Day 20 – [20th July 2025]

TOPICS COVERED

Exporting Models in Mongoose:

Mongoose allows us to define schemas for MongoDB collections. In real-world projects, we modularize our code by separating schema/model definitions from our main server file.

We use module.exports to export models so that they can be reused anywhere.

```
models/userModel.js
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String
});
const User = mongoose.model("User", userSchema);
module.exports = User;
```

```
index.js or routes/userRoutes.js
const User = require("./models/userModel"); // Now we can use User.find(), save(), etc.
```

Why important?

This makes our code clean, reusable, and scalable — especially when working in teams or bigger apps.

HTTP Status Codes (2xx, 4xx, 5xx):

HTTP messages tell us how a request/response cycle went.

Code	Category	Description
200	✅ Success	Request succeeded
201	✅ Created	Resource was successfully created
400	✕ Bad Request	Client sent invalid data

401	✕ Unauthorized	No/invalid auth credentials
404	✕ Not Found	Resource does not exist
500	✕ Server Error	Server-side failure

Example:

res.status(400).json({ error: "Email is required" });

res.status(201).send("User created successfully");

Why important?

These messages help frontend and backend communicate clearly. The frontend can show proper messages based on the status code returned.

Hashing vs Encryption:

Hashing:

- One-way transformation (cannot be reversed)
- Used for storing passwords securely

Example: bcrypt.hash("mypassword") → returns a hashed string

Encryption:

- Two-way (can be decrypted back to original)
- Used for sensitive information like tokens, messages, etc.

bcrypt & Salt Rounds:

- bcrypt is a secure way to hash passwords before saving them to the database.

Installed via:

npm install bcrypt

- A salt adds randomness before hashing so that two same passwords don’t produce the same hash.
- Salt rounds = number of iterations = more secure but slower.

Example – Password Hashing

const bcrypt = require("bcrypt");

```
const saltRounds = 10;
async function registerUser(req, res) {
  try {
    const hashedPassword = await bcrypt.hash(req.body.password, saltRounds);
    const newUser = new User({
      name: req.body.name,
      email: req.body.email,
      password: hashedPassword,
    });
    await newUser.save();
    res.status(201).send("User registered securely");
  } catch (err) {
    res.status(500).send("Something went wrong");
  }
}
```

Why important?

Hashing ensures that even if your database is compromised, user passwords are not exposed.

TOOLS USED:

VS Code

Express.js

MongoDB Atlas

Mongoose (ODM to interact with MongoDB using JavaScript)

bcrypt (Library to hash and secure passwords)

Nodemon

Hoppscotch

TASK:

Read about tokens, cookies, http header, authentication, authorisation