userModel.js

```
const crypto = require("crypto");
const mongoose = require("mongoose");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const { Schema } = mongoose;

// define a user schema
const userSchema = new Schema({
  names: {
    type: String,
    required: [true, "name field shouldn't be blank! required"],
    trim: true,
    lowercase: true,
  },
  email: {
    type: String,
    required: [true, "email field shouldn't be blank! required"],
    unique: true,
    lowercase: true
  },
  phones: {
    type: String,
    required: [true, "phone field shouldn't be blank! required"],
  },
  password: {
    type: String,
    required: [true, "password field shouldn't be blank! required"],
    min: 6,
    max: 1024,
  },
  orders: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Order', // Reference to the Order model for customer's orders.
  }],
  passwordResetAt: Date,
  passwordResetToken: String,
  passwordResetExpires: Date
}, { versionKey: false, timestamps: true });

// encrypt the password before saving the document
userSchema.pre("save", async function (next) {
  const user = this;
  // check if the password field is modified or not
  if (!user.isModified('password')) {
    next();
  }
  // create a salt for password
  const salt = await bcrypt.genSalt(12);

  // create hashed password
  user.password = await bcrypt.hash(user.password, salt);

  next();
});

userSchema.pre("save", function (next) {
  if (this.isModified("password") || this.isNew) return next();

  this.passwordResetAt = Date.now() - 1000;
  next();
});

// compare password
userSchema.methods.compare = async function (password) {
  return await bcrypt.compare(password, this.password);
};

// create token
userSchema.methods.createToken = async function (payload) {
  return jwt.sign(
    payload,
    process.env.ACCESS_TOKEN,
    { expiresIn: process.env.EXPIRES_IN }
  );
};


userSchema.methods.createPasswordResetToken = function () {
  const resetToken = crypto.randomBytes(32).toString("hex");

  this.passwordResetToken = crypto
    .createHash('sha256')
    .update(resetToken)
    .digest('hex');

  this.passwordResetExpires = Date.now() + 10 * 60 * 1000;

  return resetToken;
};

// compile a model
const User = mongoose.model("User", userSchema);
module.exports = User;
```

restaurantModel.js

```
const crypto = require("crypto");
const mongoose = require("mongoose");
const bcrypt = require("bcrypt");
const { Schema } = require("mongoose");

const restaurantSchema = new Schema({
  names: {
    type: String,
    required: true,
  },
  address: {
    type: String,
    required: true,
  },
  cuisines: {
    type: [String], // Indian, Italian, American
    required: true,
  },
  rating: {
    type: Number,
    default: 0, // You can set a default rating value if needed.
  },
  openingTimes: {
    type: String, // You can store the opening time as a string, e.g., "08:00 AM"
    required: true,
  },
  closingTimes: {
    type: String, // You can store the closing time as a string, e.g., "10:00 PM"
    required: true,
  },
  // Authentication fields
  username: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  role: {
    type: String,
    enum: ["admin", "owner", "moderator"],
  },
  menu: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'FoodItem', // Reference to the FoodItem model for the restaurant's menu.
  }],
}, { versionKey: false, timestamps: true });

// encrypt the password before saving the document
restaurantSchema.pre("save", async function (next) {
  const restaurentUser = this;
  // check if the password field is modified or not
  if (!restaurentUser.isModified('password')) {
    next();
  }
  // create a salt for password
  const salt = await bcrypt.genSalt(12);

  // create hashed password
  restaurentUser.password = await bcrypt.hash(restaurentUser.password, salt);

  next();
});

// compare password
restaurantSchema.methods.compare = async function (password) {
  return await bcrypt.compare(password, this.password);
};

const Restaurant = mongoose.model('Restaurant', restaurantSchema);

module.exports = Restaurant;
```

orderModel.js

```
const mongoose = require('mongoose');
const { ObjectId } = mongoose.Schema.Types;

// Create order schema
const orderSchema = new mongoose.Schema({
  users: {
    type: ObjectId,
    ref: 'User', // Reference to the Customer/user model for the customer who placed the order.
    required: true,
  },
  restaurant: {
    type: ObjectId,
    ref: 'Restaurant', // Reference to the Restaurant model for the restaurant where the order was placed.
    required: true,
  },
  items: [{
    foodItems: {
      type: ObjectId,
      ref: 'FoodItem', // Reference to the FoodItem model for the items in the order.
      required: true,
    },
    quantity: {
      type: Number,
      required: true,
    },
  }],
  price: {
    type: Number,
    required: true,
  },
  orderDate: {
    type: Date,
    default: Date.now, // You can set a default order date to the current date and time.
  },
  paid: {
    type: Boolean,
    default: true
  }
}, { versionKey: false });

orderSchema.pre(/^find/, function (next) {
  this.populate({ path: "user", select "-_id name email" })
    .populate({ path: "restaurant", select: "_id name email address" })
    .populate({ path: "items.foodItem", select: "itemName price description picture" });
  next();
});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;
```

foodItemModel.js

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

const foodItemSchema = new mongoose.Schema({
  itemNames: {
    type: String,
    required: true,
  },
  quantity: {
    type: Number,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
  descriptions: {
    type: String,
    required: true,
  },
  picture: {
    type: String, // You can store the URL of the picture or the image file name.
    required: true,
  },
}, { versionKey: false, timestamps: true });

const FoodItem = mongoose.model('FoodItem', foodItemSchema);

module.exports = FoodItem;
```

queryModel.js

```
const mongoose = require('mongoose');
const { Schema } = mongoose;

const querySchema = new Schema({
  names: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  message: {
    type: String,
    required: true,
  }
}, { versionKey: false, timestamps: true });

const Query = mongoose.model('Query', querySchema);
module.exports = Query;
```