ASSIGNMENT(220126)

Scenario:

An online Event Management System stores event details such as Θtle,
date, locaΘon, and parΘcipants. The backend must support CRUD
operaΘons using MongoDB.
QuesΘons:
1. How would you implement Create, Read, Update, and Delete
operaΘons using Mongoose?
2. What is the role of methods like save(), find(),
findByIdAndUpdate(), and findByIdAndDelete()?
3. How does Mongoose handle data validaΘon at the schema level?
4. How can you implement paginaΘon and filtering in MongoDB
queries?
5. How would you handle invalid ObjectId or missing document
Errors?

Scenario: Online Event Management System (MongoDB + Mongoose)

---

1. Implementing CRUD operations using Mongoose

🔹 Event Schema

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

const eventSchema = new mongoose.Schema({
  title: { type: String, required: true },
  date: { type: Date, required: true },
  location: { type: String, required: true },
  participants: { type: Number, default: 0 }
});

module.exports = mongoose.model("Event", eventSchema);
```

🔹 Create (C)

```
const event = new Event({
  title: "Tech Conference",
  date: "2026-02-10",
  location: "Bangalore",
  participants: 150
});

await event.save();
```

🔹 Read (R)

```
// Get all events
const events = await Event.find();

// Get single event by ID
const event = await Event.findById(id);
```

🔹 Update (U)

```
await Event.findByIdAndUpdate(id, { location: "Hyderabad" }, { new:
true });
```

🔹 Delete (D)

```
await Event.findByIdAndDelete(id);
```

---

2. Role of Mongoose methods

Method	Role
`save()`	Saves a new document or updates an existing one
`find()`	Fetches multiple documents

`findById()`	Fetches one document using ObjectId
`findByIdAndUpdate()`	Updates a document by ID
`findByIdAndDelete()`	Deletes a document by ID

📌 These methods simplify MongoDB queries and return **Promises**.

---

3. Schema-level data validation in Mongoose

Mongoose validates data **before saving to MongoDB**.

🔹 Example validations

```
const eventSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    minlength: 3
  },
  date: {
    type: Date,
    required: true
  },
  participants: {
    type: Number,
    min: 0
  }
});
```

✅ Prevents invalid data
✅ Ensures consistency
❌ Rejects incorrect input automatically

---

4. Pagination and Filtering

🔹 Pagination

```
const page = 2;
const limit = 5;

const events = await Event.find()
  .skip((page - 1) * limit)
  .limit(limit);
```

🔹 Filtering

```
// Filter by location
const events = await Event.find({ location: "Bangalore" });
```

🔹 Pagination + Filtering

```
const events = await Event.find({ location: "Bangalore" })
  .skip((page - 1) * limit)
  .limit(limit);
```

---

5. Handling invalid ObjectId or missing documents

🔹 Invalid ObjectId

```
if (!mongoose.Types.ObjectId.isValid(id)) {
  return res.status(400).json({ message: "Invalid Event ID" });
}
```

🔹 Missing document

```
const event = await Event.findById(id);

if (!event) {
  return res.status(404).json({ message: "Event not found" });
}
```

🔹 Error handling

```
try {
  const event = await Event.findById(id);
} catch (error) {
  res.status(500).json({ message: "Server error" });
}
```

Conclusion

- Mongoose provides easy CRUD operations using built-in methods
- Schema validation ensures clean and valid data
- Pagination improves performance for large datasets
- Proper error handling avoids application crashes