MongoDB Document Data Model: Hands-On Lab (30–60 minutes)

This lab introduces the Document Data Model using MongoDB. You'll install MongoDB via Docker, perform CRUD operations with realistic sample data, and model an applied scenario (a product catalog with reviews and categories). The lab is suitable for beginners and is fully reproducible.

Prerequisites

- Docker (version 24+ recommended) and Docker Compose (v2+)
- A terminal (macOS/Linux/WSL/PowerShell)
- Curl or a REST client (e.g., Postman)

What you'll do

1. Set up MongoDB in Docker and seed sample data
2. Explore the Document Data Model basics
3. Perform CRUD operations using Mongo Shell
4. Apply the model to a product catalog scenario
5. View outputs and respond to practical questions
6. Summarize team contributions

---

1. Setup Instructions

We'll use Docker Compose to start:

- MongoDB Community Server 7.0
- Mongo Express (simple web UI) pinned to 1.0.2

Files:

- `docker-compose.yml` — services and networking
- `seed/` — JSON data to pre-populate the database
- `scripts/` — helper scripts for seeding and testing

Steps

1. Clone this repository

```
git clone <your-repo-url>
cd <your-repo>
```

2. Start services

```
docker compose up -d
```

- Verifies:
  - MongoDB listening on `mongodb://localhost:27017`
  - Mongo Express UI at `http://localhost:8081`

3. Check container status

```
docker compose ps
```

Expected:

- `mongo` Up
- `mongo-express` Up

4. Seed sample data

```
./scripts/seed.sh
```

This loads `products`, `categories`, and `reviews` into a `shop` database.

Versions

- MongoDB: 7.0
- Mongo Express: 1.0.2
- Docker Engine: 24+
- Docker Compose: v2+

Notes:

- Configuration is set in `docker-compose.yml`. Auth is enabled with `MONGO_INITDB_ROOT_USERNAME` and `MONGO_INITDB_ROOT_PASSWORD`.
- For Windows, run scripts via PowerShell equivalents or use Git Bash.

---

2. Document Data Model Basics

MongoDB stores data as BSON (binary JSON). Collections hold documents (like JSON objects) with flexible schemas.

Example product document (denormalized data):

```
{
  "_id": "SKU-1001",
  "name": "Noise-Cancelling Headphones",
  "brand": "AcoustiX",
  "price": 149.99,
  "in_stock": true,
  "categories": ["audio", "accessories"],
  "specs": {
    "color": "black",
    "weight_grams": 250,
    "battery_hours": 30
  },
  "tags": ["wireless", "bluetooth", "ANC"],
  "ratings": {
    "average": 4.5,
    "count": 124
  }
}
```

Key strengths:

- Flexible schema: evolve fields without migrations
- Nested documents: embed related data (e.g., specs, ratings)
- Rich querying on nested fields, arrays, and text

---

3. CRUD Operations

You can use either:

- MongoDB Shell (`mongosh` inside the container)
- Mongo Express UI for quick visibility

3.1 Connect to Mongo Shell

```
docker exec -it mongo mongosh -u root -p rootpassword --authenticationDatabase
admin
```

Switch to the `shop` database:

```
use shop
```

Create

Insert a new product:

```
db.products.insertOne({
  _id: "SKU-2013",
  name: "USB-C Charger 65W",
  brand: "ChargePro",
  price: 39.99,
  in_stock: true,
  categories: ["power", "accessories"],
  specs: { color: "white", wattage: 65, ports: ["USB-C"] },
  tags: ["fast-charge", "compact"],
  ratings: { average: 4.3, count: 56 }
})
```

Insert a review:

```
db.reviews.insertOne({
  product_id: "SKU-2013",
  user: { id: "U-100", name: "Jane Doe" },
  rating: 5,
  comment: "Charges my laptop and phone fast!",
  created_at: new Date()
})
```

Read

Find by ID:

```
db.products.findOne({ _id: "SKU-2001" })
```

Filter and projection:

```
db.products.find(
  { price: { $lt: 100 }, in_stock: true, categories: "accessories" },
  { name: 1, brand: 1, price: 1, _id: 0 }
)
```

Nested field query:

```
db.products.find({ "specs.wattage": { $gte: 60 } })
```

Update

Update a price and add a tag:

```
db.products.updateOne(
  { _id: "SKU-2001" },
  { $set: { price: 34.99 }, $addToSet: { tags: "travel" } }
)
```

Upsert (insert if missing):

```
db.products.updateOne(
  { _id: "SKU-9999" },
  { $set: { name: "Demo Product", price: 9.99, in_stock: false } },
  { upsert: true }
)
```

Delete

Remove a product:

```
db.products.deleteOne({ _id: "SKU-9999" })
```

Cascade-like cleanup (manual):

```
db.reviews.deleteMany({ product_id: "SKU-9999" })
```

4. Applied Scenario: Product Catalog with Reviews

Problem

An online store needs to manage:

- Products with flexible attributes (specs vary by category)
- Categories and tags for navigation
- Customer reviews with ratings and comments
- Efficient reads of product details and related reviews

Why Document Model?

- Products are naturally document-shaped with nested attributes
- Schema flexibility allows adding new specs without migrations
- Embedding summary fields (like ratings.average) supports fast reads
- Separate `reviews` collection keeps large lists manageable

Data Model

Collections:

- `products`: product documents with nested `specs`, `ratings`, arrays `categories`, `tags`

- `categories`: top-level category definitions and metadata
- `reviews`: customer reviews referencing `product_id`

Indexes (performance):

```
db.products.createIndex({ categories: 1 })
db.products.createIndex({ price: 1, in_stock: 1 })
db.reviews.createIndex({ product_id: 1, created_at: -1 })
```

Read product with recent reviews (aggregation):

```
db.products.aggregate([
  { $match: { _id: "SKU-2001" } },
  { $lookup: {
      from: "reviews",
      localField: "_id",
      foreignField: "product_id",
      as: "recent_reviews"
  }},
  { $addFields: {
      recent_reviews: { $slice: ["$recent_reviews", 5] }
  }}
])
```

Update rating summary after new review:

```
// Compute new average and count
const pid = "SKU-2001";
const summary = db.reviews.aggregate([
  { $match: { product_id: pid } },
  { $group: { _id: "$product_id", count: { $sum: 1 }, avg: { $avg: "$rating" } } }
]).toArray()[0];

db.products.updateOne(
  { _id: pid },
  { $set: { "ratings.count": summary.count, "ratings.average": summary.avg } }
);
```

---

5. Visuals and Output

Use Mongo Express at `http://localhost:8081`:

- View `shop` database and browse `products` and `reviews`.
- Confirm inserts, updates, and deletes.

6. Clarity and Reproducibility Checklist

- Deterministic Docker versions
- Seed script for consistent data
- Index creation for realistic performance
- Fully annotated commands

If anything fails:

- Run `docker compose logs mongo` and `docker compose logs mongo-express`
- Ensure ports 27017, 8081 are free
- Use `docker compose down -v` to reset volumes, then up and re-seed

Appendix: Direct (Non-Docker) Installation (Optional)

macOS (Homebrew):

```
brew tap mongodb/brew
brew install mongodb-community@7.0
brew services start mongodb-community@7.0
```

Ubuntu:

```
# Follow https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
sudo systemctl start mongod
```

Connect:

```
mongosh
use shop
```

Seed using `scripts/seed.mongo.js`:

```
mongosh < scripts/seed.mongo.js
```
