MongoDB Schema Design Challenge

for Real Startup

1. E-Commerce Store – Product & Orders

Scenario:

You are building a backend for an online e-commerce store (like Flipkart or Amazon). Customers browse products, place orders, and leave reviews.

Task:

Design schemas for the following collections:

- `users`
- `products`
- `orders`
- `reviews`

Schema Requirements:

- Each user must have name, email (unique), and password
- A product has a title, description, price, category, and stock
- An order must store userId, a list of productIds with quantities, total amount, and orderDate
    - A review links to both userId and productId, contains a rating (1–5), and a comment

Constraints to Apply:

- Use bsonType, required, enum, pattern, and minimum
- Enforce email uniqueness via index
- Validate that ratings are between 1–5

USERS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "email", "password"],
    "properties": {
      "name": { "bsonType": "string" },
      "email": { "bsonType": "string", "pattern": "^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,}$" },
      "password": { "bsonType": "string" }
    }
  }
}

db.users.createIndex({ email: 1 }, { unique: true });
```

PRODUCTS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["title", "description", "price", "category", "stock"],
    "properties": {
      "title": { "bsonType": "string" },
      "description": { "bsonType": "string" },
      "price": { "bsonType": "double", "minimum": 0 },
      "category": { "bsonType": "string" },
      "stock": { "bsonType": "int", "minimum": 0 }
    }
  }
}
```

ORDERS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["userId", "products", "totalAmount", "orderDate"],
    "properties": {
      "userId": { "bsonType": "objectId" },
      "products": {
        "bsonType": "array",
        "items": {
          "bsonType": "object",
          "required": ["productId", "quantity"],
          "properties": {
            "productId": { "bsonType": "objectId" },
            "quantity": { "bsonType": "int", "minimum": 1 }
          }
        }
      },
      "totalAmount": { "bsonType": "double", "minimum": 0 },
      "orderDate": { "bsonType": "date" }
    }
  }
}
```

REVIEWS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["userId", "productId", "rating", "comment"],
    "properties": {
      "userId": { "bsonType": "objectId" },
      "productId": { "bsonType": "objectId" },
      "rating": { "bsonType": "int", "minimum": 1, "maximum": 5,
      },
      "comment": { "bsonType": "string" }
    }
  }
}
```

2. Online Course Platform – Instructors & Students

Scenario:

You're designing a database for an online learning platform like Udemy.

Task:

Design schemas for:

Note: `role: ['student','instructor']`

- `users (can be students or instructors)`
- `courses`
- `enrollments`
- `lessons`

Schema Requirements:

- Users must include name, email, role (student or instructor)
- A course must include title, instructorId, category, price, and createdAt
- Lessons are embedded in the course, and include title, videoURL, and duration (in minutes) •

  Students enroll in courses through the enrollments collection

Constraints to Apply:

- Role should be validated with enum
- Course price should be a number ≥ 0
- Lesson duration must be a number > 0

USERS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "email", "role"],
    "properties": {
      "name": { "bsonType": "string" },
      "email": { "bsonType": "string", "pattern": "^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,}$" },
      "role": { "bsonType": "string", "enum": ["student", "instructor"] }
    }
  }
}
```

COURSES (EMBEDDED WITH LESSONS) SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["title", "instructorId", "category", "price", "createdAt", "lessons"],
    "properties": {
      "title": { "bsonType": "string" },
      "instructorId": { "bsonType": "objectId" },
      "category": { "bsonType": "string" },
      "price": { "bsonType": "double", "minimum": 0 },
      "createdAt": { "bsonType": "date" },
      "lessons": {
        "bsonType": "array",
        "items": {
          "bsonType": "object",
          "required": ["title", "videoURL", "duration"],
          "properties": {
            "title": { "bsonType": "string" },
            "videoURL": { "bsonType": "string" },
            "duration": { "bsonType": "double", "minimum": 0.01 }
          }
        }
      }
    }
  }
}
```

ENROLLMENTS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["studentId", "courseId", "enrolledAt"],
    "properties": {
      "studentId": { "bsonType": "objectId" },
      "courseId": { "bsonType": "objectId" },
      "enrolledAt": { "bsonType": "date" }
    }
  }
}

db.users.createIndex({ email: 1 }, { unique: true });
```

3. Event Booking System – Organizers & Attendees

Scenario:

You are building an event management system like Eventbrite.

Task:

Design collections for:

- `users`
- `events`
- `bookings`

Schema Requirements:

- Users have name, email, and role (organizer or attendee)
- Events include title, organizerId, location, startTime, endTime, and capacity
- Bookings store eventId, attendeeId, and bookingDate

Constraints to Apply:

- Validate that capacity is a positive integer
- Event startTime and endTime should be date types
- Email should follow a valid pattern and be unique

USERS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "email", "role"],
    "properties": {
      "name": { "bsonType": "string" },
      "email": { "bsonType": "string", "pattern": "^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,}$" },
      "role": { "bsonType": "string", "enum": ["organizer", "attendee"] }

}
  }
}
```

EVENTS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["title", "organizerId", "location", "startTime", "endTime", "capacity"],
    "properties": {
      "title": { "bsonType": "string" },
      "organizerId": { "bsonType": "objectId" },
      "location": { "bsonType": "string" },
      "startTime": { "bsonType": "date" },
      "endTime": { "bsonType": "date" },
      "capacity": { "bsonType": "int", "minimum": 1 }
    }
  }
}
```

BOOKINGS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["eventId", "attendeeId", "bookingDate"],
    "properties": {
      "eventId": { "bsonType": "objectId" },
      "attendeeId": { "bsonType": "objectId" },
      "bookingDate": { "bsonType": "date" }
    }
  }
}
```

4. Blogging Platform – Authors & Articles

Scenario:

You are creating a lightweight CMS/blogging system like Medium.

Task:

Design collections for:

- `authors`
- `articles`
- `comments`

Schema Requirements:

- Each author has name, email, and bio
- Articles contain title, content, authorId, tags (array of strings), published (boolean), and createdAt
- Comments reference articleId and include userName, commentText, and postedAt

Constraints to Apply:

- Ensure article title and content are required
- published must be a boolean
- tags should be an array of strings
- Use date type for timestamps

AUTHORS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "email", "bio"],
    "properties": {
      "name": { "bsonType": "string" },
      "email": { "bsonType": "string", "pattern": "^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,}$" },
      "bio": { "bsonType": "string" }
    }
  }
}
```

ARTICLES SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["title", "content", "authorId", "tags", "published", "createdAt"],
    "properties": {
      "title": { "bsonType": "string" },
      "content": { "bsonType": "string" },
      "authorId": { "bsonType": "objectId" },
      "tags": { "bsonType": "array", "items": { "bsonType": "string" } },
      "published": { "bsonType": "bool" },
      "createdAt": { "bsonType": "date" }
    }
  }
}
```

COMMENTS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["articleId", "userName", "commentText", "postedAt"],
    "properties": {
      "articleId": { "bsonType": "objectId" },
      "userName": { "bsonType": "string" },
      "commentText": { "bsonType": "string" },
      "postedAt": { "bsonType": "date" }
    }
  }
}
```

5. Subscription App – Users & Plans

Scenario:

You're building the backend for a SaaS app with subscription plans (like Notion or Canva).

Task:

Design schemas for:

- `users`
- `plans`
- `subscriptions`

Schema Requirements:

- Users should have email, name, and signupDate
- Plans include name, price, features, and billingCycle (monthly, yearly)
- Subscriptions include userId, planId, startDate, and isActive

Constraints to Apply:

- Validate that plan price is ≥ 0
- Billing cycle must use enum
- features should be an array of strings

USERS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["email", "name", "signupDate"],
    "properties": {
      "email": { "bsonType": "string", "pattern": "^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,}$" },
      "name": { "bsonType": "string" },
      "signupDate": { "bsonType": "date" }
    }
  }
}
```

```
db.users.createIndex({ email: 1 }, { unique: true });
```

PLANS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["name", "price", "features", "billingCycle"],
    "properties": {
      "name": { "bsonType": "string" },
      "price": { "bsonType": "double", "minimum": 0 },
      "features": { "bsonType": "array", "items": { "bsonType": "string" } },
      "billingCycle": { "bsonType": "string", "enum": ["monthly", "yearly"] }
    }
  }
}
```

SUBSCRIPTIONS SCHEMA :-

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["userId", "planId", "startDate", "isActive"],
    "properties": {
      "userId": { "bsonType": "objectId" },
      "planId": { "bsonType": "objectId" },
      "startDate": { "bsonType": "date" },
      "isActive": { "bsonType": "bool" }
    }
  }
}
```