



INFORMATION TECHNOLOGY PROJECT (IT2080)

Activity 04_ ITP25_B7.2_168

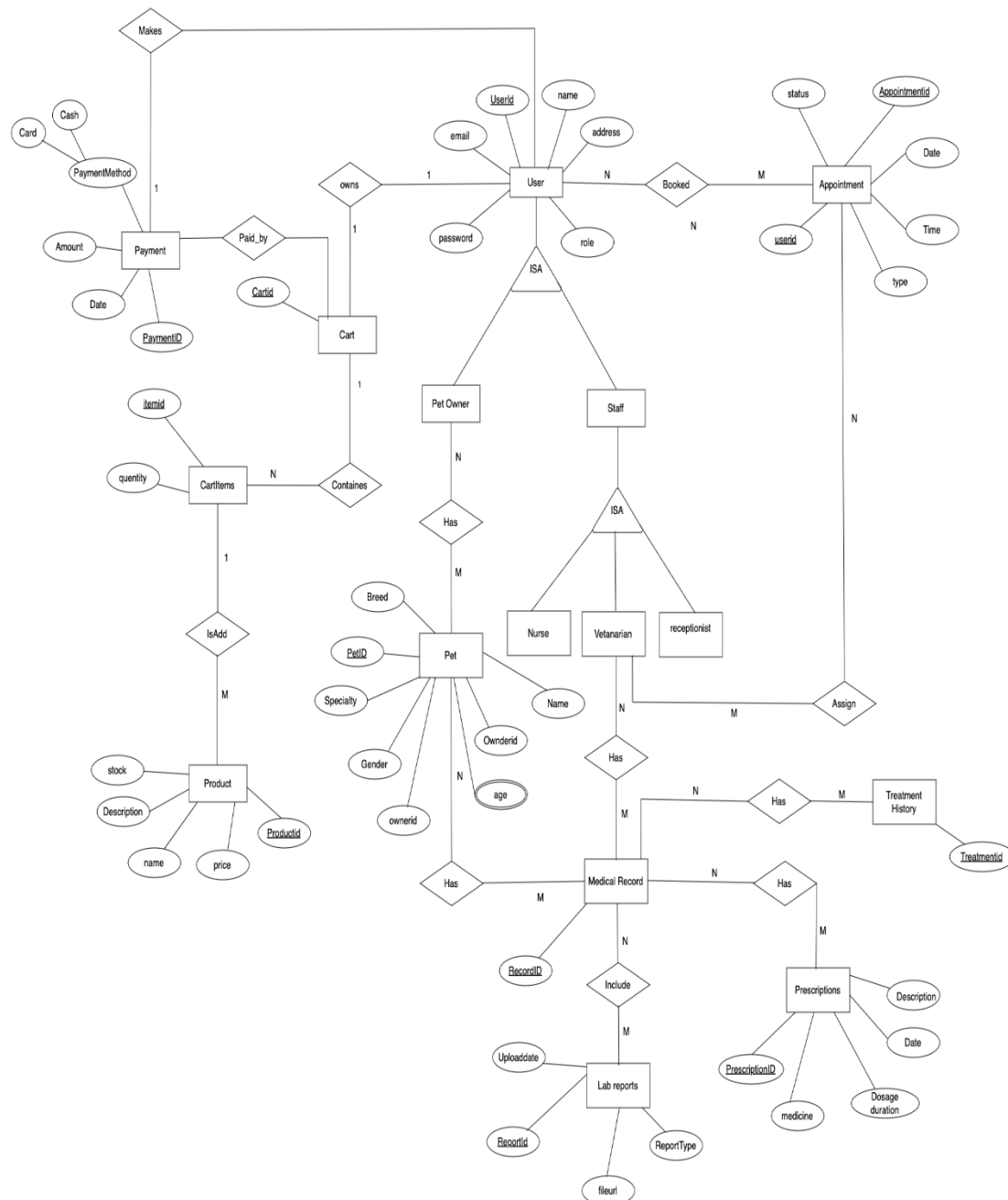
**Project Title: Veterinary Management System
(PetIQ.lk)**

Group: B7.2_168

Group Member 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	IT23413474	Rathnayake W. P. D.D. W	it23413474@my.sliit.lk	0702979383
3	IT23631106	Kavindu J. M. R	it23631106@my.sliit.lk	0703274142
4	IT23631274	Abeysinghe A. M. B.N	it23631274@my.sliit.lk	0761260885
5	IT23594722	Wanasinghe W. M.D.T	it23594722@my.sliit.lk	0774376338

1.Design a comprehensive ER diagram to represent the data model, capturing all entities, attributes, and relationships.



2. Normalize the database schema to eliminate redundancy, improve data integrity, and ensure optimal performance.

```
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);
```

```
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 }
});
```

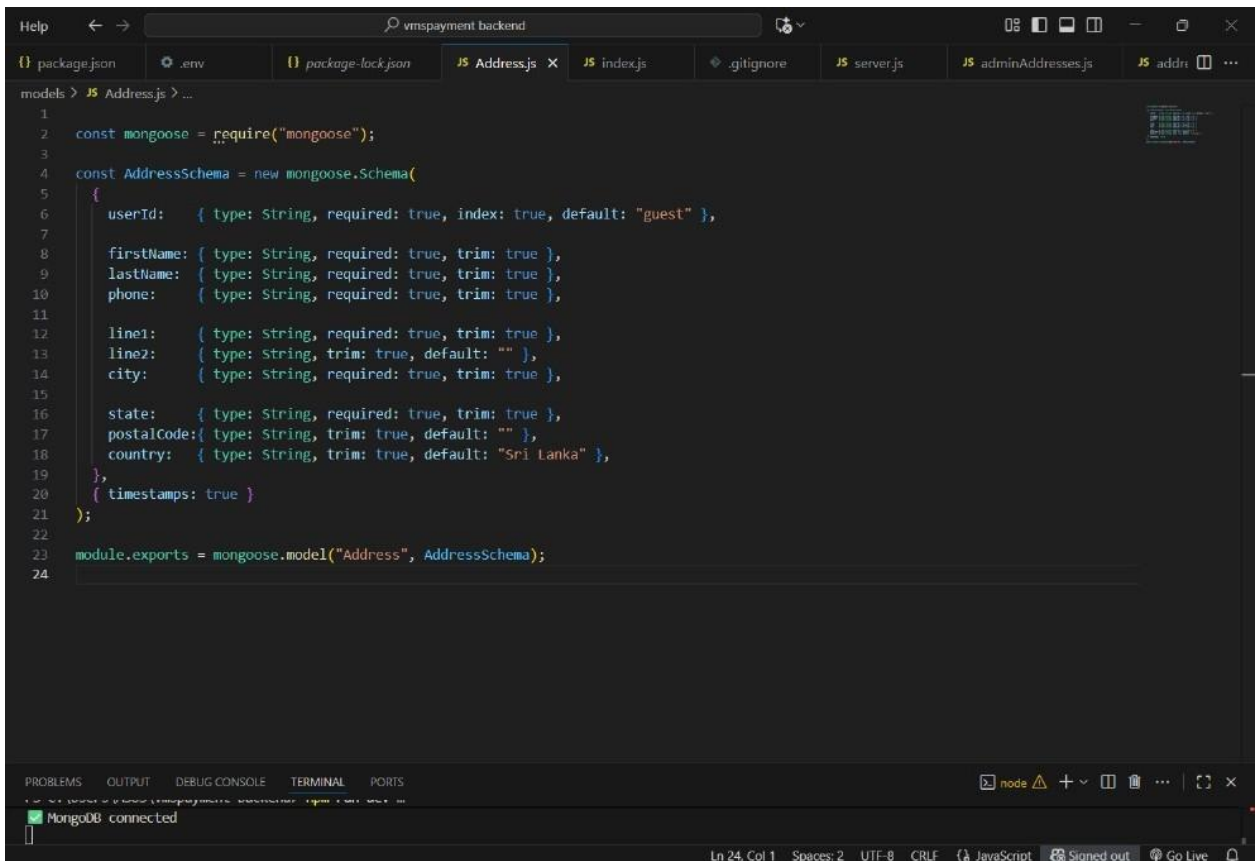
```
import mongoose from "mongoose";
import bcrypt from "bcryptjs";

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 }
});
```

```

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', 'Horse', 'Fish', 'Reptile', 'Amphibian', 'Insect', 'Mammal', 'Bird', 'Cat', 'Dog', 'Fish', 'Horse', 'Insect', 'Mammal', 'Reptile', 'Amphibian']},
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,uploadDate: { type: Date, default: Date.now}}],
20   status: { type: String,enum: ['active', 'completed', 'archived'],default: 'active'}
21 }, {
22   timestamps: true
23 });
24
25 // Indexes for better performance

```



```

Help  ← →  vmspayment backend
package.json  .env  package-lock.json  JS Address.js  JS index.js  .gitignore  JS server.js  JS adminAddresses.js  JS address.js  ...
models > JS Address.js > ...
1  const mongoose = require("mongoose");
2
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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

node + - [] ... [] ×

MongoDB connected

Ln 24, Col 1 Spaces: 2 UTF-8 CRLF JavaScript Signed out Go Live

```

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         pild: { 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

```

3. Build the database using the chosen technology, ensuring adherence to the designed schema and incorporating all necessary constraints, indexes, and relationships.

```

server > .env
1 MONGO_URI = mongodb+srv://dayanwishwanathr_db_user:RtYVrZDHef1CUWuv@cluster0.23zknzn.mongodb.net/petipDB?retryWrites=true&w=majority&appName=Cluster
2
3 PORT=5000

```

```

JS db.js
server > config > JS db.js > ...
1 import mongoose from 'mongoose';
2
3 export const connectDB = async () => {
4   try {
5     const conn = await mongoose.connect(process.env.MONGO_URI);
6     console.log(`MongoDB Connected: ${conn.connection.host}`);
7   } catch (error) {
8     console.error(`Error: ${error.message}`);
9     process.exit(1); // process code 1 means exit with failure, 0 means success
10  }
11 }

```


JS Admin.js > ...

```
1 const mongoose = require("mongoose");
2
3 const AdminSchema = new mongoose.Schema({
4   firstname: { type: String, required: true },
5   lastname: { type: String, required: true },
6   adminNIC: { type: String, unique: true, required: true },
7   password: { type: String, required: true } // hash this before saving
8 }, { timestamps: true });
9
10 module.exports = mongoose.model("Admin", AdminSchema);
11
```

JS AddressModel.js X

server > Model > JS AddressModel.js > ...

```
1 const mongoose = require("mongoose");
2
3 const addressSchema = new mongoose.Schema({
4   userId: { type: String, required: true, index: true, default: "guest" },
5   firstName: { type: String, required: true, trim: true },
6   lastName: { type: String, required: true, trim: true },
7   phone: { type: String, required: true, trim: true },
8   line1: { type: String, required: true, trim: true },
9   line2: { type: String, trim: true, default: "" },
10  city: { type: String, required: true, trim: true },
11  state: { type: String, required: true, trim: true },
12  postalCode: { type: String, trim: true, default: "" },
13  country: { type: String, trim: true, default: "Sri Lanka" },
14 }, { timestamps: true });
15
16 module.exports = mongoose.model("Address", addressSchema);
17
```

JS AppointmentModel.js X

server > Model > JS AppointmentModel.js > ...

```
1 import mongoose from "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 export default mongoose.model("Appointment", appointmentSchema);
```

JS Employees.js X

server > Model > JS Employees.js > ...

```
1  const mongoose = require("mongoose");
2
3  const employeeSchema = new mongoose.Schema({
4    name: { type: String, required: true },
5    email: { type: String, required: true, unique: true },
6    password: { type: String, required: true },
7    avatarUrl: { type: String },
8  });
9
10 const Employees = mongoose.model("Employees", employeeSchema);
11
12 module.exports = Employees;
```

JS MedicalRecord.js X

server > Model > JS MedicalRecord.js > ...

```
3  const medicalRecordSchema = new mongoose.Schema({
89    status: {
92      default: 'active'
93    }
94  }, {
95    timestamps: true
96  });
97
98 // Indexes for better performance
99 medicalRecordSchema.index({ owner: 1, visitDate: -1 });
100 medicalRecordSchema.index({ petName: 'text', symptoms: 'text', diagnosis: 'text' });
101 medicalRecordSchema.index({ veterinarian: 1 });
102
103 export default mongoose.model('MedicalRecord', medicalRecordSchema);
```

server > Model > JS PaymentModel.js > ...

```
1  import mongoose from "mongoose";
2  ▶ const cardSchema = new mongoose.Schema({
3      pmId: { type: String, unique: true, index: true },
4      brand: String,
5      last4: String,
6      exp_month: Number,
7      exp_year: Number,
8      billing_name: String,
9      stripe_customer: String,
10 } , { timestamps: true });
11 ▶ const txSchema = new mongoose.Schema({
12     piId: { type: String, unique: true, index: true },
13     amount: Number,
14     currency: String,
15     status: String,
16     source: String,
17     ref_id: String,
18     description: String,
19 } , { timestamps: true });
20 ▶ const Card = mongoose.model("Card", cardSchema);
21 ▶ export const Tx = mongoose.model("Tx", txSchema);
22 ▶ export default Card;
```



```
1 import mongoose from "mongoose";
2 const productSchema = new mongoose.Schema({
3   name :{
4     type: String,
5     required: true},
6   price:{
7     type: Number,
8     required: true},
9   description: {
10    type: String,
11    required: true },
12   category: {
13     type: String,
14     required: false,
15     default: "Other"},
16   stock: {
17     type: Number,
18     required: true,
19     min: 0 },
20   image: {
21     type: String,
22     required: true},
23 }, {
24   timestamps: true // createdAt, updatedAt
25 });
26 const Product = mongoose.model('Product', productSchema);
27 export default Product;
```