IT2020 – IT Project

Year 2, Semester 2

2025

Activity 4

Group Details

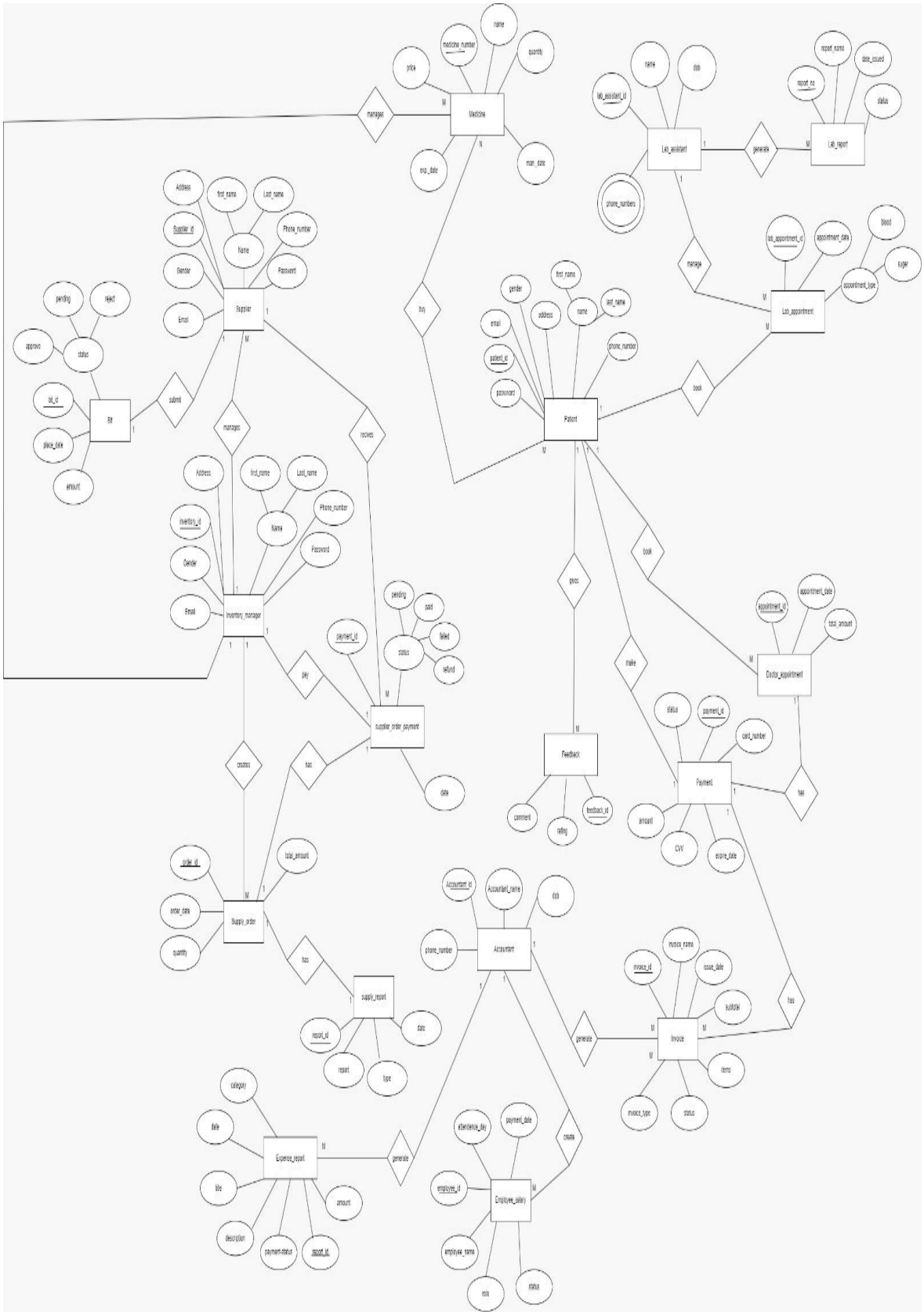
Group Number: ITP25_WE_B01_01_204

Topic: Medical Center Management System

	Student Registration No	Student Name	Student Email	Mobile no
1	IT23810464	Ekanayaka E W I D	IT23810464@my.sliit.lk	0768370886
2	IT23669758	Liayanaarachchi L A D A	IT23669758@my.sliit.lk	0772712630
3	IT23588714	Dinsara K H S	IT23588714@my.sliit.lk	0703287271
4	IT23543232	Navodya A K	IT23543232@my.sliit.lk	0764449102
5	IT23572638	Thathsarani J R	IT23572638@my.sliit.lk	0703119829

Contents

01. ER Diagram – (ER.png)



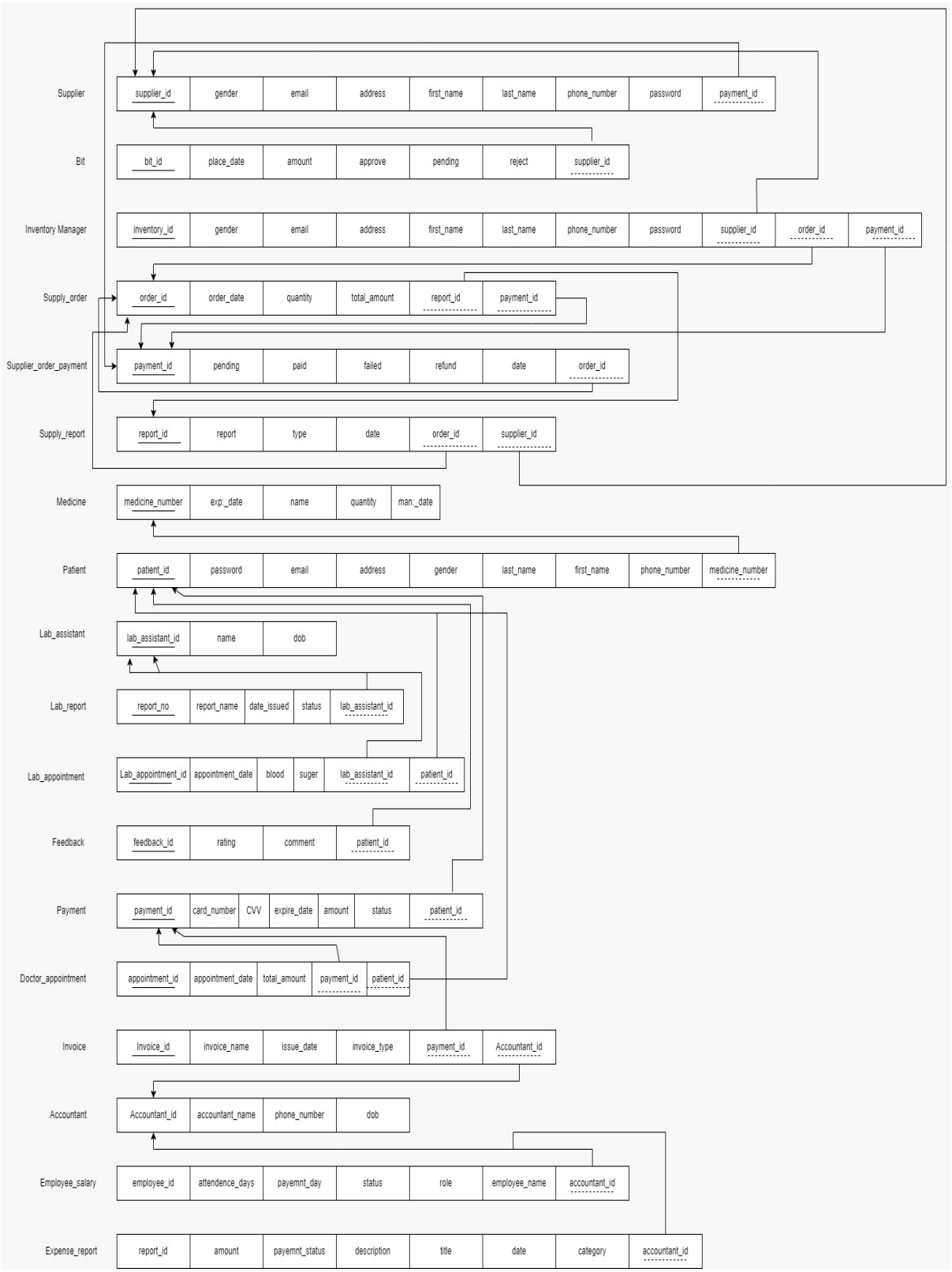
02. Normalization

Entity	1NF	2NF	3NF	BCNF
Patient	Patient_ID (PK), First_Name, Last_Name, Email, Phone_Number, Address, Gender, Password	(Single key → already in 2NF) Patient_ID (PK), First_Name, Last_Name, Email, Phone_Number, Address, Gender, Password	(No transitive dependencies) Patient_ID (PK), First_Name, Last_Name, Email, Phone_Number, Address, Gender, Password	Same as 3NF → Patient_ID (PK), First_Name, Last_Name, Email, Phone_Number, Address, Gender, Password
Doctor_Appointment	Appointment_ID (PK), Patient_ID (FK), Appointment_Date, Test_Amount	Appointment_ID (PK), Patient_ID (FK), Appointment_Date, Test_Amount	Appointment_ID (PK), Patient_ID (FK), Appointment_Date, Test_Amount	Same as 3NF → Appointment_ID (PK), Patient_ID (FK), Appointment_Date, Test_Amount
Lab_Assistant	Lab_Assistant_ID (PK), Name, Phone_Number, Email, Address, DOB	Lab_Assistant_ID (PK), Name, Phone_Number, Email, Address, DOB	Lab_Assistant_ID (PK), Name, Phone_Number, Email, Address, DOB	Same as 3NF → Lab_Assistant_ID (PK), Name, Phone_Number, Email, Address, DOB
Lab_Appointment	Lab_Appointment_ID (PK), Patient_ID (FK), Lab_Assistant_ID (FK), Appointment_Date, Appointment_Type	Lab_Appointment_ID (PK), Patient_ID (FK), Lab_Assistant_ID (FK), Appointment_Date, Appointment_Type	Lab_Appointment_ID (PK), Patient_ID (FK), Lab_Assistant_ID (FK), Appointment_Date, Appointment_Type	Same as 3NF → Lab_Appointment_ID (PK), Patient_ID (FK), Lab_Assistant_ID (FK), Appointment_Date, Appointment_Type
Lab_Report	Report_ID (PK), Lab_Appointment_ID (FK), Blood, Sugar, Result_Date, Export_Date, Status	Report_ID (PK), Lab_Appointment_ID (FK), Blood, Sugar, Result_Date, Export_Date, Status	Report_ID (PK), Lab_Appointment_ID (FK), Blood, Sugar, Result_Date, Export_Date, Status	Same as 3NF → Report_ID (PK), Lab_Appointment_ID (FK), Blood, Sugar, Result_Date, Export_Date, Status
Feedback	Feedback_ID (PK), Patient_ID (FK), Comment, Rating, Date	Feedback_ID (PK), Patient_ID (FK), Comment, Rating, Date	Feedback_ID (PK), Patient_ID (FK), Comment, Rating, Date	Same as 3NF → Feedback_ID (PK), Patient_ID (FK), Comment, Rating, Date
Payment	Payment_ID (PK), Amount, CVV,	Payment_ID (PK), Amount, CVV,	Payment_ID (PK), Amount, CVV,	Same as 3NF → Payment_ID (PK), Amount,

	Card_Number, Expire_Date	Card_Number, Expire_Date	Card_Number, Expire_Date	CVV, Card_Number, Expire_Date
Invoice	Invoice_ID (PK), Invoice_Name, Invoice_Type, Invoice_Date, Amount, Status	Invoice_ID (PK), Invoice_Name, Invoice_Type, Invoice_Date, Amount, Status	Invoice_ID (PK), Invoice_Name, Invoice_Type, Invoice_Date, Amount, Status	Same as 3NF → Invoice_ID (PK), Invoice_Name, Invoice_Type, Invoice_Date, Amount, Status
Supplier	Supplier_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Supplier_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Supplier_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Same as 3NF → Supplier_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password
Supplier_Order	Order_ID (PK), Supplier_ID (FK), Order_Date, Quantity, Item	Order_ID (PK), Supplier_ID (FK), Order_Date, Quantity, Item	Order_ID (PK), Supplier_ID (FK), Order_Date, Quantity, Item	Same as 3NF → Order_ID (PK), Supplier_ID (FK), Order_Date, Quantity, Item
Supplier_Order_Payment	Payment_ID (PK), Order_ID (FK), Status, Amount, Date	Payment_ID (PK), Order_ID (FK), Status, Amount, Date	Payment_ID (PK), Order_ID (FK), Status, Amount, Date	Same as 3NF → Payment_ID (PK), Order_ID (FK), Status, Amount, Date
Inventory_Manager	Manager_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Manager_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Manager_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password	Same as 3NF → Manager_ID (PK), First_Name, Last_Name, Phone_Number, Address, Email, Password
Medicine	Medicine_ID (PK), Name, Quantity, MFD, EXP, Price	Medicine_ID (PK), Name, Quantity, MFD, EXP, Price	Medicine_ID (PK), Name, Quantity, MFD, EXP, Price	Same as 3NF → Medicine_ID (PK), Name, Quantity, MFD, EXP, Price
Employee	Employee_ID (PK), Employee_Name, Role, Phone_Number	Employee_ID (PK), Employee_Name, Role, Phone_Number	Employee_ID (PK), Employee_Name, Role, Phone_Number	Same as 3NF → Employee_ID (PK), Employee_Name, Role, Phone_Number
Employee_Salary	Salary_ID (PK), Employee_ID (FK), Attendance_Days, Base_Salary,	Salary_ID (PK), Employee_ID (FK), Attendance_Days, Base_Salary,	Salary_ID (PK), Employee_ID (FK), Attendance_Days, Base_Salary,	Same as 3NF → Salary_ID (PK), Employee_ID (FK), Attendance_Days, Base_Salary,

	Total_Salary, Status	Total_Salary, Status	Total_Salary, Status	Total_Salary, Status
Expense_Report	Expense_ID (PK), Category, Date, Description, Amount	Expense_ID (PK), Category, Date, Description, Amount	Expense_ID (PK), Category, Date, Description, Amount	Same as 3NF → Expense_ID (PK), Category, Date, Description, Amount

03.Database Schema – (Normalization.png)



04.Build the Database

Appointment model

```
import mongoose from "mongoose";

const appointmentSchema = new mongoose.Schema(
  {
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'user', required:
true },
    doctorId: { type: String, required: true },
    doctorName: { type: String, default: '' },
    speciality: { type: String, default: '' },
    date: { type: Date, required: true },
    slot: { type: String, required: true },
    status: { type: String, enum: ['booked', 'completed', 'cancelled'],
default: 'booked' },
    notes: { type: String, default: '' },
    fees: { type: Number, default: 0 }
  },
  { timestamps: true }
);

const appointmentModel = mongoose.models.appointment ||
mongoose.model('appointment', appointmentSchema);

export default appointmentModel;
```

CartModel.js

```
import mongoose from "mongoose";

const cartItemSchema = new mongoose.Schema(
  {
    inventoryItem: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Inventory",
      required: [true, "Inventory item is required"],
      validate: {
        validator: async function (id) {
          const inventory = await mongoose.model("Inventory").findById(id);
          return !!inventory;
        },
        message: "Referenced inventory item does not exist",
      },
    },
```

```
    quantity: {
      type: Number,
      required: [true, "Quantity is required"],
      min: [1, "Quantity must be at least 1"],
      validate: {
        validator: Number.isInteger,
        message: "Quantity must be an integer",
      },
    },
  },
  { _id: false }
);

const cartSchema = new mongoose.Schema(
  {
    userEmail: {
      type: String,
      required: [true, "User email is required"],
      lowercase: true,
      trim: true,
      match: [/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please provide a valid email"],
    },
    items: {
      type: [cartItemSchema],
      validate: {
        validator: function (items) {
          const ids = items.map((item) => item.inventoryItem.toString());
          return new Set(ids).size === ids.length;
        },
        message: "Cart cannot contain duplicate inventory items",
      },
    },
    total: {
      type: Number,
      required: [true, "Total is required"],
      min: [0, "Total cannot be negative"],
      get: function (v) {
        return parseFloat(v.toFixed(2));
      },
    },
  },
  {
    timestamps: true,
    toJSON: { getters: true },
    toObject: { getters: true },
  }
);
```

```
cartSchema.index({ userEmail: 1 });

cartSchema.virtual("itemCount").get(function () {
  return this.items.reduce((sum, item) => sum + item.quantity, 0);
});

cartSchema.pre("save", async function (next) {
  if (this.isModified("items")) {
    let calculatedTotal = 0;
    for (const item of this.items) {
      const inventoryItem = await mongoose
        .model("Inventory")
        .findById(item.inventoryItem);
      if (inventoryItem) {
        calculatedTotal += inventoryItem.price * item.quantity;
      }
    }
    this.total = parseFloat(calculatedTotal.toFixed(2));
  }
  next();
});

const Cart = mongoose.model("Cart", cartSchema);
export default Cart;
```

UserModel.js

```
import mongoose from "mongoose";

const userSchema = new mongoose. Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    image: { type: String, default:"
    address: { type: Object, default:{line1:'',line2:''} },
    gender:{type:String,default:"Not Selected"},
    dob:{type:String,default:"Not Selected"},
    phone:{type:String,default:"0000000000"},
})

const userModel = mongoose.models.user || mongoose.model('user', userSchema)

export default userModel
```

Client.js

```
import mongoose from "mongoose";

const ClientSchema = new mongoose.Schema(
  {
    name: { type: String, required: true, trim: true },
    email: { type: String, required: true, trim: true, lowercase: true },
    phone: { type: String, required: true, trim: true },
    status: { type: String, enum: ["ACTIVE", "PENDING", "INACTIVE"], default:
"ACTIVE" },
    clientType: {
      type: String,
      enum: [
        "Independent Pharmacy",
        "Pharmacy Chain",
        "Hospital",
        "Medical Center",
      ],
      default: "Independent Pharmacy",
    },
    address: { type: String, required: true, trim: true },
    notes: { type: String, default: "" },
    totalOrders: { type: Number, default: 0 },
    lastOrderDate: { type: Date, default: null },
    rating: { type: Number, default: 0 },
    isDeleted: { type: Boolean, default: false },
  },
  { timestamps: true }
);

ClientSchema.index({ email: 1 });
ClientSchema.index({ name: 1 });

export default mongoose.model("Client", ClientSchema);
```

ContactModel.js

```
import mongoose from 'mongoose'

const contactSchema = new mongoose.Schema(
  {
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'user' },
    name: { type: String, required: true },
    email: { type: String, required: true },
```

```
    phone: { type: String, default: '' },
    subject: { type: String, default: '' },
    message: { type: String, required: true },
    status: { type: String, enum: ['new','replied'], default: 'new' },
    adminReply: { type: String, default: '' },
    repliedAt: { type: Date }
  },
  { timestamps: true }
)

const contactModel = mongoose.models.contact || mongoose.model('contact',
contactSchema)

export default contactModel
```

Doctormodel.js

```
import mongoose from "mongoose";

const doctorSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    image: { type: String, required: true },
    speciality: { type: String, required: true },
    degree: { type: String, required: true },
    experience: { type: String, required: true },
    about: { type: String, required: true },
    available: { type: Boolean, required: true },
    fees: { type: Number, required: true },
    address: { type: Object, required: true },
    date: { type: Number, required: true },
    slots_booked: { type: Object, default: {} }
}, { minimize: false })

const doctorModel = mongoose.models.doctor || mongoose.model('doctor',
doctorSchema)

export default doctorModel
```

InventoryInvoice.js

```
import mongoose from "mongoose";

const medicineItemSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Medicine name is required"],
      trim: true,
      maxlength: [150, "Medicine name cannot exceed 150 characters"],
    },
    price: {
      type: Number,
      required: [true, "Price is required"],
      min: [0, "Price cannot be negative"],
      get: (v) => parseFloat(v.toFixed(2)),
    },
    quantity: {
      type: Number,
      required: [true, "Quantity is required"],
      min: [1, "Quantity must be at least 1"],
      validate: {
        validator: Number.isInteger,
        message: "Quantity must be an integer",
      },
    },
  },
  { _id: false }
);

const inventoryInvoiceSchema = new mongoose.Schema(
  {
    items: {
      type: [medicineItemSchema],
      required: [true, "Invoice must contain at least one medicine item"],
      validate: {
        validator: (arr) => arr.length > 0,
        message: "Invoice must contain at least one medicine item",
      },
    },
    status: {
      type: String,
      enum: ["pending", "approved", "cancelled"],
      default: "pending",
    },
  },
  {
    timestamps: true,
```

```
    toJSON: { getters: true },
    toObject: { getters: true },
  }
);

inventoryInvoiceSchema.virtual("totalAmount").get(function () {
  return this.items.reduce((sum, item) => sum + item.price * item.quantity,
0);
});

export default mongoose.model("InventoryInvoice", inventoryInvoiceSchema);
```

Inventory_manageer_model.js

```
import mongoose from "mongoose";

const inventoryManagerSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    image: { type: String, default: "Not Selected"},
    address: { type: Object, default: { line1: '', line2: '' } },
    gender: { type: String, default: "Not Selected" },
    dob: { type: String, default: "Not Selected" },
    phone: { type: String, default: "0000000000" },
    department: { type: String, required: true },
    specialization: { type: String, required: true },
    employeeId: { type: String, required: true },
    date: { type: Number, required: true }
});

const inventoryManagerModel = mongoose.models.inventoryManager ||
mongoose.model('inventoryManager', inventoryManagerSchema);

export default inventoryManagerModel;
```

InventoryModel

```
import mongoose from "mongoose";

// Custom validation for email format
const validateEmail = (email) => {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
};

const inventorySchema = new mongoose.Schema(
  {
    itemName: {
      type: String,
      required: [true, "Item name is required"],
      trim: true,
      maxlength: [100, "Item name cannot exceed 100 characters"],
    },
    inStockQuantity: {
      type: Number,
      required: [true, "Stock quantity is required"],
      min: [0, "Stock quantity cannot be negative"],
      validate: {
        validator: Number.isInteger,
        message: "Stock quantity must be an integer",
      },
    },
    supplierEmail: {
      type: String,
      required: [true, "Supplier email is required"],
      lowercase: true,
      trim: true,
      validate: {
        validator: validateEmail,
        message: "Please provide a valid supplier email",
      },
    },
    imageUrl: {
      type: String,
      trim: true,
      validate: {
        validator: function (v) {
          if (!v) return true; // Allow empty URLs
          return /^https?:\/\/.+\.(jpg|jpeg|png|gif|webp)$/i.test(v);
        },
        message:
          "Image URL must be a valid HTTPS URL ending with image extension",
      },
```

```
    },
    description: {
      type: String,
      trim: true,
      maxlength: [500, "Description cannot exceed 500 characters"],
    },
    price: {
      type: Number,
      required: [true, "Price is required"],
      min: [0, "Price cannot be negative"],
      get: function (v) {
        return parseFloat(v.toFixed(2));
      },
    },
    expireDate: {
      type: Date,
      validate: {
        validator: function (v) {
          return !v || v > new Date(); // Allow null/undefined, otherwise must
be future date
        },
        message: "Expiration date must be in the future",
      },
    },
    itemCode: {
      type: String,
      required: [true, "Item code is required"],
      unique: true,
      trim: true,
      uppercase: true,
      minlength: [3, "Item code must be at least 3 characters"],
      maxlength: [20, "Item code cannot exceed 20 characters"],
      match: [
        /^[A-Z0-9-_]+$/,
        "Item code can only contain uppercase letters, numbers, hyphens, and
underscores",
      ],
    },
    category: {
      type: String,
      trim: true,
      maxlength: [100, "Category cannot exceed 100 characters"],
    },
    isDeleted: {
      type: Boolean,
      default: false,
    },
  },
```

```
  {
    timestamps: true,
    toJSON: { getters: true },
    toObject: { getters: true },
  }
);

// Indexes
inventorySchema.index({
  itemName: "text",
  description: "text",
  itemCode: "text",
  category: "text",
});
inventorySchema.index({ expireDate: 1 });
inventorySchema.index({ supplierEmail: 1 });
inventorySchema.index({ inStockQuantity: 1, expireDate: 1 });

inventorySchema.virtual("isExpired").get(function () {
  return this.expireDate && this.expireDate < new Date();
});

inventorySchema.pre("save", function (next) {
  if (this.isModified("inStockQuantity")) {
    this.inStockQuantity = Math.floor(this.inStockQuantity);
  }
  next();
});

const Inventory = mongoose.model("Inventory", inventorySchema);

export default Inventory;
```

InventoryRequest

```
import mongoose from "mongoose";

const medicineItemSchema = new mongoose.Schema(
  {
    medicineName: {
      type: String,
      required: [true, "Medicine name is required"],
      trim: true,
      maxlength: [150, "Medicine name cannot exceed 150 characters"],
    },
    manufacturer: {
      type: String,
      trim: true,
      maxlength: [150, "Manufacturer name cannot exceed 150 characters"],
    },
    quantity: {
      type: Number,
      required: [true, "Quantity is required"],
      min: [1, "Quantity must be at least 1"],
      validate: {
        validator: Number.isInteger,
        message: "Quantity must be an integer",
      },
    },
  },
  { _id: false }
);

const inventoryRequestSchema = new mongoose.Schema(
  {
    medicines: {
      type: [medicineItemSchema],
      required: [true, "At least one medicine item is required"],
      validate: {
        validator: function (medicines) {
          return medicines && medicines.length > 0;
        },
        message: "Request must contain at least one medicine item",
      },
    },
    message: {
      type: String,
      trim: true,
      maxlength: [500, "Message cannot exceed 500 characters"],
    },
    status: {
      type: String,
```

```
      enum: ["sent", "approved", "ignored"],
      default: "sent",
    },
    isDeleted: {
      type: Boolean,
      default: false,
    },
  },
  {
    timestamps: true, // createdAt & updatedAt
  }
);

// Indexes
inventoryRequestSchema.index({
  "medicines.medicineName": "text",
  "medicines.manufacturer": "text",
  message: "text",
});
inventoryRequestSchema.index({ status: 1, isDeleted: 1 });

// Pre-save middleware for integer quantities
inventoryRequestSchema.pre("save", function (next) {
  if (this.isModified("medicines")) {
    this.medicines.forEach((medicine) => {
      medicine.quantity = Math.floor(medicine.quantity);
    });
  }
  next();
});

const InventoryRequest = mongoose.model("InventoryRequest",
inventoryRequestSchema);
export default InventoryRequest;
```

labAssistantModel.js

```
import mongoose from "mongoose";

const labAssistantSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    image: { type: String, default: "Not selected" },
    address: { type: Object, default: { line1: '', line2: '' } },
    gender: { type: String, default: "Not Selected" },
    dob: { type: String, default: "Not Selected" },
    phone: { type: String, default: "0000000000" },
```

```
    department: { type: String, required: true },
    specialization: { type: String, required: true },
    licenseNumber: { type: String, required: true },
    date: { type: Number, required: true }
});

const labAssistantModel = mongoose.models.labAssistant ||
mongoose.model('labAssistant', labAssistantSchema);

export default labAssistantModel;
```

SupplierModel.js

```
import mongoose from "mongoose";

const supplierSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    image: { type: String, default: "Not Selected" },
    address: { type: Object, default: { line1: '', line2: '' } },
    gender: { type: String, default: "Not Selected" },
    dob: { type: String, default: "Not Selected" },
    phone: { type: String, default: "0000000000" },
    companyName: { type: String, required: true },
    businessType: { type: String, required: true },
    licenseNumber: { type: String, required: true },
    date: { type: Number, required: true }
});

const supplierModel = mongoose.models.supplier || mongoose.model('supplier',
supplierSchema);

export default supplierModel;
```

Appointment.js

```
import mongoose from "mongoose";

const appointmentSchema = new mongoose.Schema({
  fullName: { type: String, required: true },
  age: { type: Number, required: true },
  email: { type: String, required: true },
  phone: { type: String, required: true },
  gender: { type: String, required: true },
  testType: { type: String, required: true },
  labLocation: { type: String, required: true },
  appointmentDate: { type: String, required: true },
  preferredTime: { type: String, required: true },
  customTime: { type: String },
  notes: { type: String },
  insuranceProvider: { type: String },
  insuranceId: { type: String },
  physician: { type: String },
  terms: { type: Boolean, required: true },
  referenceNumber: { type: String, required: true },

  // ◇ Status controlled manually in dashboard (Scheduled |
Completed)
  status: {
    type: String,
    enum: ["Scheduled", "Completed"],
    default: "Scheduled"
  }
}, { timestamps: true });

const Appointment = mongoose.model("Appointment",
appointmentSchema);
export default Appointment;
```

PatienForm.js

```
import mongoose from "mongoose";

const patientFormSchema = new mongoose.Schema({
  firstName:     { type: String, required: true },
  lastName:      { type: String, required: true },
  email:         { type: String, required: true },
  phoneNumber:   { type: String, required: true },
  dateOfBirth:   { type: Date,   required: true },
  gender:        { type: String, required: true },
  preferredDate: { type: Date,   required: true },
```

```
  preferredTime: { type: String, required: true },
  customTime:    { type: String },
  testType:      { type: String, required: true },
  urgencyLevel:  { type: String, enum:
["Normal","High","Urgent"], default: "Normal" },
  referringDoctor:{ type: String },
  insuranceProvider:{ type: String },
  specialRequirements:{ type: String },
  contactPreference: [{ type: String }]
}, {
  collection: "patient-form",
  timestamps: true
});

const PatientForm = mongoose.model("PatientForm",
patientFormSchema);
export default PatientForm;
```

EmployeePayments.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const EmployeePaymentSchema = new Schema(
  {
    employeeId: { type: String, required: true },
    employeeName: { type: String, required: true },
    role: {
      type: String,
      enum: ["Doctor", "Nurse", "Attendant", "Receptionist"],
      required: true,
    },
    baseSalary: { type: Number, required: true },
    attendanceDays: { type: Number, required: true },
    totalSalary: { type: Number, required: true }, //  Auto-calculated
    paymentDate: { type: Date, required: true },
    status: {
      type: String,
      enum: ["pending", "paid", "failed"],
      default: "pending",
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("EmployeePayment", EmployeePaymentSchema);
```

Expense.js

```
const mongoose = require('mongoose');

const ExpenseSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  amount: { type: Number, required: true, min: 0 },
  category: { type: String, default: "General", trim: true },
  date: { type: Date, required: true, default: Date.now },
  description: { type: String, default: "" },
  addedBy: { type: String, default: null },
  paymentMethod: { type: String, enum: ['cash', 'card', 'online', 'other'],
default: 'cash' },
  receiptNumber: { type: String, default: '' },
  attachments: [{ type: String }],
  tags: [{ type: String }],

  // Payment tracking fields
  paymentStatus: {
    type: String,
    enum: ['unpaid', 'pending', 'paid', 'failed'],
    default: 'unpaid'
  },
  paymentOrderId: { type: String, default: null },
  paymentId: { type: String, default: null }, // PayHere payment ID
  paymentDate: { type: Date, default: null },
  paymentAmount: { type: Number, default: null }
}, { timestamps: true });

module.exports = mongoose.model('Expense', ExpenseSchema);
```

Invoice.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const InvoiceSchema = new Schema(
  {
    id: { type: String, required: true }, // Business ID (e.g., "INV001")
    patientId: { type: String, required: true },
    date: { type: Date, required: true },
    items: [
```

```
      {
        service: { type: String, required: true },
        quantity: { type: Number, required: true },
        price: { type: Number, required: true }
      }
    ],
    subtotal: { type: Number, required: true },
    tax: { type: Number, required: true },
    total: { type: Number, required: true },
    status: {
      type: String,
      enum: ["unpaid", "paid", "pending", "cancelled"],
      default: "unpaid"
    },
    cashierId: { type: String } // optional
  },
  { timestamps: true }
);

module.exports = mongoose.model("Invoice", InvoiceSchema);
```

PayhereController.js

```
const crypto = require("crypto");

// PayHere Configuration (Use environment variables in production)
const MERCHANT_ID = "1232089"; // Your PayHere Merchant ID
const MERCHANT_SECRET =
"MTExNjEyNjQxNjE4MjM0MzE3NTIxODc4NTM5MDg3Mzc5MDA0NTg2NA=="; // Your PayHere
Secret Key
const RETURN_URL = "http://localhost:3000/success";
const CANCEL_URL = "http://localhost:3000/cancel";
const NOTIFY_URL = "http://localhost:5000/api/payhere/notify";

// Generate MD5 Hash for PayHere
const generateHash = (merchantId, orderId, amount, currency, merchantSecret)
=> {
  const hashedSecret = crypto
    .createHash("md5")
    .update(merchantSecret)
    .digest("hex")
    .toUpperCase();

  const amountFormatted = parseFloat(amount).toFixed(2);
  const hashString =
`${merchantId}${orderId}${amountFormatted}${currency}${hashedSecret}`;

  return crypto
```

```
    .createHash("md5")
    .update(hashString)
    .digest("hex")
    .toUpperCase();
};

// Create Payment Order
exports.createOrder = async (req, res) => {
  try {
    const {
      order_id,
      items,
      currency,
      amount,
      first_name,
      last_name,
      email,
      phone,
      address,
      city,
      country,
    } = req.body;

    // Validate required fields
    if (!order_id || !amount || !first_name || !last_name || !email || !phone)
{
      return res.status(400).json({ message: "Missing required fields" });
    }

    // Generate hash
    const hash = generateHash(MERCHANT_ID, order_id, amount, currency,
MERCHANT_SECRET);

    // Prepare PayHere form fields
    const payhereData = {
      merchant_id: MERCHANT_ID,
      return_url: RETURN_URL,
      cancel_url: CANCEL_URL,
      notify_url: NOTIFY_URL,
      order_id: order_id,
      items: items,
      currency: currency,
      amount: amount,
      first_name: first_name,
      last_name: last_name,
      email: email,
      phone: phone,
      address: address,
```

```
      city: city,
      country: country,
      hash: hash,
    };

    console.log("PayHere Order Created:", payhereData);
    res.status(200).json(payhereData);
  } catch (error) {
    console.error("Error creating PayHere order:", error);
    res.status(500).json({ message: "Failed to create order", error:
error.message });
  }
};

// Payment Notification Handler (IPN)
exports.handleNotification = async (req, res) => {
  try {
    const {
      merchant_id,
      order_id,
      payment_id,
      payhere_amount,
      payhere_currency,
      status_code,
      md5sig,
    } = req.body;

    console.log("PayHere Notification Received:", req.body);

    // Verify the hash
    const localMd5sig = crypto
      .createHash("md5")
      .update(
        `${merchant_id}${order_id}${payhere_amount}${payhere_currency}${status
_code}${crypto
          .createHash("md5")
          .update(MERCHANT_SECRET)
          .digest("hex")
          .toUpperCase()}`
      )
      .digest("hex")
      .toUpperCase();

    if (localMd5sig !== md5sig) {
      console.error("Hash verification failed!");
      return res.status(400).json({ message: "Invalid hash" });
    }
```

```
    if (status_code === "2") {
      console.log(`Payment Success: Order ${order_id}, Payment ID:
${payment_id}`);
      // TODO: Update your database with payment success
      // Example: await Order.findOneAndUpdate({ orderId: order_id }, {
status: 'paid', paymentId: payment_id });
    } else if (status_code === "0") {
      console.log(`Payment Pending: Order ${order_id}`);
      // TODO: Mark as pending
    } else {
      console.log(`Payment Failed/Canceled: Order ${order_id}, Status:
${status_code}`);
      // TODO: Mark as failed
    }

    res.status(200).send("OK");
  } catch (error) {
    console.error("Error handling notification:", error);
    res.status(500).json({ message: "Notification handling failed", error:
error.message });
  }
};

// Verify Payment Status (Optional)
exports.verifyPayment = async (req, res) => {
  try {
    const { order_id } = req.params;

    // TODO: Query your database to check payment status
    // const order = await Order.findOne({ orderId: order_id });

    res.status(200).json({
      order_id: order_id,
      status: "success", // or "pending", "failed"
      message: "Payment verified successfully",
    });
  } catch (error) {
    console.error("Error verifying payment:", error);
    res.status(500).json({ message: "Verification failed", error:
error.message });
  }
};
```

05.Contribution

Registration No	Name	Functions
IT23810464	Ekanayaka E W I D	• Draw ER Diagram • Normalize Diagram • Create Database schema • Built Database
IT23669758	Liayanaarachchi L A D A	• Draw ER Diagram • Normalize Diagram • Create Database schema • Built Database
IT23588714	Dinsara K H S	• Draw ER Diagram • Normalize Diagram • Create Database schema • Built Database
IT23543232	Navodya A K	• Draw ER Diagram • Normalize Diagram • Create Database schema • Built Database
IT23572638	Thathsarani J R	• Draw ER Diagram • Normalize Diagram • Create Database schema Built Database