccination & Reminder System:

- Implement a dedicated section to track vaccination history and set up automated reminders for upcoming booster shots or annual check-ups for patients.

Role-Based Access Control (RBAC) for Records:

- Define and implement stricter access permissions, ensuring that sensitive medical records can only be viewed or edited by authorized veterinary staff, not general administrative users.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Reporting and Analytics:

- Develop a reporting dashboard to analyze common diagnoses, frequently prescribed medications, and patient visit statistics to support clinic management decisions.

3.3.5 queries used

Medical Record Creation with Auto-User Registration

```
const { ownerEmail, ...recordData } = req.body;

let owner = await User.findOne({ email: ownerEmail });
if (!owner) {
  owner = await User.create({
    name: ownerEmail.split('@')[0],
    email: ownerEmail,
    role: 'owner',
    password: 'temporary'
  });
}
```

JWT Authentication Middleware

```
const token = req.header('Authorization')?.replace('Bearer ', '');
const decoded = jwt.verify(token, process.env.JWT_SECRET);
const user = await User.findById(decoded.id).select('-password');
req.user = user;
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Advanced Search with Pagination

```
if (req.query.search) {  
  query.$text = { $search: req.query.search };  
}  
  
const [records, total] = await Promise.all([  
  MedicalRecord.find(query)  
    .populate('owner', 'name email phone')  
    .sort({ visitDate: -1 })  
    .skip(skip)  
    .limit(limit),  
  MedicalRecord.countDocuments(query)  
]);
```

Secure File Upload Configuration

```
const upload = multer({  
  storage,  
  fileFilter,  
  limits: {  
    fileSize: 10 * 1024 * 1024,  
    files: 5  
  }  
});
```

Database Queries & Operations

Medical Record Aggregation for Statistics:

```
recordsByPetType = await MedicalRecord.aggregate([
  { $match: query },
  { $group: { _id: '$petType', count: { $sum: 1 } } },
  { $sort: { count: -1 } }
])
```

User Authentication Query:

```
const user = await User.findOne({ email }).select('+password');
const isMatch = await user.comparePassword(password);
```

Search Algorithm

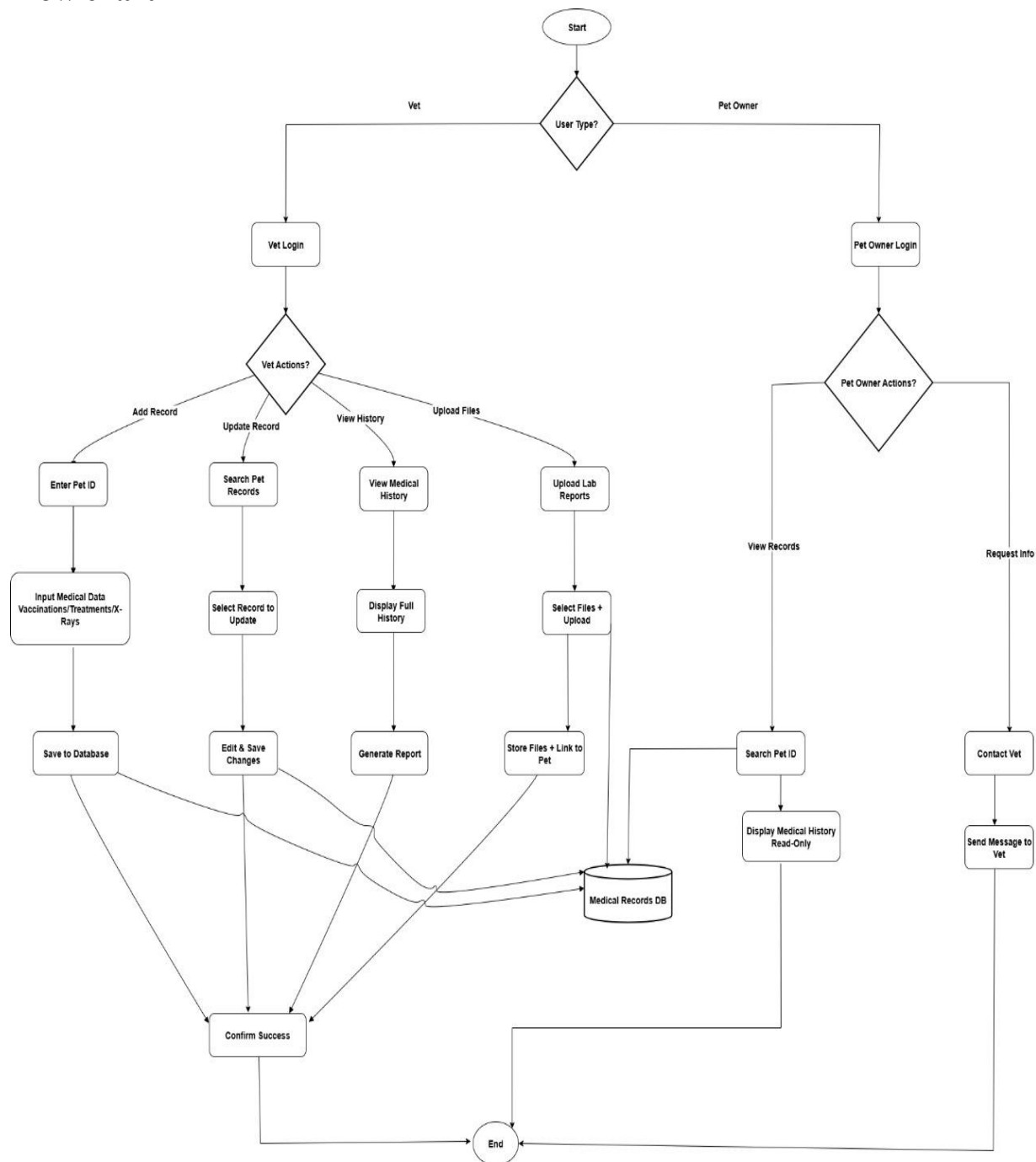
```
if (req.query.search) {
  query.$text = { $search: req.query.search };
}
```

Password Hashing Algorithm

```
userSchema.pre('save', async function(next) {
  const salt = await bcrypt.genSalt(12);
  this.password = await bcrypt.hash(this.password, salt);
});
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Flow chart



B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.4 - Rathnayake W P D D W (IT23413474)

(Pet Product Management)

3.4.1 Completion Level:

Current Status: **80%** Completed

3.4.2 Completed:

- **Backend & Database Operations:**

- Complete Pet Product CRUD Operations (Create, Read, Update, Delete products)
- RESTful API endpoints for all product operations
- Server-side validation and error handling
- Secure database updates with proper data validation

- **Frontend Implementation:**

- Complete Admin product management dashboard (frontend) with all navigations
- Product Dashboard with statistics overview and quick actions
- Comprehensive product listing with data table interface
- Designed and implemented user-friendly and responsive vertical form for product management (Create, Update)
- Integrated with existing admin sidebar navigation

- **Data Validation:**

- Added validation for required fields, including name, price, stock quantity > 0
- Client-side form validation with error messaging
- Stock level indicators with color-coded status system
- Proper error handling for API failures

3.4.3 To Be Completed:

- **Add to Cart Function:** Implementing cart management system with quantity selection and stock validation.
- Available Product list **download as a PDF.**
- **Product View Page:** Developing detailed product view interface with product images, descriptions, and specifications.
- **Navigation Integration:** Setting up seamless navigation from cart to payment module (integrated with payment function)
- **UI Polishing:** Improving responsive design, loading states, error messages, and overall user experience.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.3.4 Algorithms and Queries used:

- **Create New Product**

```
// Create Product
export const createProduct = async (req , res) => {
  const product = req.body; // user will send this data
  console.log("Received product data:", product);

  if(!product.name || !product.price || !product.image || !product.description || product.stock === undefined) {
    return res.status(400).json({ success: false, message: "All required fields must be provided (name, price, description, stock, image)" });
  }

  // Validate data types
  if(typeof product.price !== 'number' || product.price <= 0) {
    return res.status(400).json({ success: false, message: "Price must be a positive number" });
  }

  if(typeof product.stock !== 'number' || product.stock < 0) {
    return res.status(400).json({ success: false, message: "Stock must be a non-negative number" });
  }

  // Set default category if not provided
  if(!product.category || product.category.trim() === '') {
    product.category = "Other";
  }

  const newProduct = new Product(product);

  try {
    await newProduct.save();
    console.log("Product created successfully:", newProduct);
    res.status(201).json({ success: true, data: newProduct });
  } catch (error) {
    console.error("Error in Create Product:", error.message);
    res.status(500).json({ success: false, message: "Server Error: " + error.message });
  }
};
```

- **View All Products**

```
// Get All Products
export const getProducts = async (req , res) => {
  try {
    const products = await Product.find({});
    res.status(200).json({ success: true, data: products });
  } catch (error) {
    console.error("Error in fetching products:", error.message);
    res.status(500).json({ success: false, message: "Server Error" });
  }
};
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

- **Update Product Using ID**

```
// Update Product
export const updateProduct = async (req , res) => {
  const { id } = req.params; // get product id from url
  // console.log("id: ", id); check if we are getting the id

  const product = req.body; // updated fields sent by client

  if(!mongoose.Types.ObjectId.isValid(id)) {
    return res.status(404).json({ success: false, message: "Invalid Product Id" });
  }

  try {
    const updateProduct = await Product.findByIdAndUpdate(id, product, { new: true });
    res.status(200).json({ success: true, data: updateProduct });
  } catch (error) {
    res.status(500).json({ success: false, message: "Server Error"})
  }
};
```

- **Delete Product Using ID**

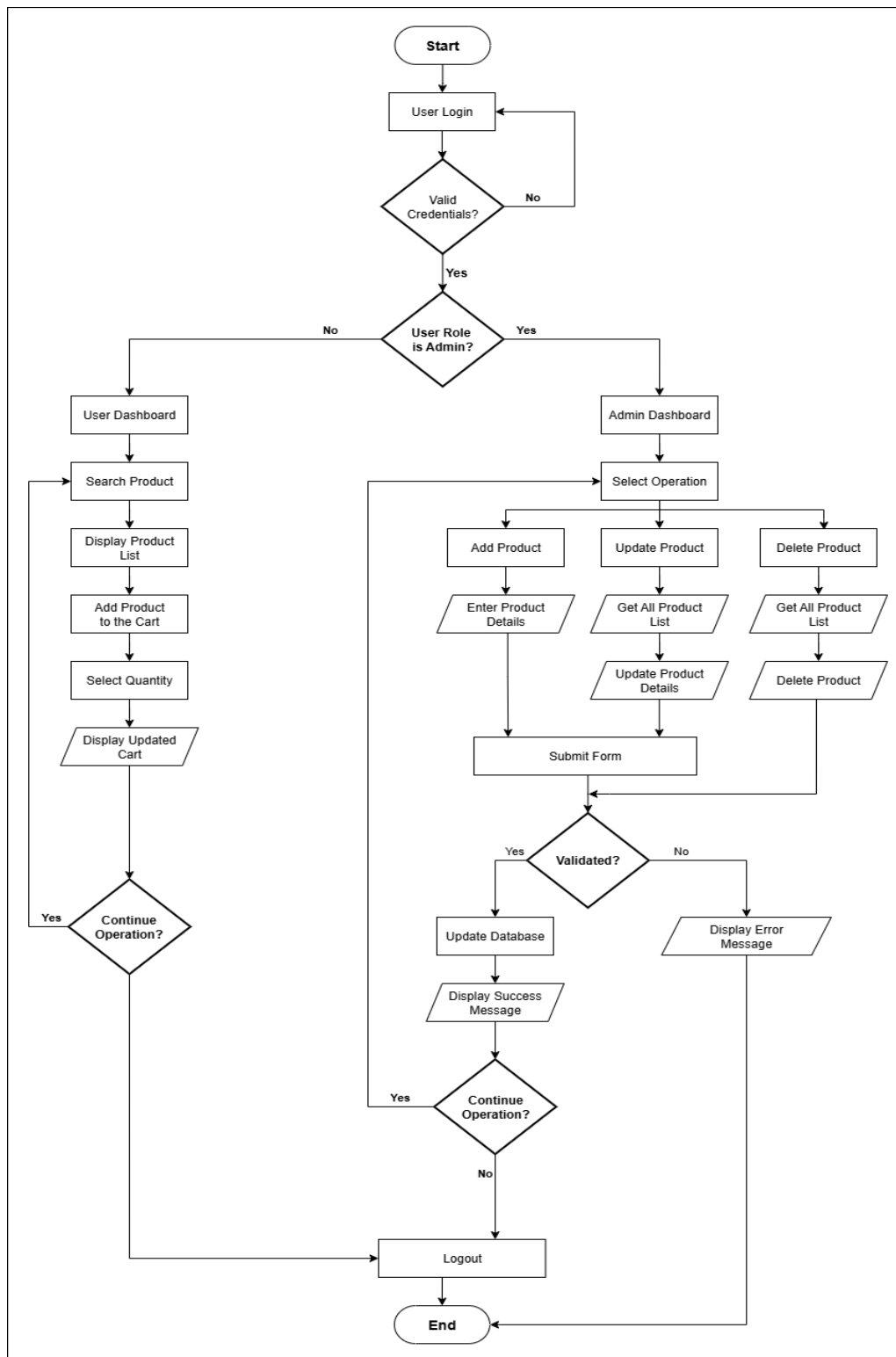
```
// Delete Product
export const deleteProduct = async (req , res) => {
  const {id} = req.params;
  // console.log("id: ", id); just to check if we are getting the id

  if(!mongoose.Types.ObjectId.isValid(id)) {
    return res.status(404).json({ success: false, message: "Invalid Product Id" });
  }

  try {
    await Product.findByIdAndDelete(id);
    res.status(200).json({ success: true, message: "Product deleted successfully" });
  } catch (error) {
    console.log("Error in Delete Product:", error.message);
    res.status(500).json({ success: false, message: "Server Error" });
  }
};
```


B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.4.5 Flow Chart: (Add to Cart Process and Product Management)



B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.5 - Abeysinghe A M B N (IT23631274)

(Payment Management)

1) The module working on (scope & responsibilities)

A) Payment Subsystem (Stripe)

- **Frontend:** PaymentPage.jsx
 - Add card (Stripe Elements), Edit card (holder/expiry), Delete card (confirm), Review & select saved card, Pay + 3DS handling, Success screen.
 - Summary panel (amount, currency, purpose, reference), trust/branding image slider.
- **Backend:** server.js endpoints
 - POST /api/create-setup-intent (save card)
 - GET /api/payment-methods (list + mirror into Mongo)
 - PATCH /api/payment-method/:pmId (update name/expiry)
 - DELETE /api/payment-method/:pmId (detach + delete mirror)
 - POST /api/create-payment-intent (charge + log Tx in Mongo)
 - Admin reporting for transactions (filters + bulk delete—already working).

B) Delivery Address Subsystem (Pet Mart)

- **Frontend:** DeliveryPage.jsx
 - Card-style address list with **radio select/unselect**, **Add/Edit modal**, **Delete with confirmation**, **client-side validations**, **limit to 3 addresses** (button hides + message).
 - Right-side summary + image slider aligned with payment page layout.
 - “Use this delivery address” navigates to Payment Review (passes context).
- **Backend:**
 - Model: models/Address.js (Mongoose).
 - Routes: routes/addresses.js (CRUD with per-user cap ≤ 3).
 - Admin view: routes/adminAddresses.js + src/AdminAddresses.jsx (read-only table with name/phone/address search, province filter).

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.5.3 Completion level of features

Feature (owned)	Status	Notes (what the reader needs to know)
Save card with Stripe SetupIntent	Done	<ul style="list-style-type: none"> User enters card on Add Card form (Stripe Elements). Server creates SetupIntent; client confirms it and saves the card. We store only brand/last4/expiry/name (no card numbers).
List + select/unselect saved card	Done	<ul style="list-style-type: none"> Loads saved cards and shows them in a list. Click radio to select; click again to unselect. Expiry shown as MM/YYYY for clarity
Edit card name/expiry	Done	<ul style="list-style-type: none"> Open Edit from card list; update name or expiry. Server updates the card in Stripe and mirrors changes in DB. <p>Inline messages show any invalid inputs.</p>
Delete card (confirm)	Done	<ul style="list-style-type: none"> Server creates and confirms the payment using the selected card. If bank asks for 3-D Secure, app shows a quick verification screen.

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

		On success, you see Payment Successful and we log it in DB.
Tx logging to Mongo + Admin Tx filters	Done	<ul style="list-style-type: none"> • Every successful payment is saved with amount/status/source/ref. • Admin page can search and filter these records for evidence.
Delivery: Add/Edit/Delete/Select	Done	<ul style="list-style-type: none"> • Manage addresses in a clean list with radio selection. • Add/Edit opens a form; Delete asks for confirmation. <p>‘Use this address’ sends you to Payment review.</p>
Delivery: Validations	Done	<ul style="list-style-type: none"> • Required: first name, last name, phone, address line 1, city, province. • Names: letters only; phone: exactly 10 digits; postal: numbers only. <p>Province picked from the 9 Sri Lankan provinces.</p>
Delivery: Max 3 addresses	Done	<ul style="list-style-type: none"> • User can save up to three addresses. <p>After three, add button hides and a message explains the limit.</p>
Flow branching (Hospital vs. Mart)	Done	<ul style="list-style-type: none"> • Hospital payments go directly to Payment review (no delivery). • Pet Mart purchases go Delivery → Payment. <p>Both pages share the right-hand summary and image slider.</p>

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

3.5.4 Work Not Completed Yet

- Delivery Detail Page – The feature has not yet been linked to the order workflow.
- Payment Page – The payment page has not been fully implemented or connected to the checkout flow

3.5.5 Queries used

A) User-facing Delivery Address CRUD

POST /api/addresses – Create (with ≤ 3 capacity)

```
const count = await Address.countDocuments({ userId });
if (count >= 3) {
  return res.status(409).json({
    error: "MAX_LIMIT_REACHED",
    message: "You can only add up to 3 delivery addresses."
  });
}

const doc = await Address.create({
  userId,
  firstName: req.body.firstName.trim(),
  lastName: req.body.lastName.trim(),
  phone: String(req.body.phone).replace(/\D/g, ""),
  line1: req.body.line1.trim(),
  line2: String(req.body.line2 || "").trim(),
  city: req.body.city.trim(),
  state: req.body.state.trim(),
  postalCode: String(req.body.postalCode || "").replace(/\D/g, ""),
  country: String(req.body.country || "Sri Lanka").trim(),
});
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

GET /api/addresses – Read

```
router.get("/", async (req, res) => {
  try {
    const userId = String(req.query.userId || "guest");
    const docs = await Address.find({ userId }).sort({ createdAt: 1 });
    res.json(docs);
  } catch (e) {
    res.status(500).json({ error: "SERVER_ERROR", message: e.message });
  }
});
```

PATCH /api/addresses/:id – Update an address

```
const doc = await Address.findOneAndUpdate(
  { _id: id, userId },
  {
    $set: {
      firstName: req.body.firstName.trim(),
      lastName: req.body.lastName.trim(),
      phone: String(req.body.phone).replace(/\D/g, ""),
      line1: req.body.line1.trim(),
      line2: String(req.body.line2 || "").trim(),
      city: req.body.city.trim(),
      state: req.body.state.trim(),
      postalCode: String(req.body.postalCode || "").replace(/\D/g, ""),
      country: String(req.body.country || "Sri Lanka").trim(),
    }
  },
  { new: true }
);
```

DELETE /api/addresses/:id – Delete an address

```
const doc = await Address.findOneAndDelete({ _id: id, userId });
```

B) Cards mirror, Transactions

GET /api/payment-methods – Mirror Stripe cards into Mongo

```
await Promise.all(
  cards.map((c) => Card.updateOne({ pmId: c.pmId }, { $set: c }, { upsert: true }))
);
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

PATCH /api/payment-method/:pmId – Mirror edited card

```
await Card.updateOne(  
  { pmId: pm.id },  
  {  
    $set: {  
      billing_name: pm.billing_details?.name || null,  
      exp_month: pm.card?.exp_month,  
      exp_year: pm.card?.exp_year,  
      brand: pm.card?.brand,  
      last4: pm.card?.last4,  
      stripe_customer: pm.customer || null,  
    },  
  },  
  { upsert: true }  
);
```

DELETE /api/payment-method/:pmId – Remove card mirror

```
await Card.deleteOne({ pmId: req.params.pmId });
```

POST /api/create-payment-intent – Log/Upsert transaction

```
await Tx.updateOne(  
  { piId: pi.id },  
  {  
    $set: {  
      amount: pi.amount,  
      currency: pi.currency,  
      status: pi.status,  
      source,  
      ref_id,  
      description: pi.description || description || null,  
    },  
  },  
  { upsert: true }  
);
```

B.Sc. (Hons) in Information Technology
IT2080 - IT Project
Activity 02 - Requirements Engineering Activity

Flow Chart

