Lecture -4 ORM, CRUD & SCHEMA

Agenda

- Crud in MongoDB
- ORM (Mongoose)
- Schema Model
  - Validators
  - Constraints
- Real World
  - User

DB Importance

The goal is to make the database `foolproof,` not just `full-proof`.

- **Foolproof**: Designing the database to be robust against user errors, system failures, and unintended misuse. This includes implementing proper validation, constraints, and access controls to ensure that the data remains accurate and consistent despite potential mistakes or issues.
- **Full-proof**: This term implies a system that is completely immune to any kind of failure or error, which is often unrealistic. No system can be entirely impervious to all possible problems.

`DB is the single source of truth` means that the database contains the most accurate and up-to-date information for a given system or application.

DB use case

Databases often play a crucial role in `enforcing business logic`.

Example:-

First User

User --> Flipkart
(User)
Name

Email
Address
Password
Product history
User order tracking
Cart, Wishlist
Products
Fulfilment Center

Second User

User --> Jio Clone
--->name
--->email {User personal info}
--->address
--->password
--->is Prime member
--->Watch History
--->new -recommendation
Movies
Data Center

Database can enforce complex business logic and ensure that data remains accurate and consistent according to the rules and requirements.

DB Business Logic Enforce

1 Entity 2 Schema
---> Set of Rules that on entity should always follow
---> Properties / Constraints
---> Validation

Entity

Entity are basically unique things which have different different business logic.

Example:

Consider a simple e-commerce database with entities like `Products`, `Orders`, and `Customers`.

- **Product:** Might have columns like `product_id`, `name`, `price`, and `quantity_in_stock`.
- **Order:** Might have columns like `order_id`, `customer_id`, `order_date`, and `total_amount`.
- **Customer:** Might have columns like `customer_id`, `name`, `address`, and `email`.

Whatever unique things there are about which I want to store data in a particular format, we call them entities.

Schema

The schema defines the structure of the documents, including fields, their data types, validation rules, and default values. This is a blueprint for what the documents in your collection will look like.

In simple word, Schema is set of Rules that on entity should always follow.

example:-

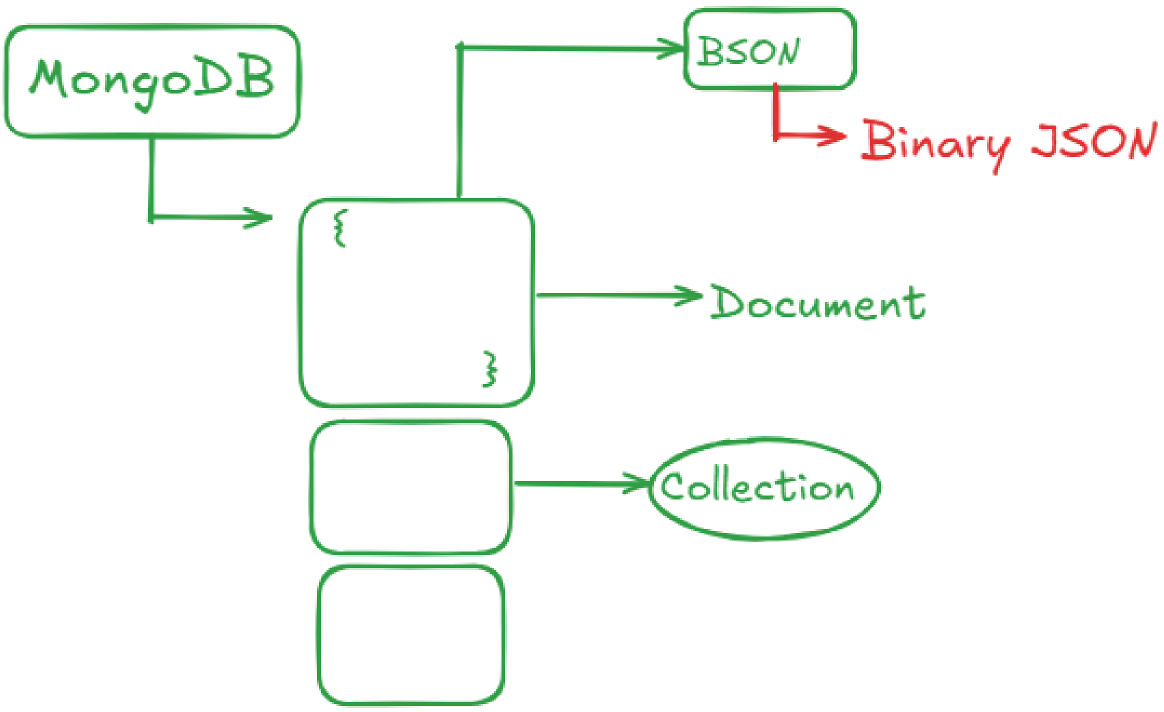
Some Business Rules Enforced by the Schema:

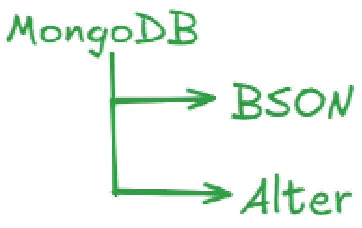
- **Product prices must be positive.**
- **Orders must have a valid customer.**
- **Quantity in stock cannot be negative.**
- **Customers must have a valid address.**

MongoDB

In MongoDB everything is represent like object. Data is stored in BSON(Binary JSON).

- **Documents** are the individual units of data in MongoDB, storing information in a flexible JSON-like structure.
- **Collections** are groups of related documents, similar to tables in relational databases.



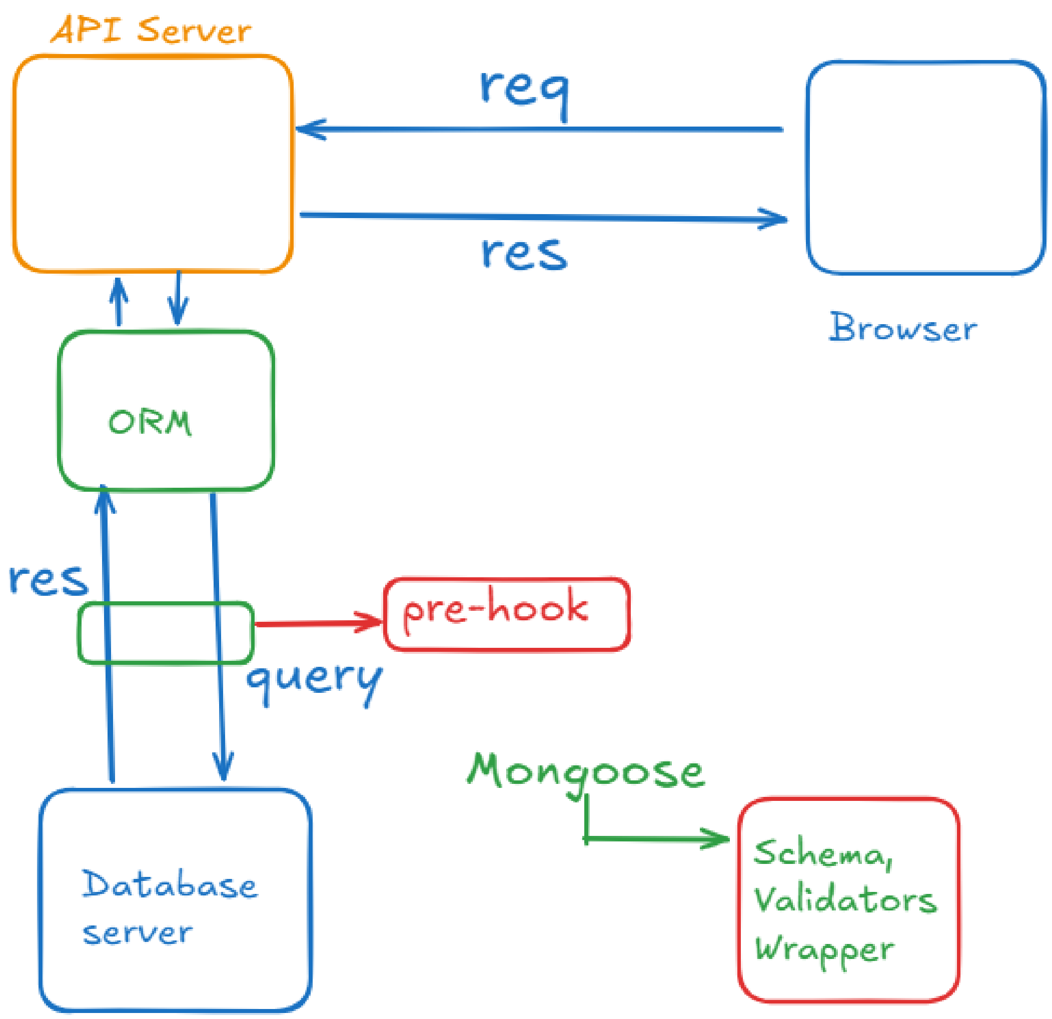


MongoDB store data in BSON format so it take less space and it is too fast in altering the data.

ORM (Mongoose)

Mongoose is an Object-Relational Mapper (ORM) specifically designed for MongoDB, a popular NoSQL database. It provides a layer of abstraction over MongoDB's native driver, making it easier to interact with MongoDB documents and collections in a more object-oriented manner.

It provides a powerful and flexible way to define schemas, enforce business logic, and manage data interactions in a MongoDB database.



Schema Model(validators & constraints)

db.js

```
const schemaRules = {
    name:{
        type: String,
        required: [true,"name is required"],
    },
    email:{
        type : String,
        required: [true,"email is required"],
        unique: [true,"email should be unique"],
    },
    password:{
        type:String,
        required:[true,"password is required"],
        minLength:[6,"password should be atleast of 6 length"],
    },
    confirmPassword:{
        type:String,
        required:true,
        minLength:6,

        // custom validation
        validate:[function(){
            return this.password == this.confirmPassword
        },"password should be equal to confirm password"]
    },
    createdAt:{
        type:Date,
        default:Date.now()
    },
    isPremium:{
        type:Boolean,
        default:false
    },
    role:{
        type:String,
        // these are the only possible values for the role
        enum:["user","admin","curator"],
        default:"user"
    }
}
```

Enum Validator: The `enum` property is used to specify a set of allowed values for the field. The field value must match one of these predefined values.

Model

The model is created using the schema and provides the interface for interacting with the MongoDB collection. It provides methods for performing CRUD(Create, Read, Update, Delete) operations and includes built-in methods for querying and updating documents.

Model are special collection where every document should follow a schema.

db.js

```
const userSchema = new mongoose.Schema(schemaRules);
// final touch point
const UserModel = mongoose.model("user",userSchema);
```

The model acts as the final touchpoint for interacting with our database. It serves as the interface through which all operations—such as creating, reading, updating, and deleting documents—are performed.

POST

```
const express = require("express");
const app = express();

const createUser = async function(req,res){
    try{
        const userObject = req.body;
        const user = await UserModel.create(userObject);
        res.status(201).json(user);
    }catch(err){
        res.status(500).json({
            message :"Internal server error",
            error:err
        })
    }
}
app.use(express.json());
app.post("/user",createUser);

app.listen(3000,function(){
    console.log("server is running at 3000 port");
})
```

Pre-hook & Post-hook

Pre-hook: Before sending to the DB
Post-hook: Return Server

```
const userSchema = new mongoose.Schema(schemaRules);
// final touch point-
userSchema.pre("save",function(next){
    console.log("pre save was called");
    this.confirmPassword = undefined;
    next();
})
userSchema.post("save",function(){
    console.log("post save was called");
    this.password = undefined;
    this.__v= undefined;
})
const UserModel = mongoose.model("user",userSchema);
```