Why Use Mongoose in Express

🔎 1. MongoDB Driver vs Mongoose

🧰 MongoDB Driver (Native)

- Low-level official library.
- Works with plain JavaScript objects.
- ❌ No schema enforcement, manual validation needed.

🦄 Mongoose (ODM Library)

- Sits on top of MongoDB Driver.
- Adds:
  - ✅ Schema & validation
  - ⏳ Middleware (hooks)
  - 📎 Relationships (Population)
  - 📝 Easier CRUD methods

🤔 Why Use Mongoose (Compared to Native Driver)

Feature	MongoDB Driver ❌	Mongoose ✅
Schema enforcement	❌	✅
Data validation	❌	✅
Middleware/hooks	❌	✅
Built-in CRUD	Basic	✅
Relationships	Manual	✅ (via `.populate`)
Type casting	❌	✅

🔑 Key Reasons

- 🛡️ Schema for consistency

- ✅ Validation before saving
- ⏳ Middleware for logic (e.g., password hashing)
- 🔗 Relationships support
- 📝 Easier coding with `.create()`, `.find()`, etc.

🛠️ MVC Setup for Mongoose with Express

```
project-folder/
├── config/
│   └── db.js
├── models/
│   └── Student.js
├── controllers/
│   └── studentController.js
├── routes/
│   └── studentRoutes.js
├── middlewares/
│   └── errorMiddleware.js
├── app.js
```

🔗 Step-by-Step Mongoose Setup (MVC)

Step 1: Install Mongoose

```
Terminal

npm install mongoose express-async-handler
```

Step 2: Database Connection (`config/db.js`)

```
js

const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await
mongoose.connect('mongodb://localhost:27017/mydb')
```

```
    console.log('✅ MongoDB connected!');
  } catch (error) {
    console.error('❌ MongoDB connection error:',
error);
  }
};

module.exports = connectDB;
```

Step 3: Define Schema & Model (`models/User.js`)

```
js

const mongoose = require("mongoose");

const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  age: { type: Number, min: 0, max: 120 },
});

const Student = mongoose.model("Student",
studentSchema);

module.exports = Student;
```

🔷 Step 4: Controller with `asyncHandler` (`controllers/studentController.js`)

```
Js

const Student = require("../models/Student");
const asyncHandler = require("express-async-
handler");

// Create a student
exports.createStudent = asyncHandler(async (req, res)
=> {
  const student = await Student.create(req.body);
  res.status(201).json(student);
});
```

```
// Get all students
exports.getStudents = asyncHandler(async (req, res)
=> {
  const students = await Student.find();
  res.json(students);
});

// Get student by ID
exports.getStudentById = asyncHandler(async (req,
res) => {
  const student = await
Student.findById(req.params.id);
  if (!student) {
    res.status(404);
    throw new Error("Student not found");
  }
  res.json(student);
});

// Update student
exports.updateStudent = asyncHandler(async (req, res)
=> {
  const student = await
Student.findByIdAndUpdate(req.params.id, req.body, {
    new: true,
  });
  if (!student) {
    res.status(404);
    throw new Error("Student not found");
  }
  res.json(student);
});

// Delete student
exports.deleteStudent = asyncHandler(async (req, res)
=> {
  const student = await
Student.findByIdAndDelete(req.params.id);
  if (!student) {
    res.status(404);
    throw new Error("Student not found");
  }
```

```
  res.json({ message: "Student deleted successfully"
});
});
```

Step 5: Define Routes (`routes/userRoutes.js`)

```
js

const express = require("express");
const router = express.Router();
const studentController =
require("../controllers/studentController");

router.post("/", studentController.createStudent);
router.get("/", studentController.getStudents);
router.get("/:id", studentController.getStudentById);
router.put("/:id", studentController.updateStudent);
router.delete("/:id",
studentController.deleteStudent);

module.exports = router;
```

Step 6: Error Handling Middleware (`middlewares/errorMiddleware.js`)

```
js
const errorMiddleware = (err, req, res, next) => {
  const statusCode = res.statusCode === 200 ? 500 :
res.statusCode;
  res.status(statusCode).json({
    message: err.message || "Internal Server Error",
  });
};

module.exports = errorMiddleware;
```

🔷 **Step 7: App Setup (`app.js`)**

```js
const express = require("express");
const connectDB = require("./config/db");
const studentRoutes =
require("./routes/studentRoutes");
const errorMiddleware =
require("./middlewares/errorMiddleware");

const app = express();
connectDB();

app.use(express.json());
app.use("/api/students", studentRoutes);

// Error Handler Middleware
app.use(errorMiddleware);

app.listen(3000, () => {
  console.log("🚀 Server running at
http://localhost:3000");
});
```

🧠 Mongoose Concepts Recap

Term	Description
🧩 Schema	Blueprint: structure, validation, defaults
🦄 Model	Usable object for DB actions (`find`, `create`)
🔗 Middleware	Hooks like `pre('save')`, `post('remove')`
🎯 Populate	Reference documents in other collections

✨ Advantages of Mongoose in Detail

Advantage	Description
🛡️ Schema	Enforces structured data

Validation	Auto-checks data before saving
Middleware	Pre/post logic like hashing passwords
Relationships	`.populate()` to fetch linked data from other collections
Type casting	Auto converts types (e.g., string to number)
Defaults	Sets fallback values
Built-in CRUD	Easy to use: `.find()`, `.create()`
Query Helpers	Custom helper methods for advanced querying