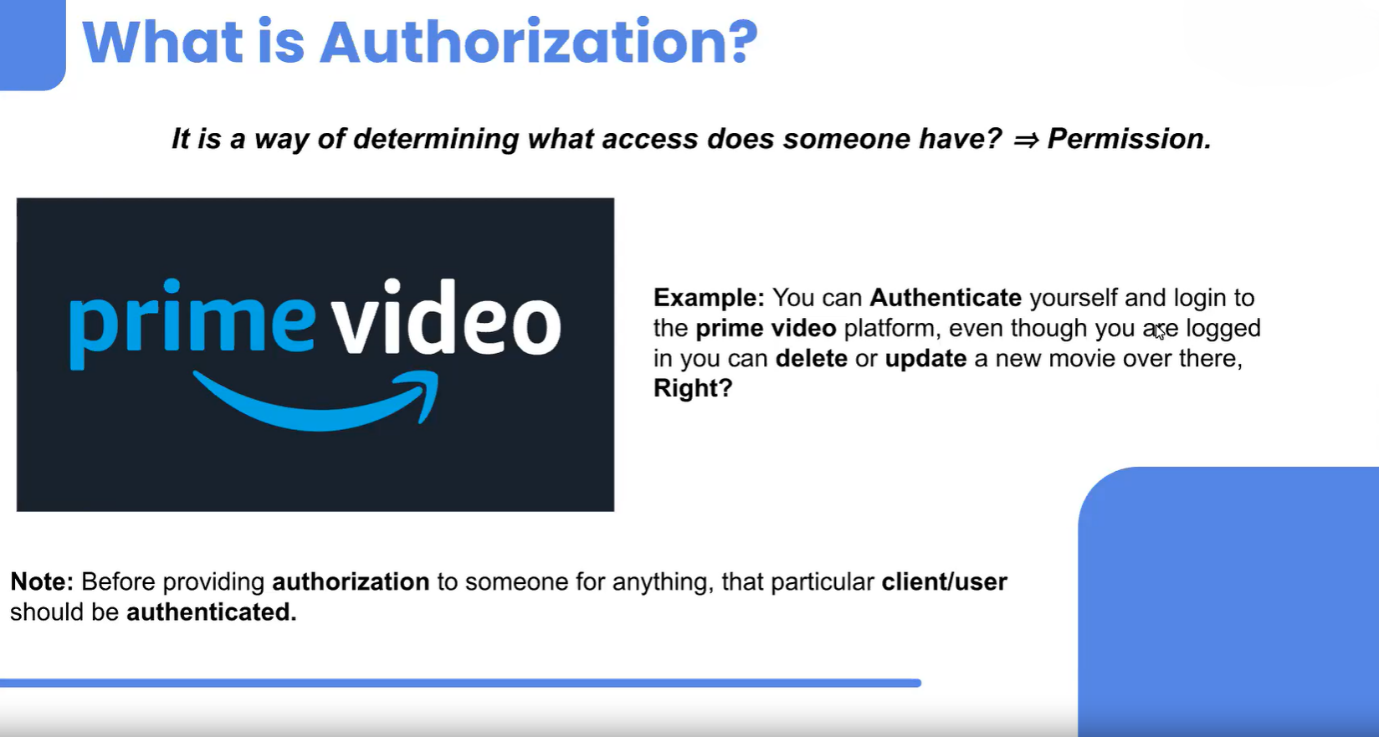
## Page:1

# Authentication-1



Authentication is the process of verifying who someone is. It's all about **identification**.



Authorization determines what access someone has. It's all about **permission**.

## Page:2

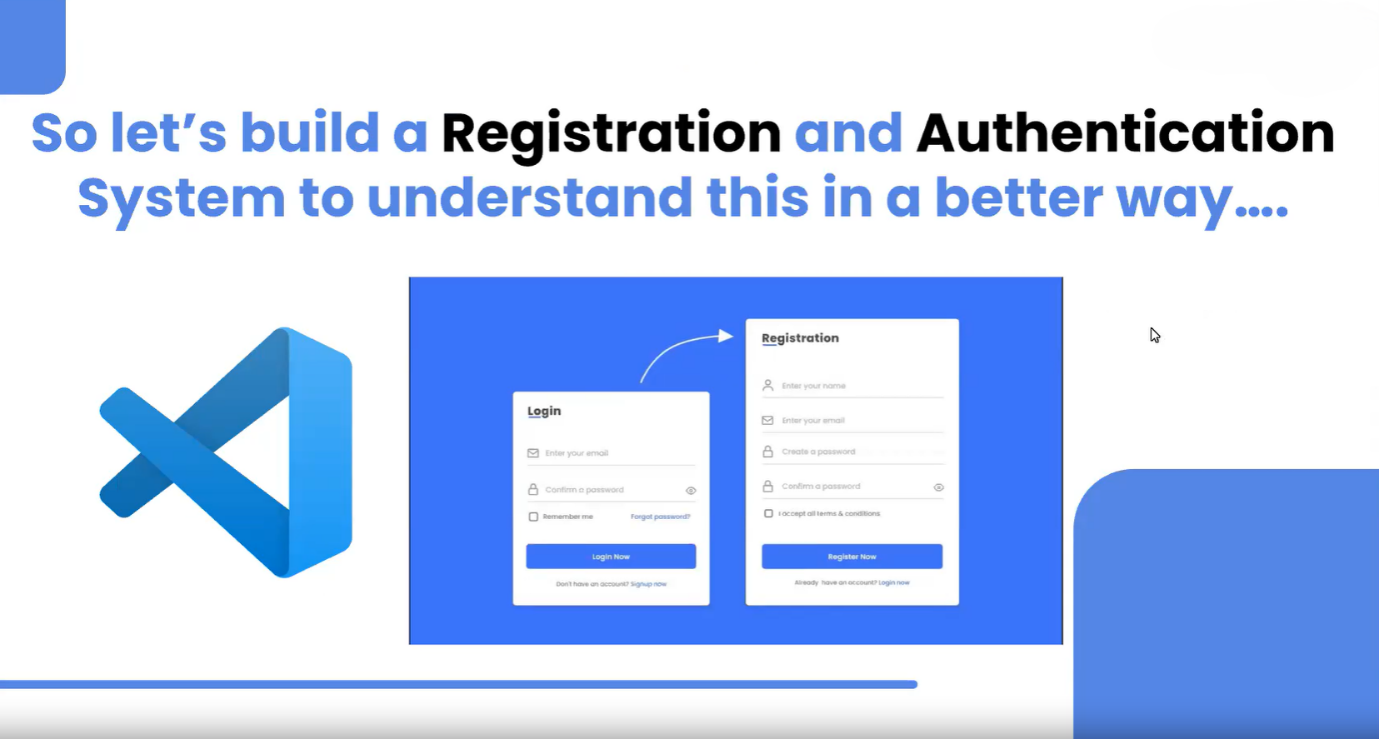
**🛒 Example: Amazon**

* **Authentication:** You cannot purchase a product on Amazon without being authenticated (logged in).
* **Authorization:** Even if you are authenticated, you cannot update or delete product details.

**🔑 Common Authentication Steps**

1. **Signup/Register/Create Account**
2. **Login**

During login, the system checks if the **email** and **password** are correct by matching them with the data stored in the database.



**🚀 Create a Backend Project**

To get started, install the necessary packages:

bashCopy code

npm i mongoose express nodemon dotenv

# 🖥️ server.js

javascriptCopy code

const express = require("express");

## Page:3

const { connection } = require("./config/db");

const app = express();

app.use(express.json());

app.post("/register", async (req, res) => {

try {

const user = new UserModel(req.body);

await user.save();

res.send("Registered");

} catch (err) {

console.log(err);

res.send("Error in registering the user");

}

});

app.post("/login", async (req, res) => {

const { email, pass } = req.body;

try {

const user = await UserModel.find({ email, pass });

if (user.length > 0) {

res.send("Login Successful");

} else {

res.send("Login Failed");

}

} catch (err) {

console.log(err);

res.send("Something went wrong");

}

## Page:4

});

app.listen(8080, async () => {

try {

await connection;

console.log("Running at Port 8080");

} catch (err) {

console.log(err);

}

});

# 📂 config/db.js

javascriptCopy code

const mongoose = require("mongoose");

const connection = mongoose.connect("mongodb\_url");

module.exports = { connection };

# 📄 models/User.model.js

const userSchema = mongoose.Schema({

email: String,

pass: String,

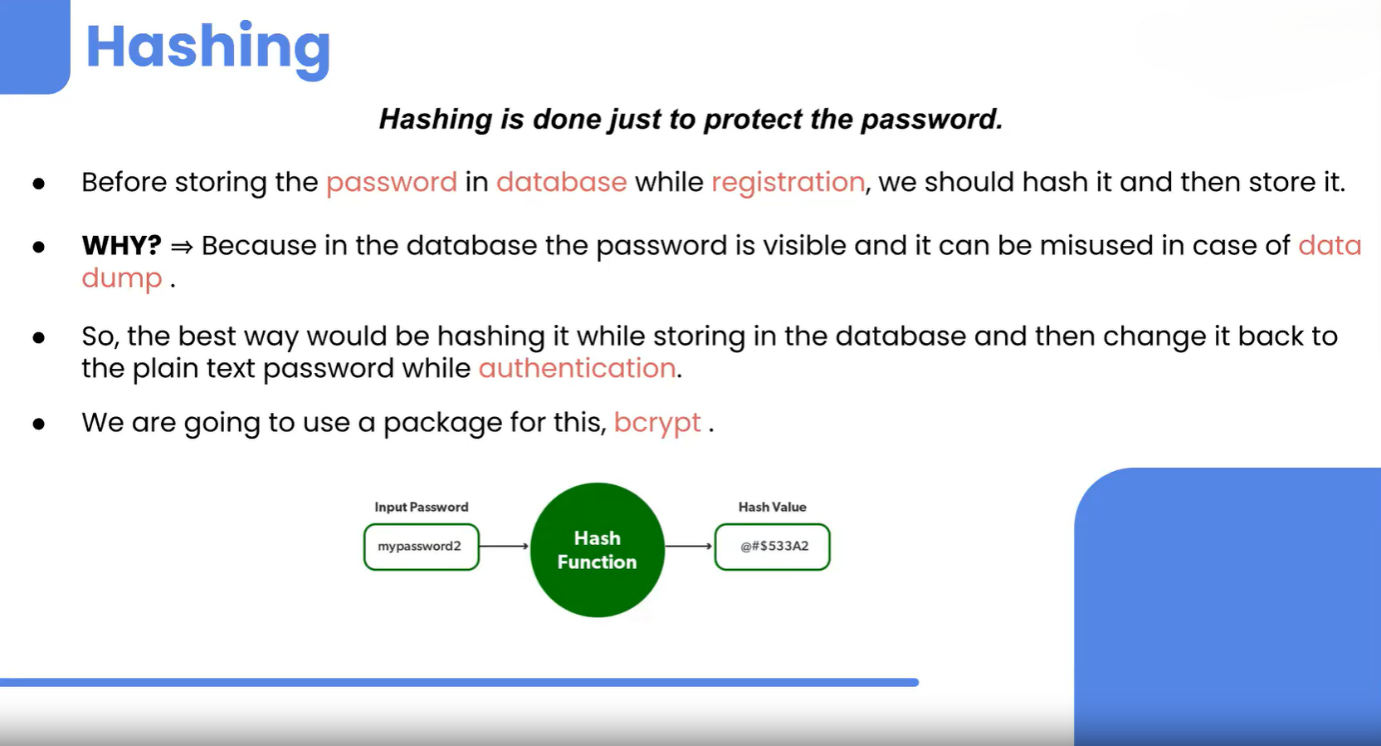
name: String,

age: Number

});

const UserModel = mongoose.model("user", userSchema);

## Page:5



# 🔐 Hashing

Hashing is used to protect passwords. Before storing a password in the database during signup, we should hash it. This prevents passwords from being visible in the database and reduces the risk of misuse in case of a data breach.

# 🧂 Why Hashing?

* Passwords are visible in the database, which can be a security risk.
* The best practice is to hash the password before storing it and compare the hashed password during login.

# 🔒 Bcrypt

Bcrypt is a password hashing function designed to be secure and resistant to attacks like brute-force and rainbow table attacks. It is widely used in web applications to store user passwords securely.

* **One-way hash:** Bcrypt creates a fixed-length output that cannot be reversed to obtain the original password.
* **Cost factor:** Bcrypt applies a cost factor to slow down the hashing process, making it harder for attackers to try out large numbers of passwords.

## Page:6

# 🧰 **Implementing Bcrypt**

**Registration:**

app.post("/register", async (req, res) => {

const { name, email, pass, age } = req.body;

try {

bcrypt.hash(pass, 8, async (err, hash) => {

const user = new UserModel({ name, email, pass: hash, age });

await user.save();

res.send("Registered");

});

} catch (err) {

res.send("Error in registering the user");

console.log(err);

}

});

# Login:

app.post("/login", async (req, res) => {

const { email, pass } = req.body;

try {

const user = await UserModel.find({ email });

if (user.length > 0) {

bcrypt.compare(pass, user[0].pass, function (err, result) {

if (result) {

const token = jwt.sign({ course: 'backend' }, 'masai');

res.send({ "msg": "Login Successful", "token": token });

## Page:7

} else {

res.send("Wrong Credentials");

}

});

} else {

res.send("Wrong Credentials");

}

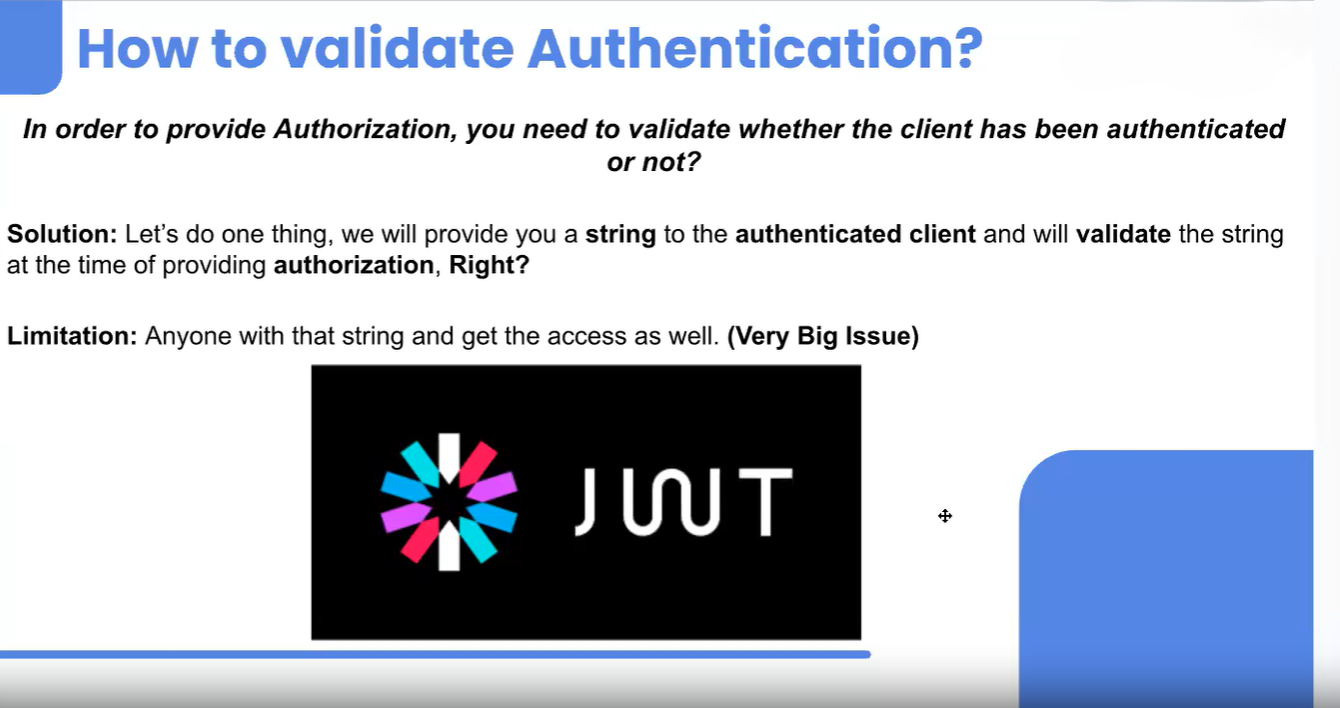
} catch (err) {

res.send("Something went wrong");

console.log(err);

}

});



# 🛡️ Authorization

Now that users can register and log in, we need to create protected endpoints.

## Page:8

# 🔒 Protecting Endpoints

We can use two methods:

1. **Login at protected endpoints:** Send email and password with each request when accessing the endpoint. This method is not ideal as it can be annoying and inefficient.
2. **Access Token:** Issue a token upon login and validate it when accessing restricted endpoints.

# Example:

app.post("/login", async (req, res) => {

const { email, pass } = req.body;

try {

const user = await UserModel.find({ email, pass });

if (user.length > 0) {

res.send({ "msg": "Login Successful", "token": "abc123" });

} else {

res.send("Login Failed");

}

} catch (err) {

console.log(err);

}

});

app.get("/data", (req, res) => {

if (req.query.token === "abc123") {

res.send("Logged in and can access the data");

} else {

res.send("Not Logged in, login first");

## Page:9

}

});

app.get("/cart", (req, res) => {

if (req.query.token === "abc123") {

res.send("Logged in and can access the cart");

} else {

res.send("Not Logged in, login first");

}

});

# 🔑 JWT: JSON Web Token

JWT is a more secure and efficient way to handle authorization. It consists of three parts:

1. **Algorithm:** Used to encrypt the string.
2. **Payload:** The data to be encrypted in the string.
3. **Secret Key:** Used to decrypt it.

# Install JWT:

npm i jsonwebtoken

# Usage:

const jwt = require("jsonwebtoken");

const token = jwt.sign({ "course": "backend" }, "masai");

This token is used in restricted routes.

# Example:

var token = req.headers.authorization;

jwt.verify(token, "batman", (err, decoded) => {

if (err) {

res.send("Invalid Token");

## Page:10

console.log(err);

} else {

res.send("Whatever you want can be sent over here");

}

});

**🌐 Resources**

1. [JWT Introduction](https://jwt.io/introduction)
2. [jsonwebtoken npm package](https://www.npmjs.com/package/jsonwebtoken)
3. [bcrypt npm package](https://www.npmjs.com/package/bcrypt)