SECTION 4 — Server Setup and MongoDB Connection

Understanding server.js, Database Configuration, and MongoDB Atlas

1. Purpose of server.js

The server.js file is the entry point of the backend application.

It is responsible for:

- Creating the Express application
- Connecting to the database
- Registering global middleware
- Registering routes
- Starting the server

All incoming requests begin execution from this file.

2. Basic Structure of server.js

A simplified version:

```
import express from "express";
import dotenv from "dotenv";
import connectDB from "./config/db.js";

dotenv.config();

const app = express();

app.use(express.json());

connectDB();

app.listen(process.env.PORT, () => {
  console.log(`Server running on port ${process.env.PORT}`);
});
Each section of this file has a specific responsibility.
```

3. Environment Variables (dotenv)

We use the dotenv package to load configuration from a .env file.

Example .env file:

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/monsta
JWT_SECRET=supersecretkey
```

We access values using:

process.env.PORT

process.env.MONGO_URI

Why environment variables are important:

- Prevents hardcoding sensitive information
- Allows different configuration for development and production
- Improves security and deployment flexibility

4. Creating the Express Application

```
import express from "express";

const app = express();
```

This creates the Express server instance.

All middleware and routes are attached to this app object.

5. Global Middleware Registration

```
app.use(express.json());
```

This middleware enables parsing of JSON request bodies.

Without this, req.body would be undefined.

Other common middleware:

```
app.use(cors());
app.use(cookieParser());
```

Middleware is executed in the order it is declared.

6. Connecting to MongoDB

Database connection logic is separated into:

config/db.js

This keeps server logic clean.

Example: config/db.js

```
import mongoose from "mongoose";

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("MongoDB Connected");
  }
catch (error) {
    console.error("Database connection failed:", error.message);
    process.exit(1);
  }
};

export default connectDB;
```

Why Use Mongoose?

Mongoose provides:

- Schema definition
- Data validation
- Query abstraction
- Middleware support
- Structured models

It makes MongoDB easier to work with in large applications.

7. MongoDB Local vs MongoDB Atlas

There are two ways to connect MongoDB:

1. Local MongoDB installation
2. MongoDB Atlas (Cloud database)

8. What Is MongoDB Atlas?

MongoDB Atlas is a cloud-hosted MongoDB service.

Instead of installing MongoDB on your system, Atlas:

- Hosts your database in the cloud
- Provides automatic backups
- Handles scaling
- Provides security configuration
- Allows remote access

Atlas is recommended for:

- Production applications
- Collaborative development
- Cloud deployment

9. Setting Up MongoDB Atlas (Student Guide)

Step 1 — Create Account

Go to:

https://www.mongodb.com/atlas

Create a free account.

Step 2 — Create a Cluster

- Choose "Build a Cluster"
- Select Free Tier
- Choose Cloud Provider
- Select Region
- Create cluster

Step 3 — Create Database User

Go to:

Database Access → Add New Database User

Set:

- Username
- Password

Example:

Username: monstaUser

Password: Monsta@123

Step 4 — Configure Network Access

Go to:

Network Access → Add IP Address

For development:

- Add 0.0.0.0/0 (allows access from anywhere)

For production:

- Use specific IP addresses

Step 5 — Get Connection String

Go to:

Cluster → Connect → Drivers

You will see something like:

mongodb+srv://username:password@cluster0.mongodb.net/?retryWrites=true&w=majority

Replace:

username

password

Example:

mongodb+srv://monstaUser:Monsta@123@cluster0.mongodb.net/monsta

Step 6 — Add to .env

MONGO_URI=mongodb+srv://monstaUser:Monsta@123@cluster0.mongodb.net/monsta

Now the backend connects to Atlas.

10. Local MongoDB vs Atlas

Feature	Local MongoDB	MongoDB Atlas
Installation	Required	Not required
Cloud Hosted	No	Yes
Production Ready	Limited	Yes
Backup	Manual	Automatic
Scalability	Manual	Built-in

For learning:

- Local is simple.
- Atlas is more production-like.

11. Starting the Server

```
app.listen(process.env.PORT, () => {
  console.log(`Server running on port ${process.env.PORT}`);
});
```

Server runs at:

http://localhost:5000

If using Atlas, database is remote but server remains local.

12. Complete Structured Example

```
import express from "express";
import dotenv from "dotenv";
import cors from "cors";
import cookieParser from "cookie-parser";
import connectDB from "./config/db.js";
import errorMiddleware from "./middlewares/error.middleware.js";
import authRoutes from "./modules/identity/auth.routes.js";
```

```
dotenv.config();
connectDB();

const app = express();

app.use(express.json());
app.use(cors());
app.use(cookieParser());

app.use("/api/auth", authRoutes);

app.use(errorMiddleware);

app.listen(process.env.PORT, () => {
  console.log(`Server running on port ${process.env.PORT}`);
});
```

13. Important Architecture Notes

1. Environment variables must be loaded before database connection.
2. Database connection must happen before starting the server.
3. Error middleware must be placed last.
4. Configuration must be separated from logic.

14. Request Lifecycle from server.js

```
Incoming Request
  ↓
Global Middleware
  ↓
Route
  ↓
Controller
  ↓
Service
  ↓
Database
  ↓
Response
```

server.js acts as the application bootstrapper.

15. Summary

By the end of this section, students should understand:

- Role of server.js
- Why environment variables are used
- How MongoDB connects using Mongoose
- Difference between local DB and Atlas
- How Atlas is configured
- Why configuration is separated

This establishes a strong backend initialization foundation.