IT2080 – Information Technology Project (ITP) Year 2, Semester 2, 2025

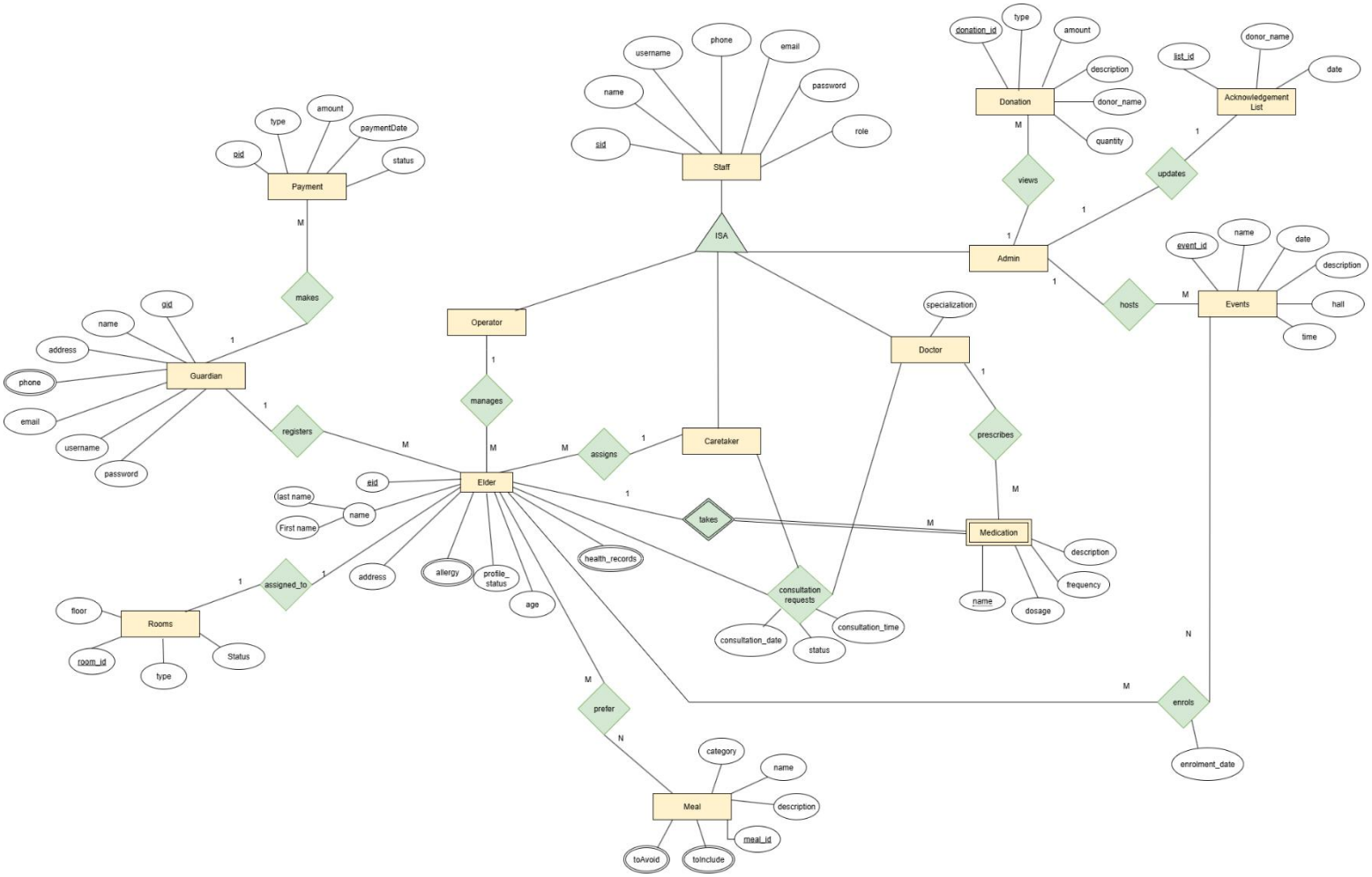
Elders Home Management System

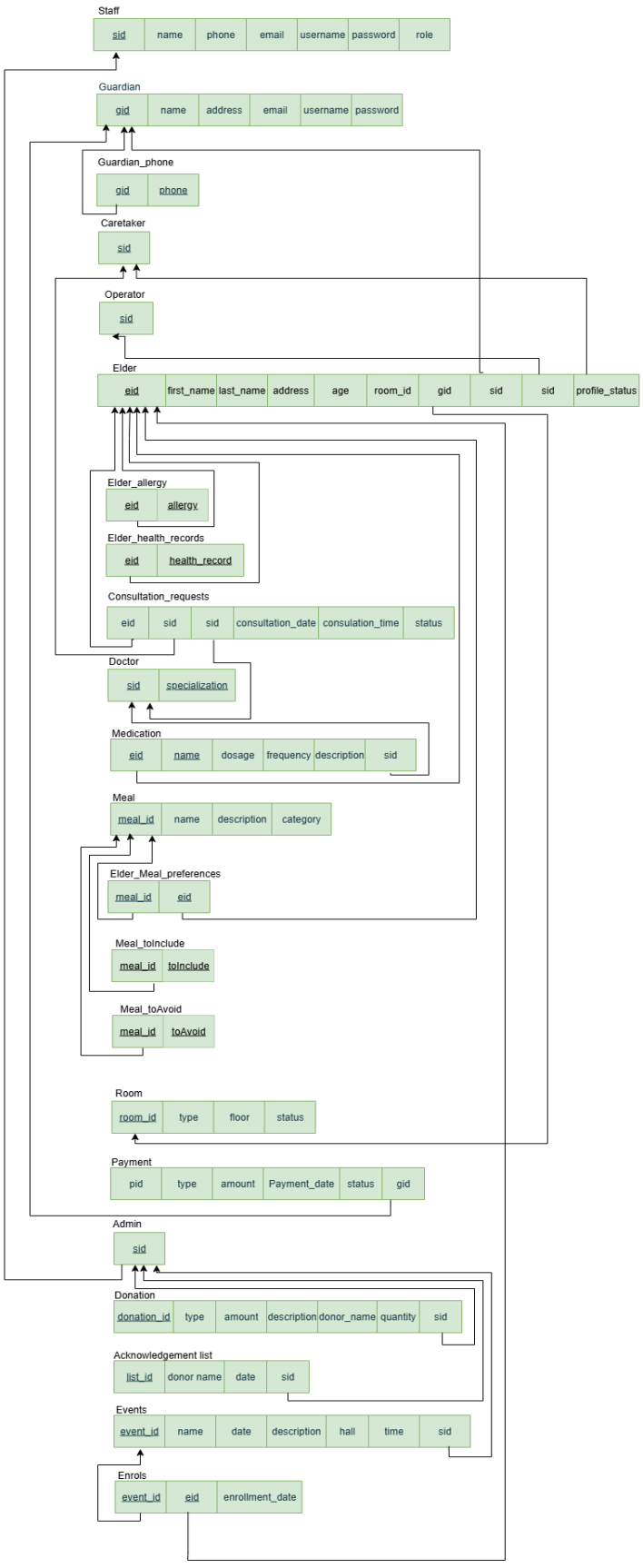
Activity 04 - Database Design & Development Activity

Group Number: 04

	Student Registration Number	Student Name	Student E-mail Address	Contact Number
1	IT23616042	L.D.B Madigasekara	it23616042@my.sliit.lk	070 153 6432
2	IT23547988	U.O.F De Run	it23547988@my.sliit.lk	071 191 1496
3	IT23838048	T.P Hansana	it23838048@my.sliit.lk	076 202 8980
4	IT23849792	K.L.T.D Sugathnath	it23849792@my.sliit.lk	070 197 3380
5	IT23850064	N.D.K.G.D.K.B Manawasinghe	it23850064@my.sliit.lk	075 560 8997

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.

•

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

1.Elder Schema

backend > models > JS Elder.js > ...

```
const elderSchema = new Schema(
  {

    eid: { type: String, trim: true, unique: true, sparse: true },

    // Identity
    firstName: { type: String, required: true, trim: true, maxlength: 80 },
    lastName: { type: String, required: true, trim: true, maxlength: 80 },
    age: { type: Number, min: 0, max: 130 },

    profile_status: {
      type: String,
      enum: ["pending", "active", "inactive", "archived"],
      default: "active",
      index: true,
    },
    address: { type: addressSchema },
    allergy: { type: [String], default: [] },

    // Relationships
    guardian: { type: Schema.Types.ObjectId, ref: "Guardian", required: true, index: true },
    caretaker: { type: Schema.Types.ObjectId, ref: "Caretaker", index: true },
    operator: { type: Schema.Types.ObjectId, ref: "Operator", index: true },

    room: {
      type: Schema.Types.ObjectId,
      ref: "Room",
      unique: true,
      sparse: true, //allow elders without rooms
      index: true,
    },
    assignedAt: { type: Date }, // current assignment date
    // Health
    health_records: { type: [healthRecordSchema], default: [] },
  },
  { timestamps: true, toJSON: { virtuals: true }, toObject: { virtuals: true } }
);


// Fast lookups by name
elderSchema.index({ lastName: 1, firstName: 1 });

elderSchema.virtual("fullName").get(function () {
  return `${this.firstName} ${this.lastName}`;
});

export const Elder = mongoose.model("Elder", elderSchema);

```

```
import mongoose from "mongoose";

const { Schema } = mongoose;

/* ---------- Subdocs ---------- */
const addressSchema = new Schema(
  {
    line1: { type: String, trim: true },
    line2: { type: String, trim: true },
    city: { type: String, trim: true, index: true },
    district: { type: String, trim: true },
    postalCode: { type: String, trim: true },
  },
  { _id: false }
);

const healthRecordSchema = new Schema(
  {
    date: { type: Date, default: Date.now, index: true },
    notes: { type: String, trim: true, maxlength: 2000 },
    doctor: { type: Schema.Types.ObjectId, ref: "Doctor" }, // from ER
    vitals: {
      bp: { type: String, trim: true },
      pulse: { type: Number, min: 0 },
      temperatureC: { type: Number, min: 20, max: 45 },
    },
    files: { type: [String], default: [] }, // URLs
  },
  { _id: true, timestamps: true }
);

```

2.Meal Schema

backend > models > Meal.js > mealSchema > toAvoid

```
import mongoose from "mongoose";

const mealSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    description: {
        type: String,
        trim: true
    },
    category: {
        type: String,
        required: true,
        enum: ["breakfast", "lunch", "dinner", "snack", "other"]
    },
    toInclude: {
        type: [String],
        default: []
    },
    toAvoid: {
        type: [String],
        default: []
    }
});

// prevent from same item appearing on both avoid and include
mealSchema.pre("validate", function (next) {
    const setA = new Set(this.toInclude || []);
    const overlap = (this.toAvoid || []).find(x => setA.has(x));
    if (overlap) next(new Error(`'${overlap}' appears in both include and avoid.`));
    else next();
});

//to avoid duplicates
mealSchema.index({ name: 1, category: 1 }, { unique: true });

export const Meal = mongoose.model("Meal", mealSchema);

```

3.Elder_Meal_preferences

backend > models > JS ElderMealPreference.js > ...

```
import mongoose from "mongoose";

const elderMealPreferenceSchema = new mongoose.Schema({
    elder: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Elder",
        required: true
    },
    meal: { type: mongoose.Schema.Types.ObjectId, ref: "Meal", required: true },
    preference: {
        type: String,
        enum: ["like", "neutral", "avoid"],
        default: "like"
    }
});

// Prevent duplicates
elderMealPreferenceSchema.index({ elder: 1, meal: 1 }, { unique: true });

export const ElderMealPreference = mongoose.model("ElderMealPreference",
    elderMealPreferenceSchema);

```

4.Room schema

backend > models > JS Room.js > ...

```
import mongoose from "mongoose";

const roomSchema = new mongoose.Schema(
    {
        room_id: {
            type: String,
            required: true,
            trim: true,
            unique: true,
        },
        floor: {
            type: String,
            required: true,
            enum: ["Ground", "1", "2"],
            index: true,
        },
        type: {
            type: String,
            required: true,
            enum: ["AC", "Non-AC"],
            index: true,
        },
        status: {
            type: String,
            enum: ["available", "occupied", "maintenance", "reserved"],
            default: "available",
            index: true,
        },
    },
    { timestamps: true }
);

// can be used for search
roomSchema.index({ floor: 1, type: 1 });

export const Room = mongoose.model("Room", roomSchema);

```

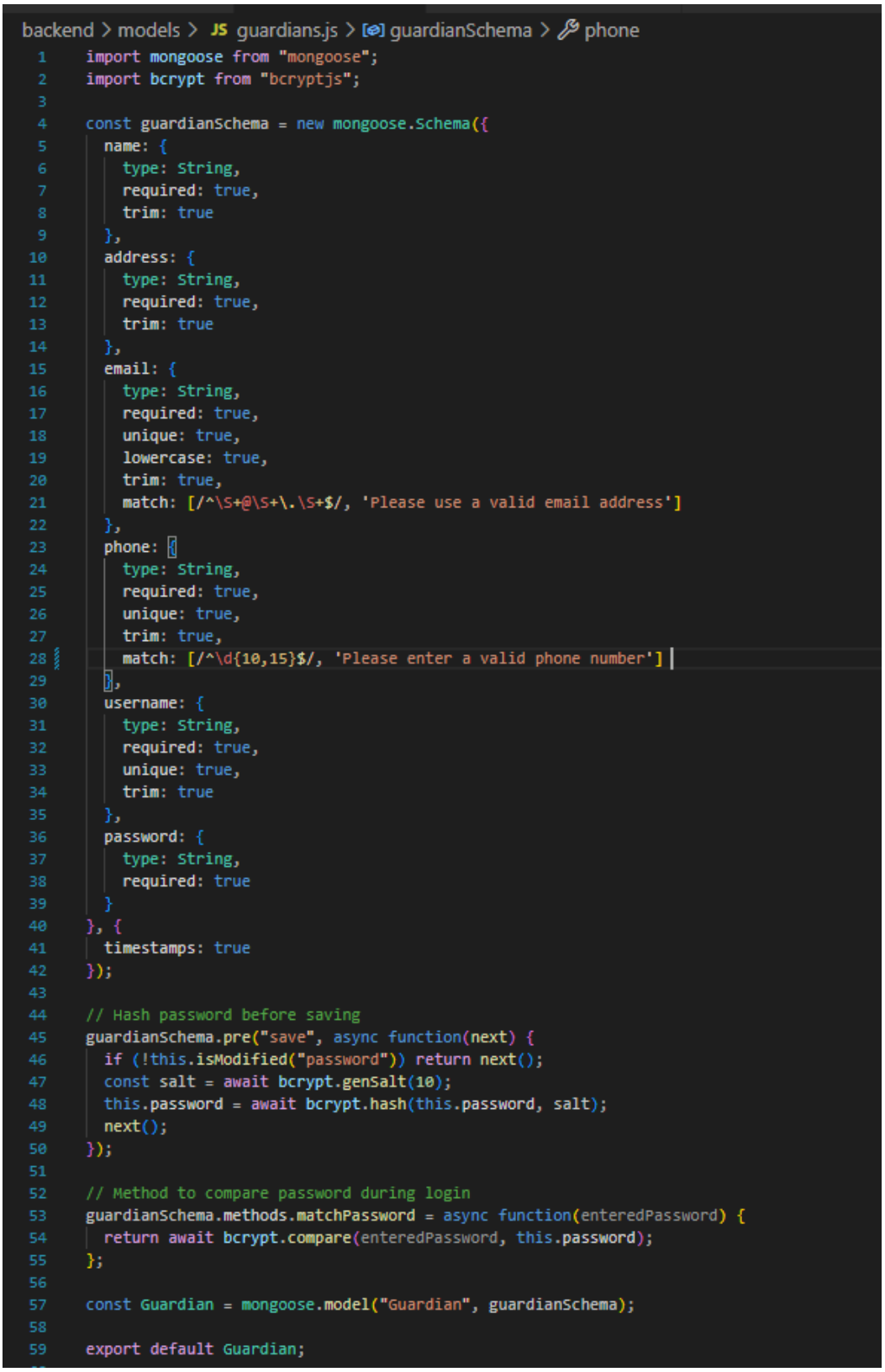
5. Event

```
end > models > JS eventModel.js > ...
import mongoose from "mongoose";

const eventSchema = new mongoose.Schema({

  title: {
    type: String,
    required: true,
    trim: true,     // removes extra spaces
  },
  description: {
    type: String,
    required: true,
  },
  start_time: {
    type: Date,
    required: true,
  },
  end_time: {
    type: Date,
    required: true,
    validate: {
      validator: function (v) {
        return v > this.start_time; // end_time must be after start_time
      },
      message: "End time must be after start time",
    },
  },
  location: {
    type: String,
    required: true,
  },
}, {
  timestamps: true // adds createdAt & updatedAt automatically
});

const Event = mongoose.model('Event', eventSchema);

export default Event;
```

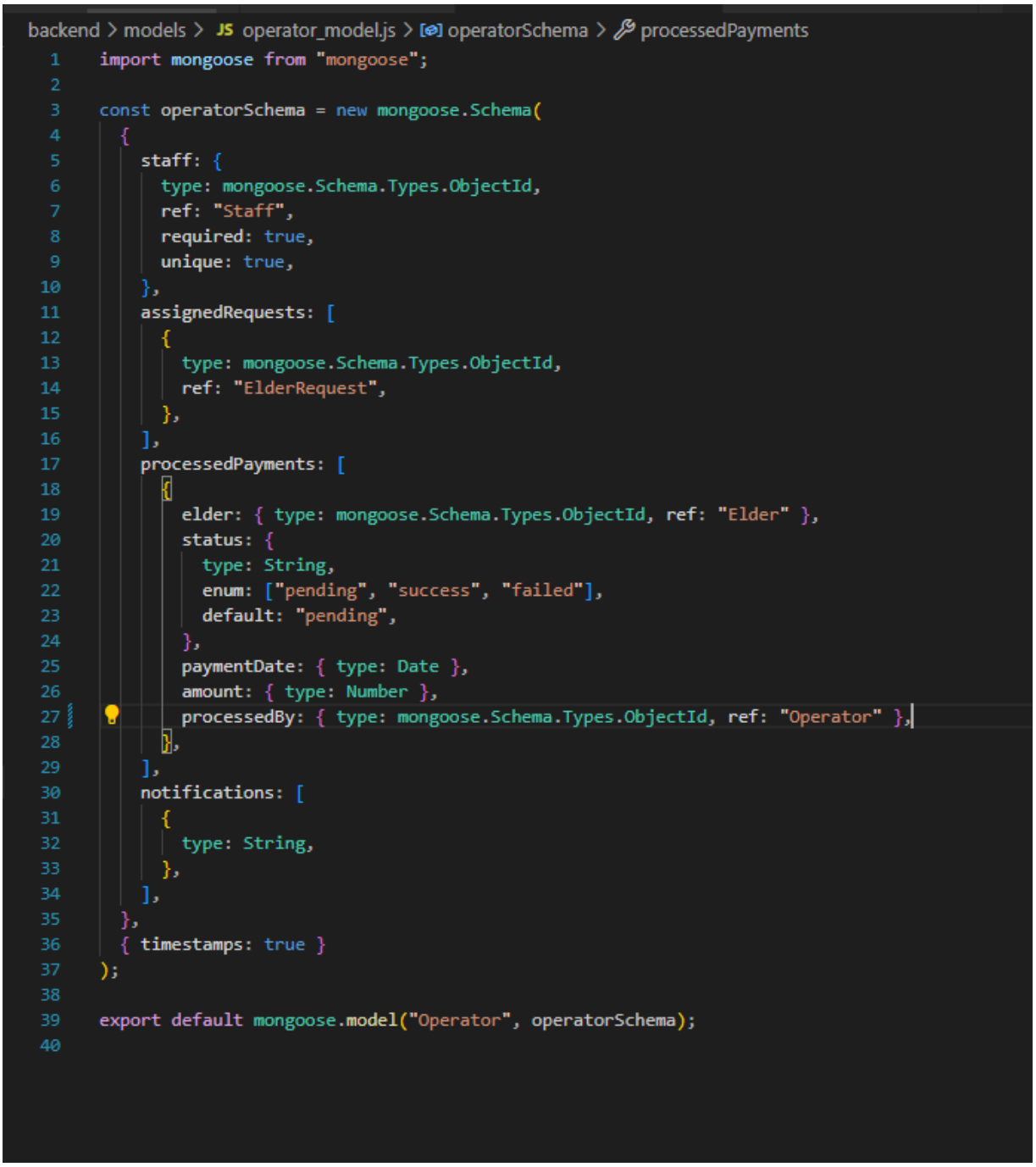
6 Donation Schema and Acknowledgement list schema

```
import mongoose from 'mongoose';

const donationSchema = new mongoose.Schema({
  type: { type: String, required: true },
  amount: { type: Number, required: true },
  description: { type: String },
  donor_name: { type: String, required: true },
  quantity: { type: Number },
  admin: { type: mongoose.Schema.Types.ObjectId, ref: 'Admin', required: true }
});

const acknowledgementSchema = new mongoose.Schema({
  donor_name: { type: String, required: true },
  date: { type: Date, default: Date.now },
  admin: { type: mongoose.Schema.Types.ObjectId, ref: 'Admin', required: true }
});

const Donation = mongoose.model('Donation', donationSchema);
const Acknowledgement = mongoose.model('Acknowledgement', acknowledgementSchema);

export { Donation, Acknowledgement };
```

7.Guardian Schema



8.Operator Schema



9.Payment Schema

backend > models > JS payment_model.js > ...

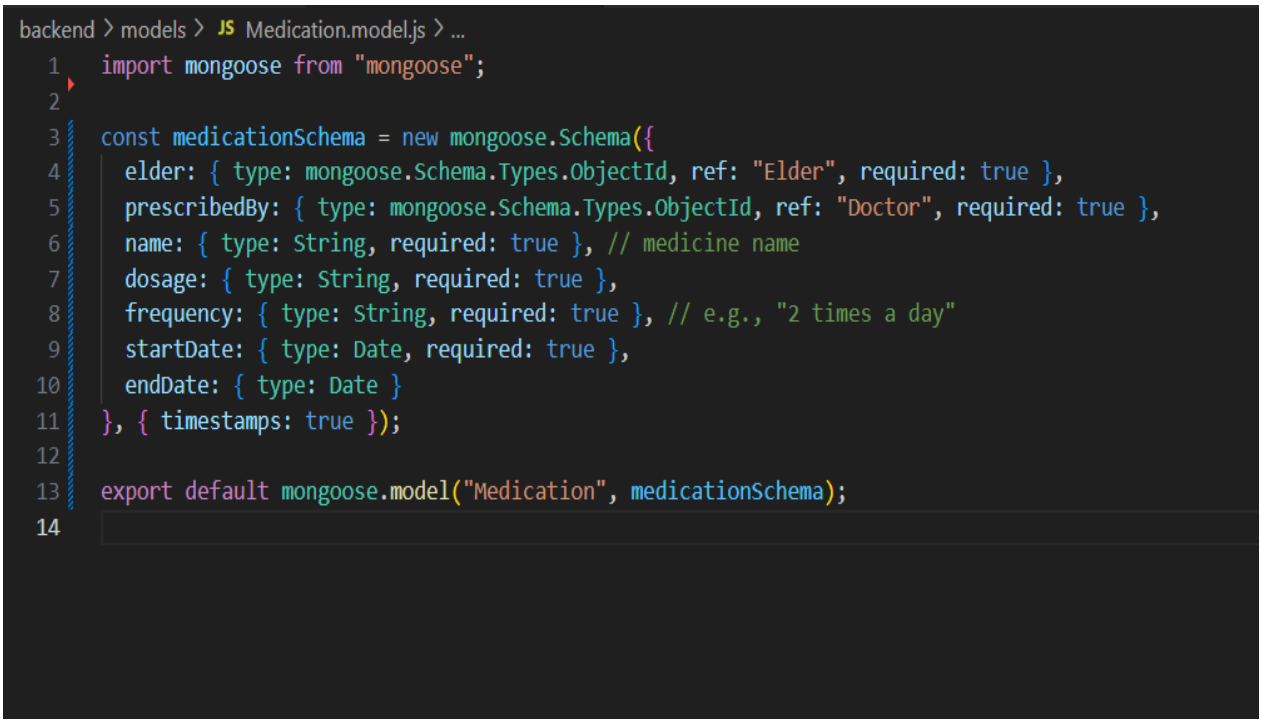
```
import mongoose from "mongoose";

export const PaymentStatus = {
  PENDING: "PENDING",
  SUCCESS: "SUCCESS",
  FAILED: "FAILED",
};

const paymentSchema = new mongoose.Schema(
  {
    elder: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Elder",
      required: true,
    },
    guardian: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Guardian",
      required: true,
    },
    amount: { type: Number, required: true },
    currency: { type: String, default: "LKR" },
    status: {
      type: String,
      enum: Object.values(PaymentStatus),
      default: PaymentStatus.PENDING,
    },
    mockCheckoutUrl: { type: String },
    reminderSentAt: {
      type: Date, // Tracks when reminder was sent
    },
  },
  { timestamps: true }
);

export default mongoose.model("Payment", paymentSchema);
```

10.Staff Schema

backend > models > JS staff_model.js > staffSchema > username

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

const staffSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    username: {
      type: String,
      required: true,
      unique: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    phone: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      enum: ["admin", "operator", "caretaker", "doctor"],
      required: true,
    },
  },
  { timestamps: true }
);

//password hashing

staffSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

//password matching
staffSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

export default mongoose.model("Staff", staffSchema);

```

11.Medication Schema



12.Consultaion Schema

13.Doctor Schema

backend > models > JS doctor_model.js > ...

```
import mongoose from "mongoose";

const doctorSchema = new mongoose.Schema(
  {
    staff: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Staff",
      required: true,
      unique: true, // Each doctor must have one staff account
    },
    specialization: {
      type: String,
      required: true,
      trim: true,
    },

    appointments: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Appointment",
    },
  },
  { timestamps: true }
);

export default mongoose.model("Doctor", doctorSchema);

```

14.Caretaker Schema

backend > models > caretaker_model.js > caretakerSchema > assignedElders

```
import mongoose from "mongoose";

const caretakerSchema = new mongoose.Schema(
  {
    staff: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Staff",
      required: true,
      unique: true,
    },
    assignedElders: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Elder",
      },
    ]
  },
  { timestamps: true }
);

export default mongoose.model("Caretaker", caretakerSchema);

```