

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

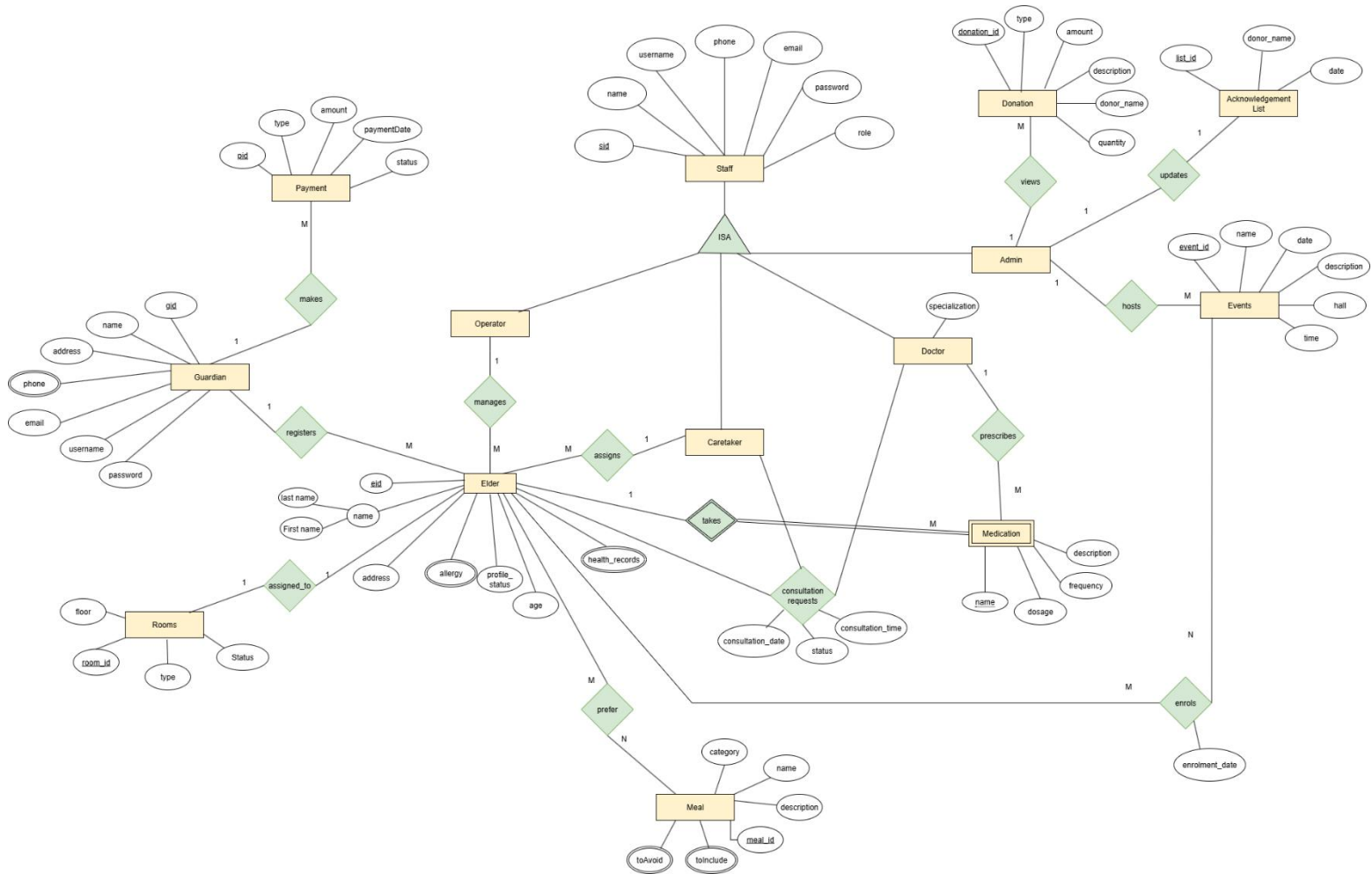


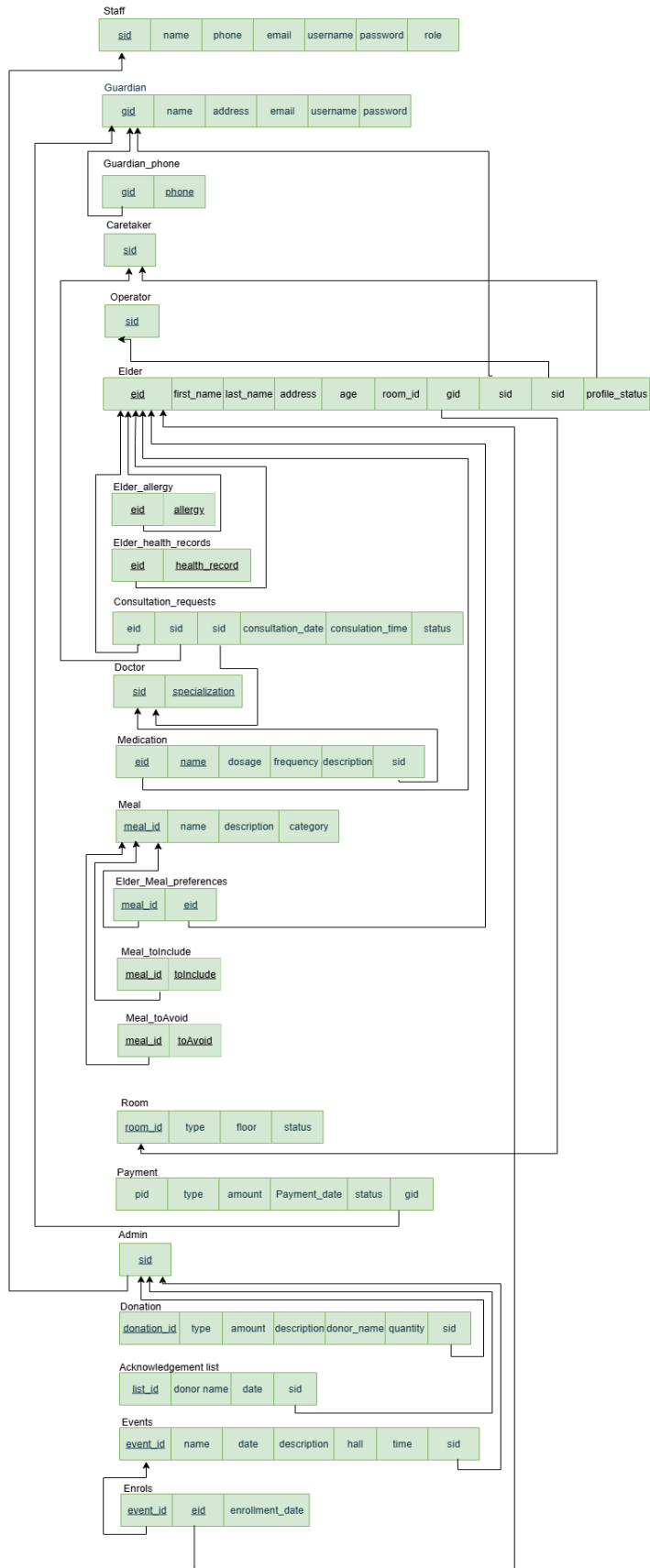
Elders Home Management System
Activity 04 - Database Design & Development
Activity

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 > ...
33 const elderSchema = new Schema(
34   {
35
36     eid: { type: String, trim: true, unique: true, sparse: true },
37
38     // Identity
39     firstName: { type: String, required: true, trim: true, maxlength: 80 },
40     lastName: { type: String, required: true, trim: true, maxlength: 80 },
41     age: { type: Number, min: 0, max: 130 },
42
43     profile_status: {
44       type: String,
45       enum: ["pending", "active", "inactive", "archived"],
46       default: "active",
47       index: true,
48     },
49     address: { type: addressSchema },
50     allergy: { type: [String], default: [] },
51
52     // Relationships
53     guardian: { type: Schema.Types.ObjectId, ref: "Guardian", required: true, index: true },
54     caretaker: { type: Schema.Types.ObjectId, ref: "Caretaker", index: true },
55     operator: { type: Schema.Types.ObjectId, ref: "Operator", index: true },
56
57     room: {
58       type: Schema.Types.ObjectId,
59       ref: "Room",
60       unique: true,
61       sparse: true, //allow elders without rooms
62       index: true,
63     },
64     assignedAt: { type: Date }, // current assignment date
65     // Health
66     health_records: { type: [healthRecordSchema], default: [] },
67   },
68   { timestamps: true, toJSON: { virtuals: true }, toObject: { virtuals: true } }
69 );
70
71
72 // Fast lookups by name
73 elderSchema.index({ lastName: 1, firstName: 1 });
74
75 elderSchema.virtual("fullName").get(function () {
76   return `${this.firstName} ${this.lastName}`;
77 });
78
79 export const Elder = mongoose.model("Elder", elderSchema);
80
```

```

1 import mongoose from "mongoose";
2
3 const { Schema } = mongoose;
4
5 /* ----- Subdocs ----- */
6 const addressSchema = new Schema(
7   {
8     line1: { type: String, trim: true },
9     line2: { type: String, trim: true },
10    city: { type: String, trim: true, index: true },
11    district: { type: String, trim: true },
12    postalCode: { type: String, trim: true },
13  },
14  { _id: false }
15 );
16
17 const healthRecordSchema = new Schema(
18   {
19     date: { type: Date, default: Date.now, index: true },
20     notes: { type: String, trim: true, maxlength: 2000 },
21     doctor: { type: Schema.Types.ObjectId, ref: "Doctor" }, // from ER
22     vitals: {
23       bp: { type: String, trim: true },
24       pulse: { type: Number, min: 0 },
25       temperatureC: { type: Number, min: 20, max: 45 },
26     },
27     files: { type: [String], default: [] }, // URLs
28   },
29   { _id: true, timestamps: true }
30 );
31

```

2.Meal Schema

```
backend > models > JS Mealjs > mealSchema > toAvoid
1  import mongoose from "mongoose";
2
3  const mealSchema = new mongoose.Schema({
4    name: {
5      type: String,
6      required: true,
7      trim: true
8    },
9    description: {
10     type: String,
11     trim: true
12   },
13   category: {
14     type: String,
15     required: true,
16     enum: ["breakfast", "lunch", "dinner", "snack", "other"]
17   },
18   toInclude: {
19     type: [String],
20     default: []
21   },
22   toAvoid: [
23     type: [String],
24     default: []
25   ]
26 });
27
28 // prevent from same item appearing on both avoid and include
29 mealSchema.pre("validate", function (next) {
30   const setA = new Set(this.toInclude || []);
31   const overlap = (this.toAvoid || []).find(x => setA.has(x));
32   if (overlap) next(new Error(`${overlap} appears in both include and avoid.`));
33   else next();
34 });
35
36 //to avoid duplicates
37 mealSchema.index({ name: 1, category: 1 }, { unique: true });
38
39 export const Meal = mongoose.model("Meal", mealSchema);
40
```

3.Elder_Meal_preferences

```
backend > models > JS ElderMealPreference.js > ...
1  import mongoose from "mongoose";
2
3  ∨ const elderMealPreferenceSchema = new mongoose.Schema({
4  ∨    elder: {
5      type: mongoose.Schema.Types.ObjectId,
6      ref: "Elder",
7      required: true
8    },
9    meal: { type: mongoose.Schema.Types.ObjectId, ref: "Meal", required: true },
10 ∨    preference: {
11      type: String,
12      enum: ["like", "neutral", "avoid"],
13      default: "like"
14    }
15  });
16
17  // Prevent duplicates
18  elderMealPreferenceSchema.index({ elder: 1, meal: 1 }, { unique: true });
19
20  export const ElderMealPreference = mongoose.model("ElderMealPreference",
21    elderMealPreferenceSchema);
22
```

4.Room schema

```
backend > models > JS Room.js > ...
1  import mongoose from "mongoose";
2
3  const roomSchema = new mongoose.Schema(
4    {
5      room_id: {
6        type: String,
7        required: true,
8        trim: true,
9        unique: true,
10     },
11     floor: {
12       type: String,
13       required: true,
14       enum: ["Ground", "1", "2"],
15       index: true,
16     },
17     type: {
18       type: String,
19       required: true,
20       enum: ["AC", "Non-AC"],
21       index: true,
22     },
23     status: {
24       type: String,
25       enum: ["available", "occupied", "maintenance", "reserved"],
26       default: "available",
27       index: true,
28     },
29   },
30   { timestamps: true }
31 );
32
33 // can be used for search
34 roomSchema.index({ floor: 1, type: 1 });
35
36 export const Room = mongoose.model("Room", roomSchema);
37
```


5. Event

```
end > models > js event\models > ...
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

```
backend > models > JS guardians.js > [🔗] guardianSchema > 📄 phone

1  import mongoose from "mongoose";
2  import bcrypt from "bcryptjs";
3
4  const guardianSchema = new mongoose.Schema({
5    name: {
6      type: String,
7      required: true,
8      trim: true
9    },
10   address: {
11     type: String,
12     required: true,
13     trim: true
14   },
15   email: {
16     type: String,
17     required: true,
18     unique: true,
19     lowercase: true,
20     trim: true,
21     match: [/^\S+@\S+\.\S+$/, 'Please use a valid email address']
22   },
23   phone: {
24     type: String,
25     required: true,
26     unique: true,
27     trim: true,
28     match: [/^\d{10,15}$/, 'Please enter a valid phone number']
29   },
30   username: {
31     type: String,
32     required: true,
33     unique: true,
34     trim: true
35   },
36   password: {
37     type: String,
38     required: true
39   }
40 }, {
41   timestamps: true
42 });
43
44 // Hash password before saving
45 guardianSchema.pre("save", async function(next) {
46   if (!this.isModified("password")) return next();
47   const salt = await bcrypt.genSalt(10);
48   this.password = await bcrypt.hash(this.password, salt);
49   next();
50 });
51
52 // Method to compare password during login
53 guardianSchema.methods.matchPassword = async function(enteredPassword) {
54   return await bcrypt.compare(enteredPassword, this.password);
55 };
56
57 const Guardian = mongoose.model("Guardian", guardianSchema);
58
59 export default Guardian;
```

8.Operator Schema

```
backend > models > JS operator_model.js > operatorSchema > processedPayments
1  import mongoose from "mongoose";
2
3  const operatorSchema = new mongoose.Schema(
4    {
5      staff: {
6        type: mongoose.Schema.Types.ObjectId,
7        ref: "Staff",
8        required: true,
9        unique: true,
10     },
11     assignedRequests: [
12       {
13         type: mongoose.Schema.Types.ObjectId,
14         ref: "ElderRequest",
15       },
16     ],
17     processedPayments: [
18       {
19         elder: { type: mongoose.Schema.Types.ObjectId, ref: "Elder" },
20         status: {
21           type: String,
22           enum: ["pending", "success", "failed"],
23           default: "pending",
24         },
25         paymentDate: { type: Date },
26         amount: { type: Number },
27         processedBy: { type: mongoose.Schema.Types.ObjectId, ref: "Operator" },
28       },
29     ],
30     notifications: [
31       {
32         type: String,
33       },
34     ],
35   },
36   { timestamps: true }
37 );
38
39 export default mongoose.model("Operator", operatorSchema);
40
```

9.Payment Schema

```
backend > models > JS payment_model.js > ...
1  import mongoose from "mongoose";
2
3  export const PaymentStatus = {
4    PENDING: "PENDING",
5    SUCCESS: "SUCCESS",
6    FAILED: "FAILED",
7  };
8
9  const paymentSchema = new mongoose.Schema(
10   {
11     elder: {
12       type: mongoose.Schema.Types.ObjectId,
13       ref: "Elder",
14       required: true,
15     },
16     guardian: {
17       type: mongoose.Schema.Types.ObjectId,
18       ref: "Guardian",
19       required: true,
20     },
21     amount: { type: Number, required: true },
22     currency: { type: String, default: "LKR" },
23     status: {
24       type: String,
25       enum: Object.values(PaymentStatus),
26       default: PaymentStatus.PENDING,
27     },
28     mockCheckoutUrl: { type: String },
29     reminderSentAt: {
30       type: Date, // Tracks when reminder was sent
31     },
32   },
33   { timestamps: true }
34 );
35
36 export default mongoose.model("Payment", paymentSchema);
37
```

10. Staff Schema

```
backend > models > JS staff_model.js > staffSchema > username
1  import mongoose from "mongoose";
2  import bcrypt from "bcryptjs";
3
4  const staffSchema = new mongoose.Schema(
5    {
6      name: {
7        type: String,
8        required: true,
9      },
10     username: {
11       type: String,
12       required: true,
13       unique: true,
14     },
15     email: {
16       type: String,
17       required: true,
18       unique: true,
19     },
20     phone: {
21       type: String,
22       required: true,
23     },
24     password: {
25       type: String,
26       required: true,
27     },
28     role: {
29       type: String,
30       enum: ["admin", "operator", "caretaker", "doctor"],
31       required: true,
32     },
33   },
34   { timestamps: true }
35 );
36
37 //password hashing
38
39 staffSchema.pre("save", async function (next) {
40   if (!this.isModified("password")) return next();
41   const salt = await bcrypt.genSalt(10);
42   this.password = await bcrypt.hash(this.password, salt);
43   next();
44 });
45
46 //password matching
47 staffSchema.methods.matchPassword = async function (enteredPassword) {
48   return await bcrypt.compare(enteredPassword, this.password);
49 };
50
51 export default mongoose.model("Staff", staffSchema);
52
```

11. Medication Schema

```
backend > models > JS Medication.model.js > ...
1  import mongoose from "mongoose";
2
3  const medicationSchema = new mongoose.Schema({
4    elder: { type: mongoose.Schema.Types.ObjectId, ref: "Elder", required: true },
5    prescribedBy: { type: mongoose.Schema.Types.ObjectId, ref: "Doctor", required: true },
6    name: { type: String, required: true }, // medicine name
7    dosage: { type: String, required: true },
8    frequency: { type: String, required: true }, // e.g., "2 times a day"
9    startDate: { type: Date, required: true },
10   endDate: { type: Date }
11 }, { timestamps: true });
12
13 export default mongoose.model("Medication", medicationSchema);
14
```

12.Consultaion Schema

```
backend > models > JS Consultation.model.js > ...
1  import mongoose from "mongoose";
2
3  const consultationRequestSchema = new mongoose.Schema({
4    elder: { type: mongoose.Schema.Types.ObjectId, ref: "Elder", required: true },
5    requestedBy: { type: mongoose.Schema.Types.ObjectId, ref: "Caretaker", required: true },
6    doctor: { type: mongoose.Schema.Types.ObjectId, ref: "Doctor", required: true },
7    date: { type: Date, required: true },
8    notes: { type: String },
9    status: { type: String, enum: ["Pending", "Approved", "Completed"], default: "Pending" }
10  }, { timestamps: true });
11
12  export default mongoose.model("ConsultationRequest", consultationRequestSchema);
13
```


13.Doctor Schema

```
backend > models > JS doctor_model.js > ...
1  import mongoose from "mongoose";
2
3  const doctorSchema = new mongoose.Schema(
4    {
5      staff: {
6        type: mongoose.Schema.Types.ObjectId,
7        ref: "Staff",
8        required: true,
9        unique: true, // Each doctor must have one staff account
10     },
11     specialization: {
12       type: String,
13       required: true,
14       trim: true,
15     },
16
17     appointments: {
18       type: mongoose.Schema.Types.ObjectId,
19       ref: "Appointment",
20     },
21   },
22   { timestamps: true }
23 );
24
25 export default mongoose.model("Doctor", doctorSchema);
26
```

14. Caretaker Schema

```
backend > models > JS caretaker_model.js > caretakerSchema > assignedElders
1  import mongoose from "mongoose";
2
3  const caretakerSchema = new mongoose.Schema(
4    {
5      staff: {
6        type: mongoose.Schema.Types.ObjectId,
7        ref: "Staff",
8        required: true,
9        unique: true,
10     },
11     assignedElders: [
12       {
13         type: mongoose.Schema.Types.ObjectId,
14         ref: "Elder",
15       },
16     ],
17   },
18   { timestamps: true }
19 );
20
21 export default mongoose.model("Caretaker", caretakerSchema);
22
```