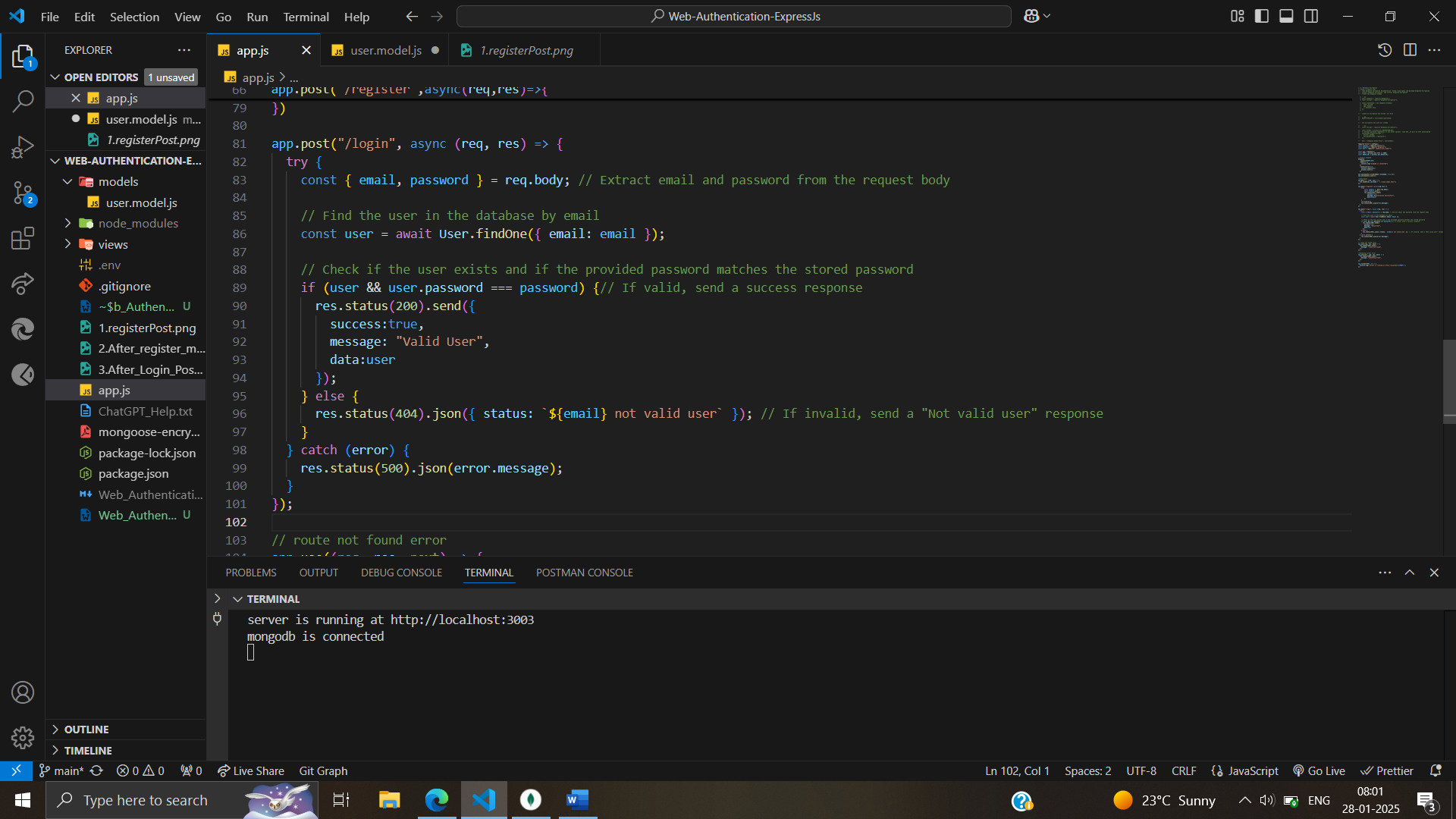
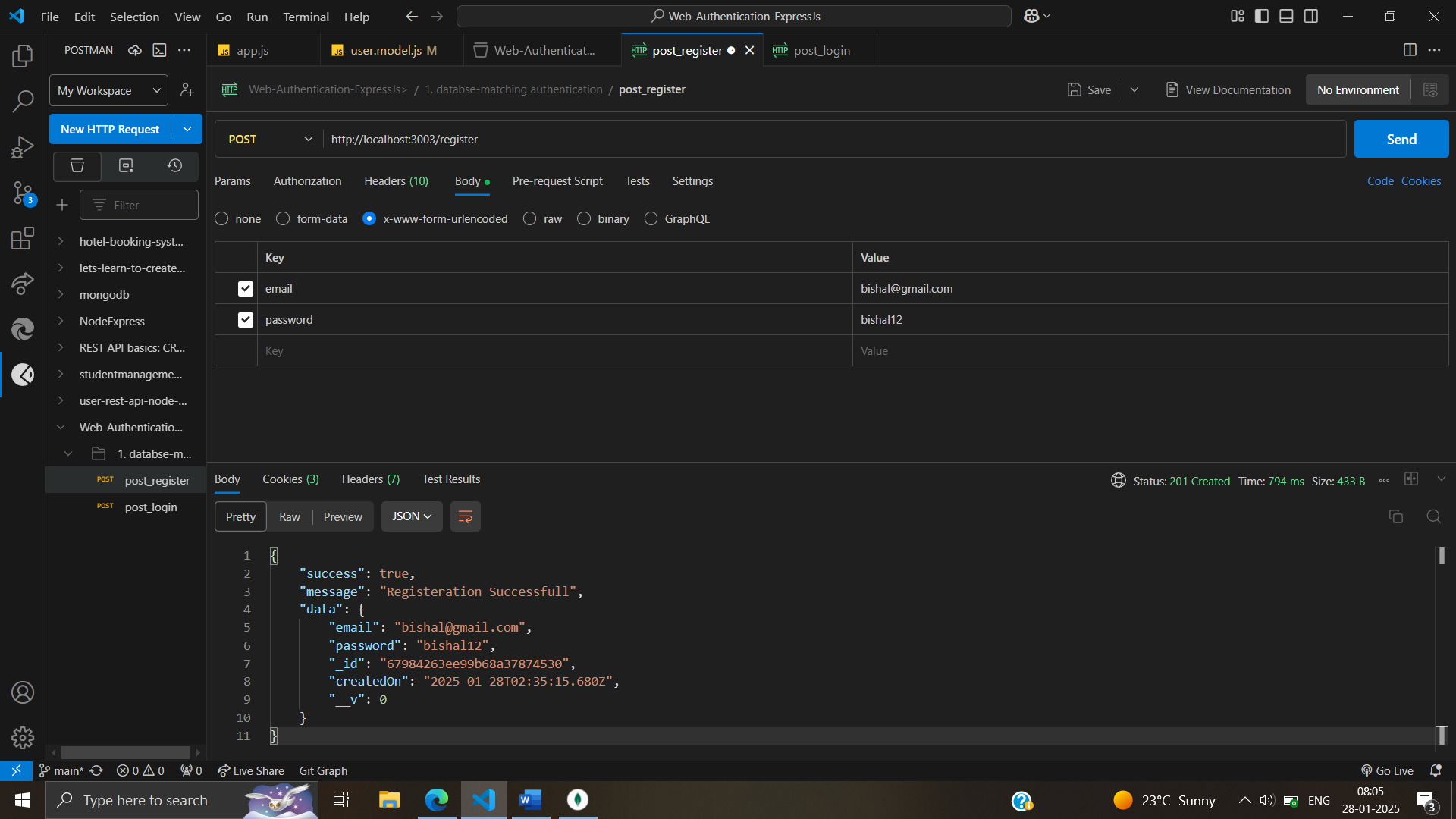
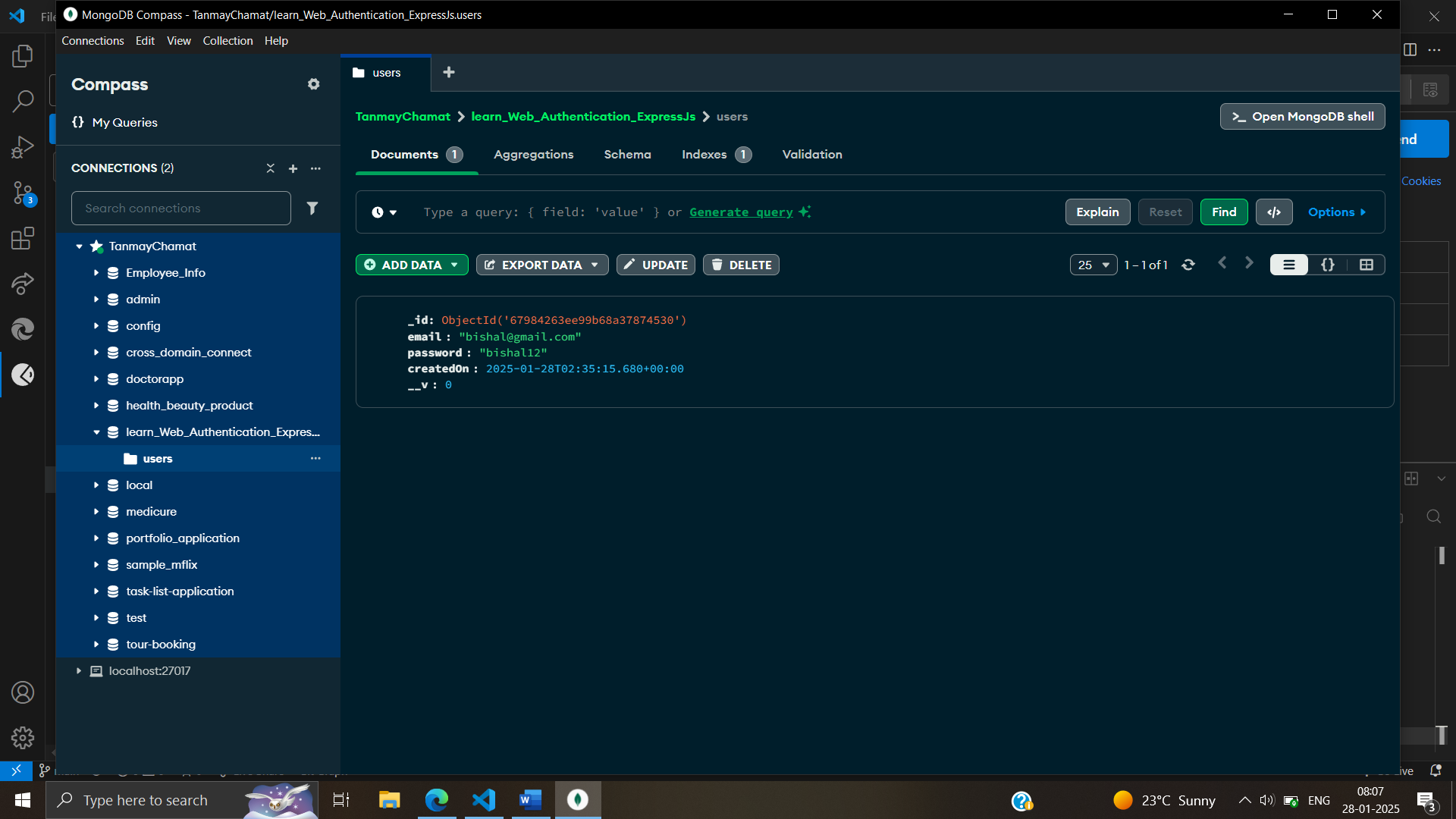
**Web authentication**

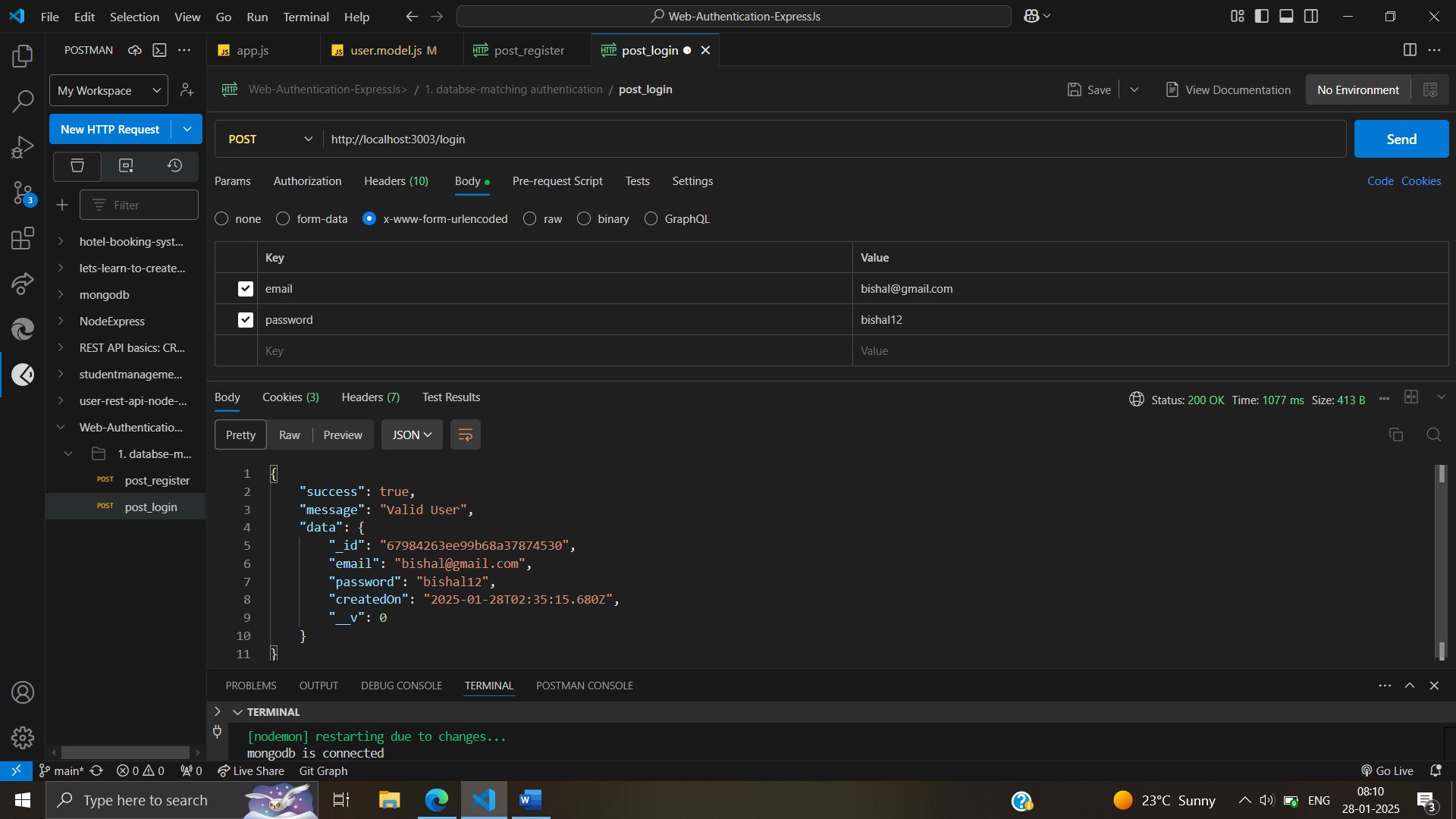
Web authentication (WebAuthn) is a modern, secure standard for authenticating users on websites and apps without relying on traditional passwords. It uses biometrics (like fingerprints or facial recognition), hardware security keys, or device-based methods to verify user identity. WebAuthn enhances security by reducing risks of phishing, password leaks, and brute-force attacks.

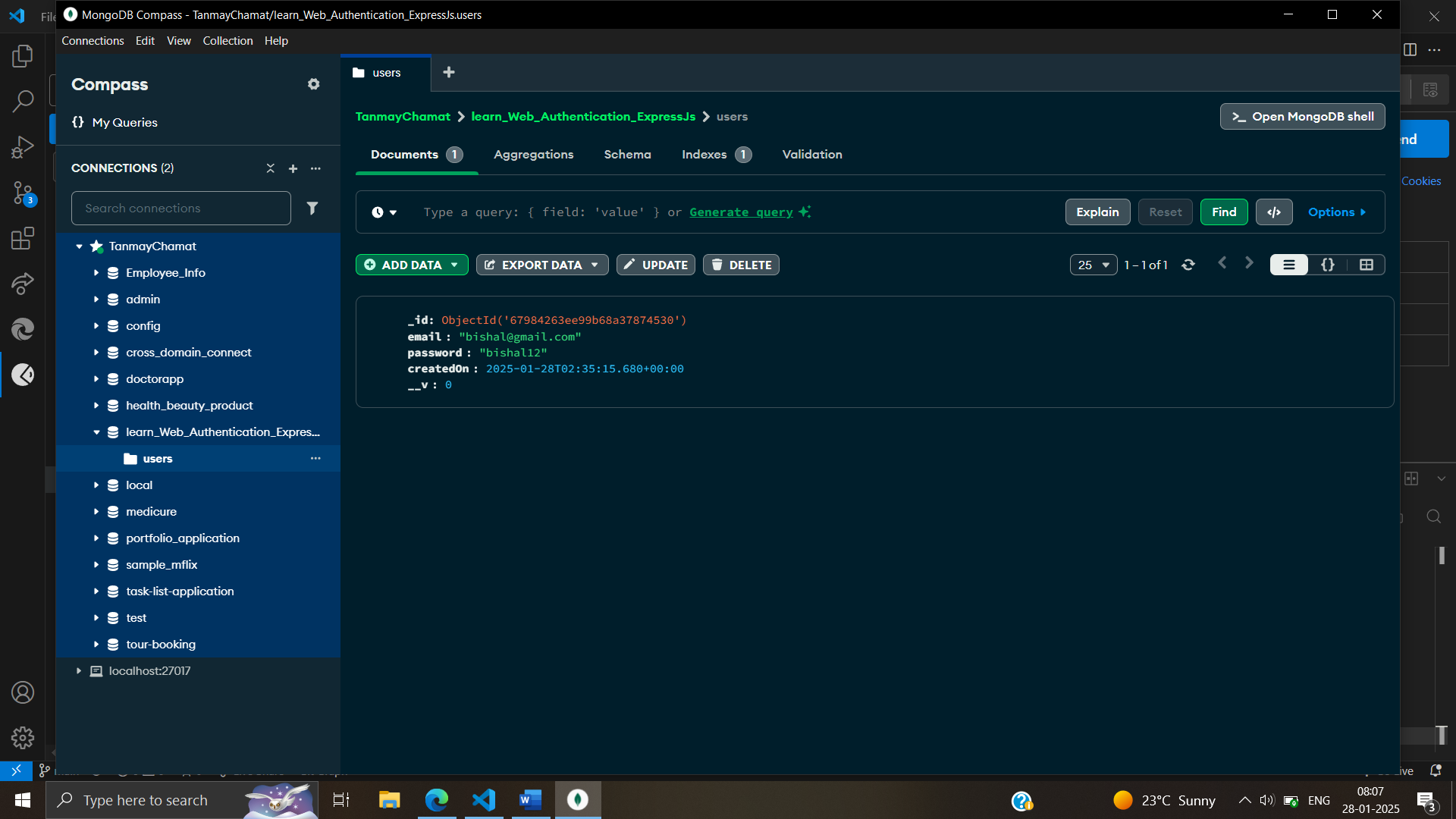
**1: Database matching :**

 **Validate Credentials**:

* **Successful Login**: If a user is found and the provided password matches the stored password, a success response (200) is sent. The response includes:
  + success: true (indicating success)
  + message: "Valid User"
  + data: user (the user's details from the database)
* **Invalid Login**: If either the user is not found or the password doesn't match, an error response (404) is sent with the message <email> not valid user.

<-





**- if hacker can access our database then our data is too much human readable**

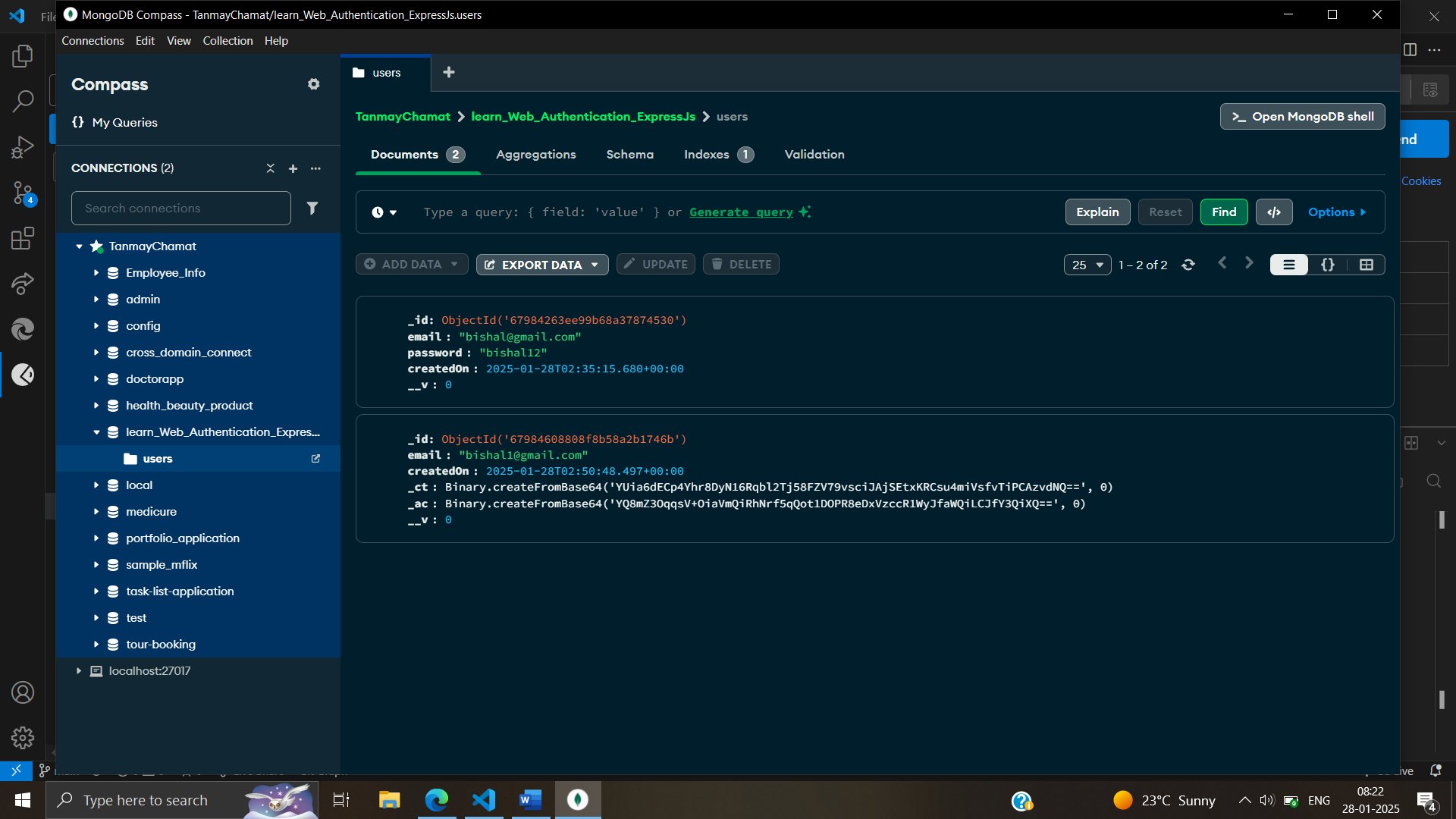
**- [password checker online] (http://password-checker.online-domain-tools.com/)**

**Improvements for Security:**

* **Hashing Passwords: Always hash passwords (e.g., using bcrypt) instead of storing and comparing them in plain text.**
* **Error Messaging: Avoid exposing detailed error messages to prevent attackers from knowing whether the email or password was incorrect.**

**2: Database Encryption**

Database encryption is the process of converting data in a database into a secure format (ciphertext) to protect it from unauthorized access. It ensures that sensitive information, like passwords or personal details, remains confidential even if the database is compromised. Decryption keys are required to access the original data, enhancing overall security. [Modular conversion, encoding and encryption online - cryptii](https://cryptii.com/)



**- read mongoose encryption documentation: https://www.npmjs.com/package/mongoose-encryption**

**- install mongoose encryption `npm install mongoose-encryption`**

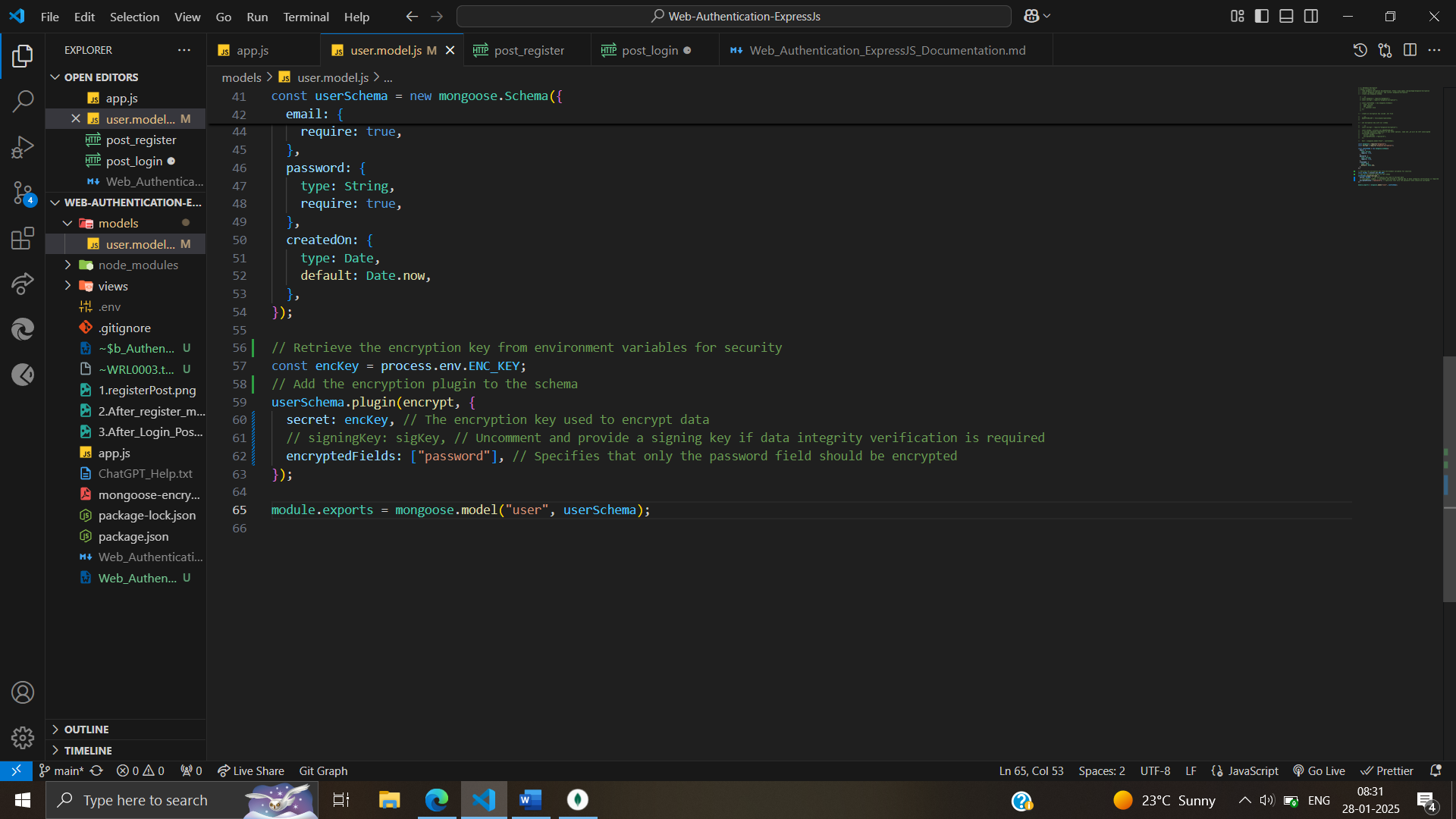
**- create new mongoose Schema**

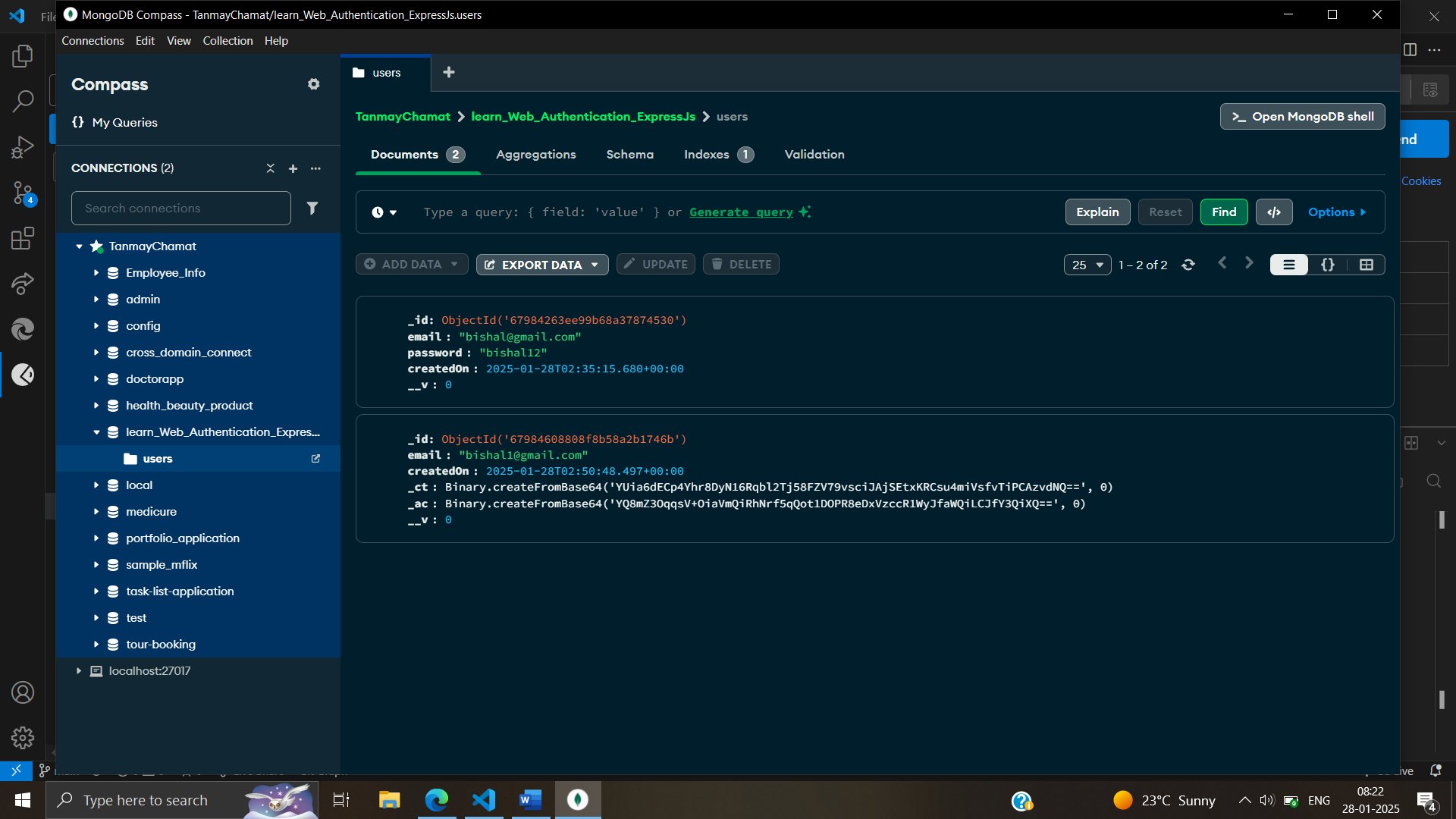
**ENCRYPTION\_KEY = thisismyencryptionkey;**

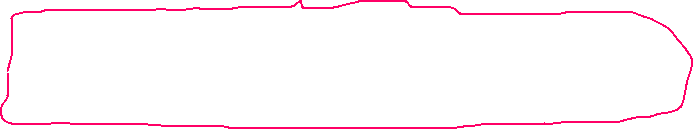
**- set encryption key with our schema**

**const encrypt = require("mongoose-encryption");**

**const encKey = process.env.ENCRYPTION\_KEY;**







**\*hackers can convert to plain text if he got the encryption key.**

**3: Hashing password**

**Hashing passwords is a process of converting plain text passwords into a fixed-length string of characters (hash) using a cryptographic algorithm. It is a one-way process, meaning the original password cannot be retrieved from the hash. This ensures secure storage of passwords and protects them from being exposed in case of a data breach. Common algorithms include bcrypt, SHA-256, and Argon2.**

**- no cncryption key; we will use hashing algorithm**

**- hackers can not convert to plain text as no encryption key is available**

**- md5 package: https://www.npmjs.com/package/md5**

**- install md5 npm package: `npm install md5`**

**- usage**

**```js**

**var md5 = require("md5");**

**console.log(md5("message"));**

**// 78e731027d8fd50ed642340b7c9a63b3**

**// hash password when create it**

**const newUser = new User({**

**email: req.body.username,**

**password: md5(req.body.password),**

**});**

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

**try {**

**const email = req.body.email;**

**const password = md5(req.body.password);**

**const user = await User.findOne({ email: email });**

**if (user && user.password === password) {**

**res.status(200).json({ status: "valid user" });**

**} else {**

**res.status(404).json({ status: "Not valid user" });**

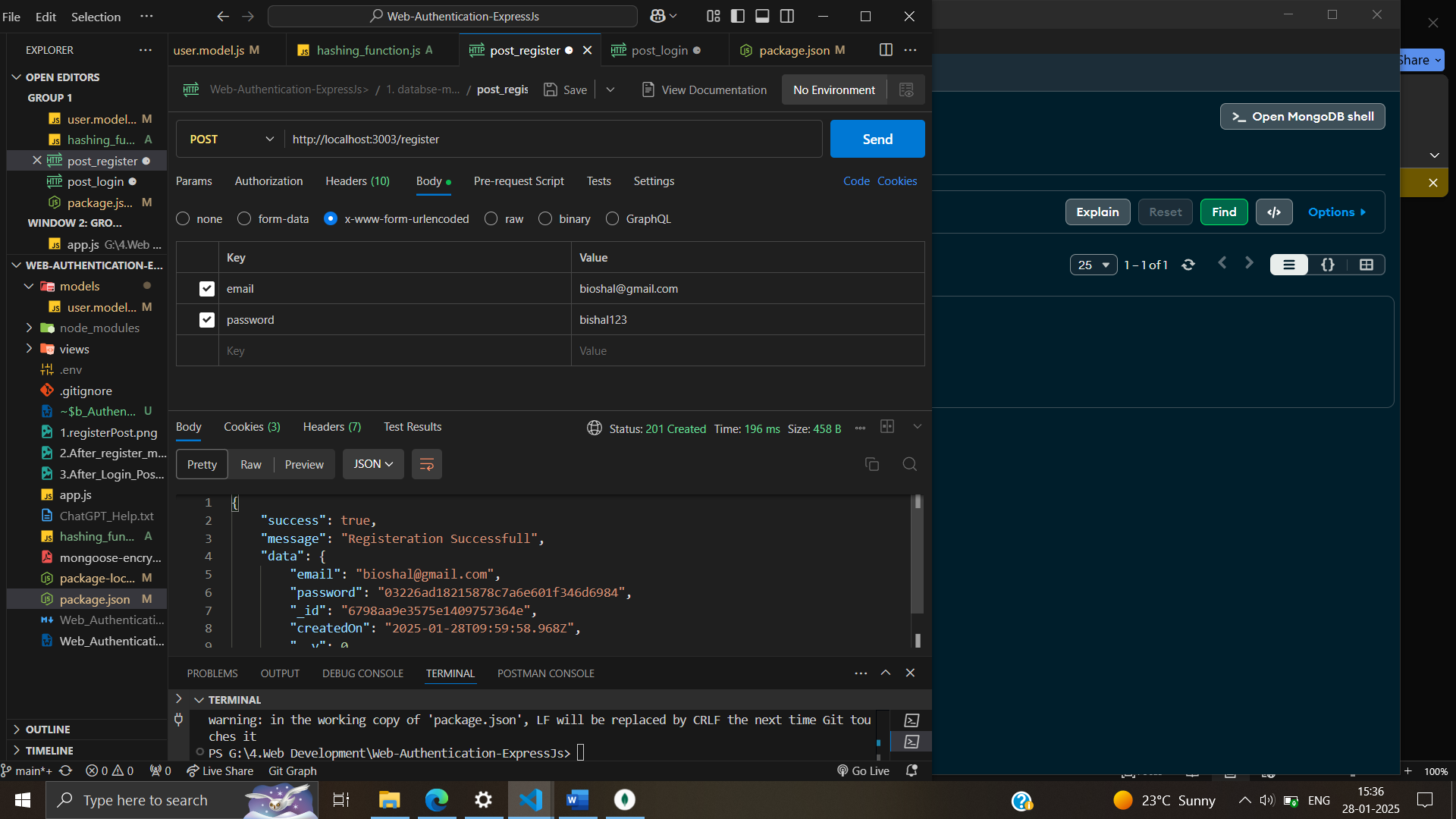
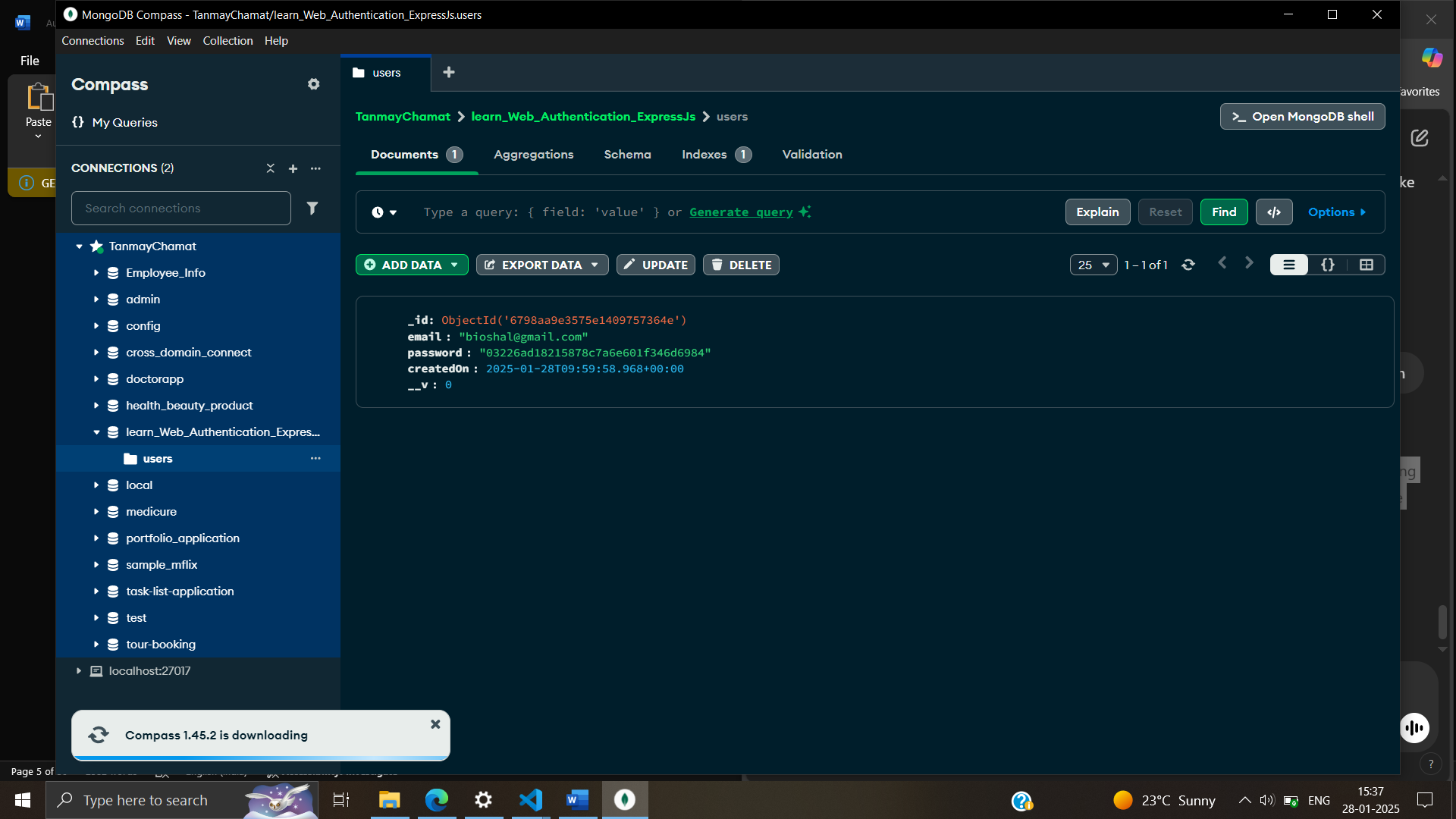
**}**

**} catch (error) {**

**res.status(500).json(error.message);**

**}**

**});**

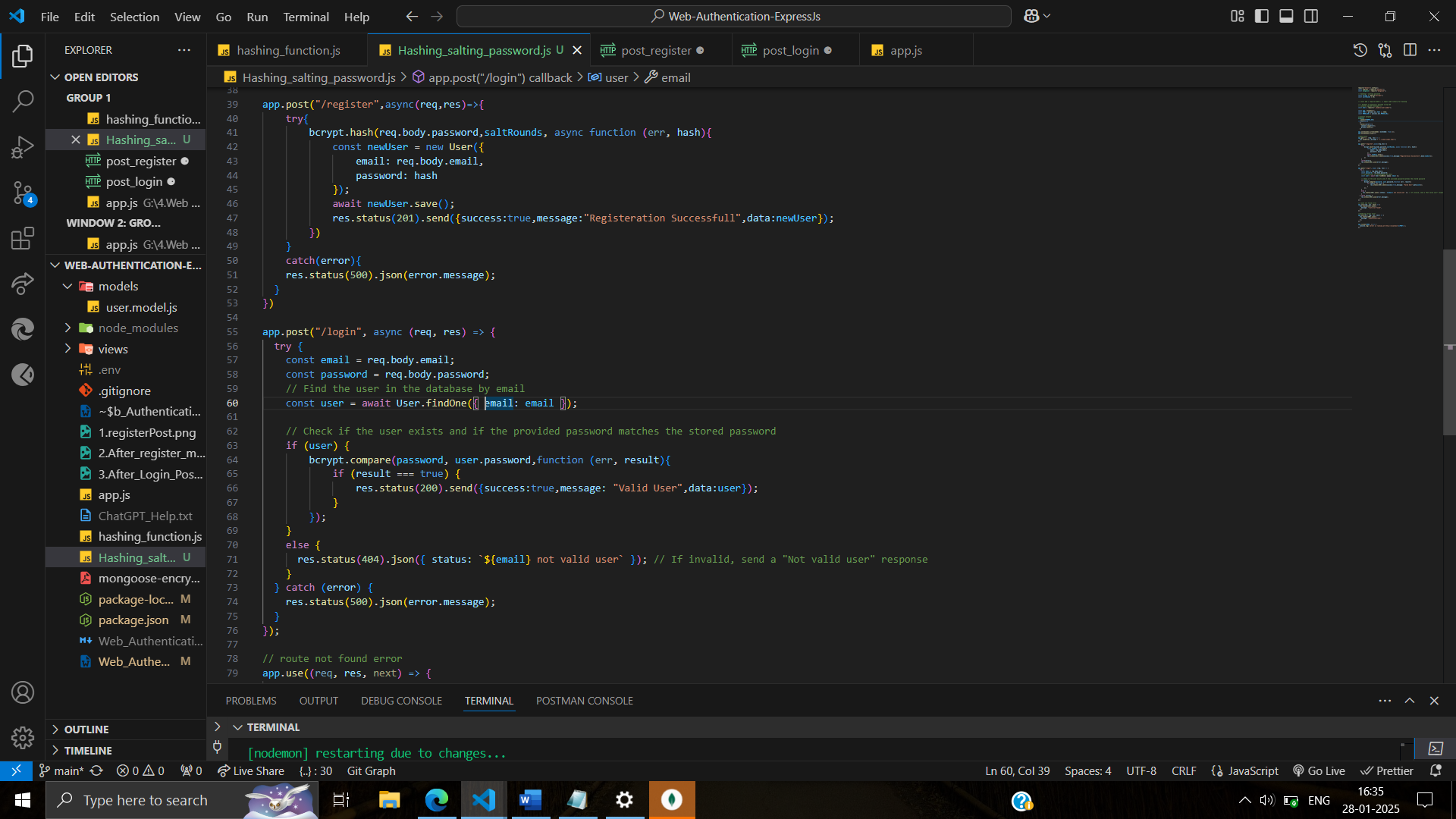
 

**4: Hashing + salting password:**

**Hashing transforms a password into a fixed-length encrypted value for secure storage. Salting adds a unique random string to each password before hashing to make the hash unique, even for identical passwords. bcrypt combines both processes, ensuring strong protection against brute force, dictionary, and rainbow table attacks.**

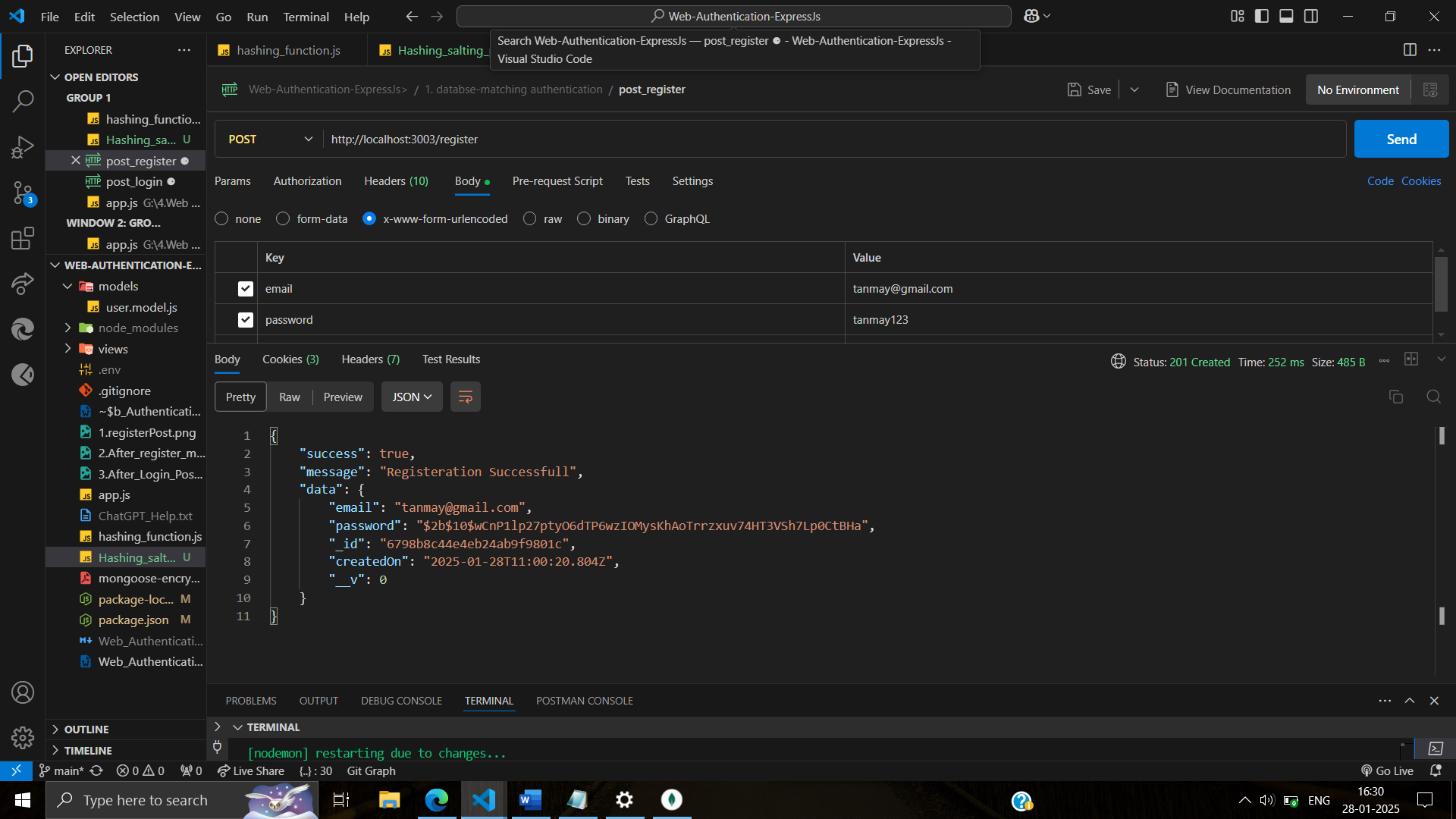
**- we can hash the password with some random number(salting)**

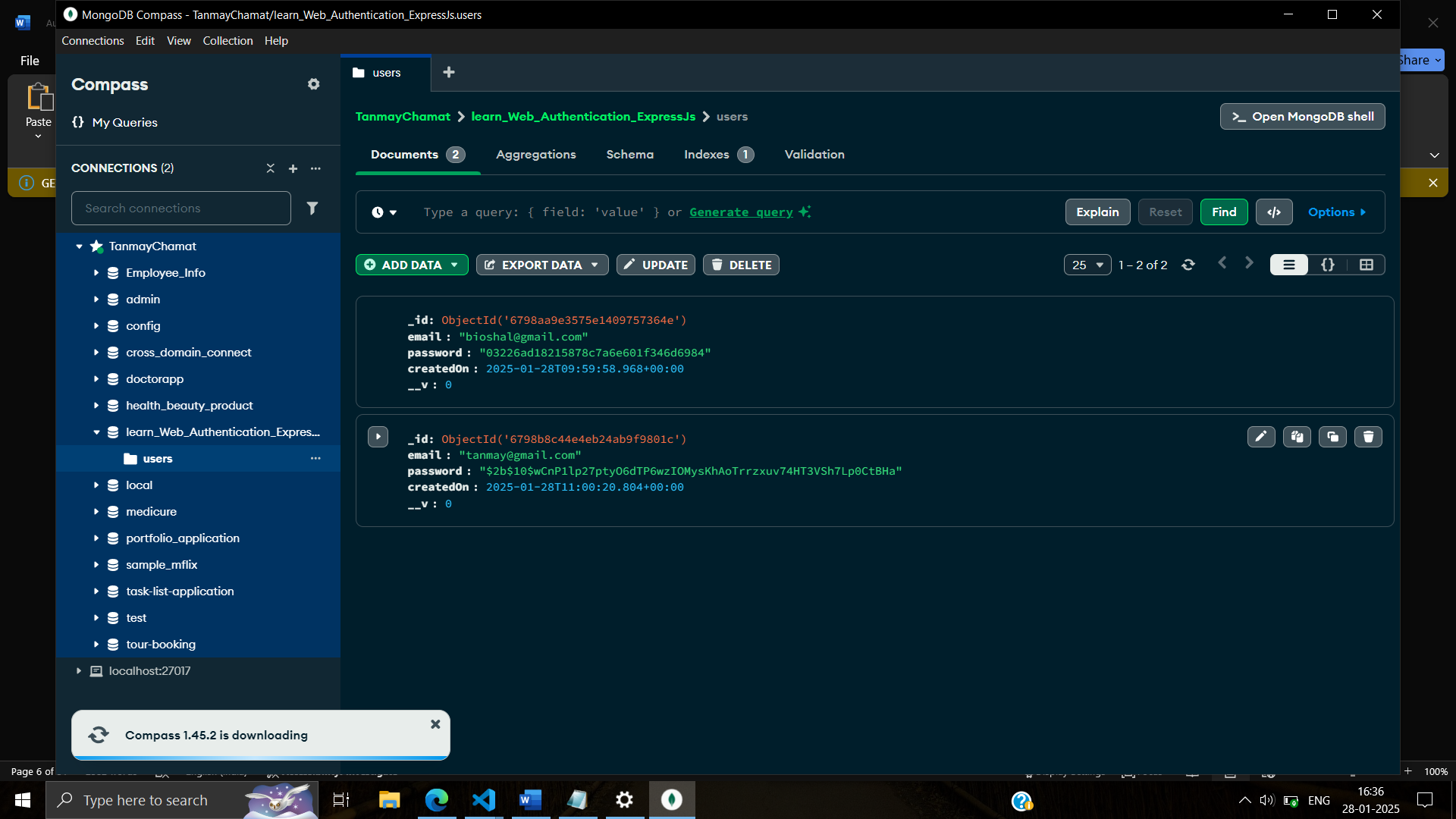
**- install bcrypt npm package `npm install bcrypt`**

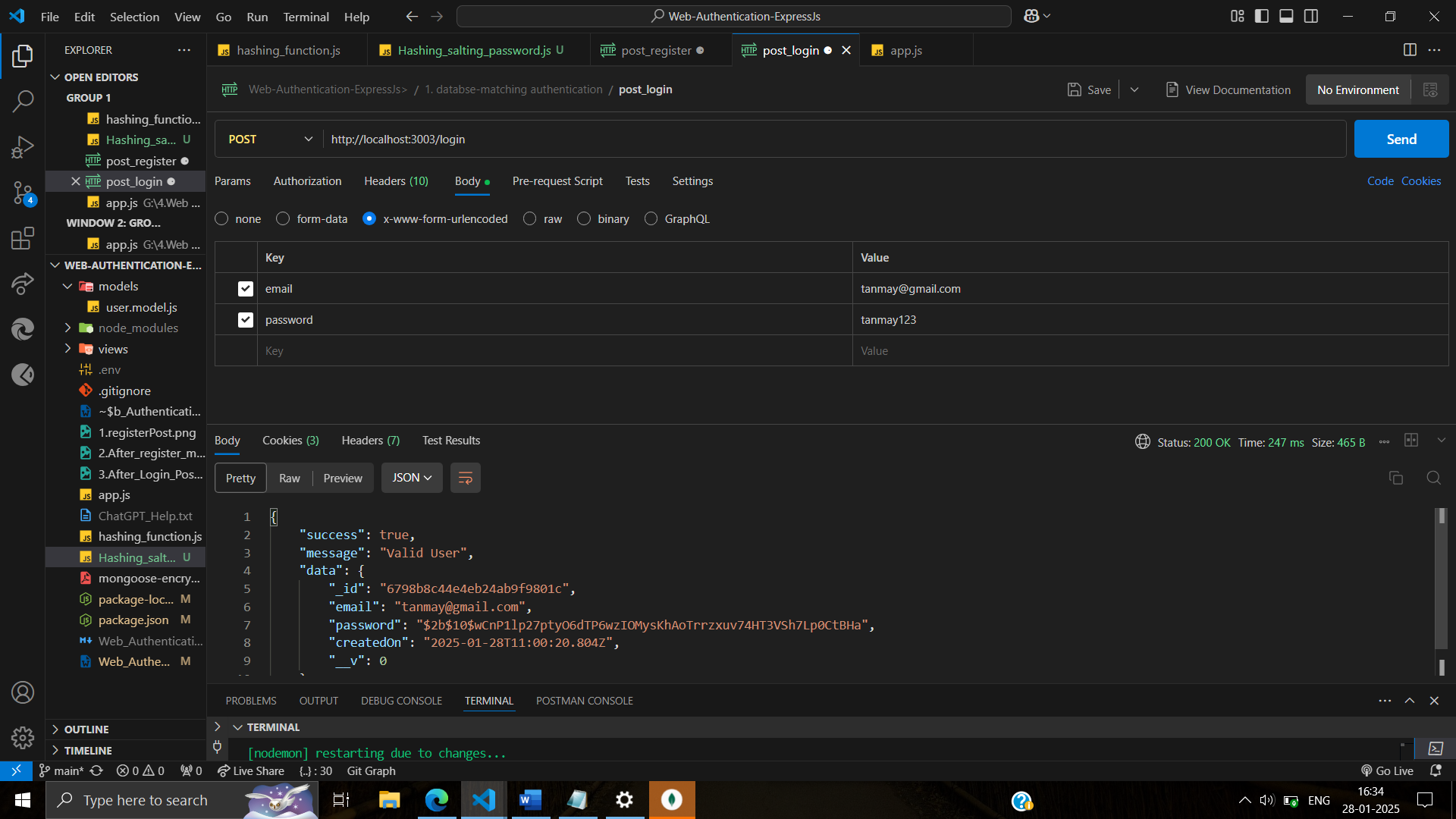


**Explanation of bcrypt usage:**

1. **Password Hashing (Registration):**
   * **During user registration, the bcrypt.hash() function is used to hash the user's password with a defined number of salt rounds (saltRounds).**
   * **The hashed password is stored in the database to ensure sensitive data isn't saved in plain text.**
2. **Password Comparison (Login):**
   * **During login, the bcrypt.compare() function compares the provided plain-text password with the hashed password stored in the database.**
   * **If the comparison is successful, the user is authenticated. If not, an error response is sent.**







**- usage**

**```js**

**const bcrypt = require("bcrypt");**

**const saltRounds = 10;**

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

**try {**

**bcrypt.hash(req.body.password, saltRounds, async function (err, hash) {**

**const newUser = new User({**

**email: req.body.email,**

**password: hash,**

**});**

**await newUser.save();**

**res.status(201).json(newUser);**

**});**

**} catch (error) {**

**res.status(500).json(error.message);**

**}**

**});**

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

**try {**

**const email = req.body.email;**

**const password = req.body.password;**

**const user = await User.findOne({ email: email });**

**if (user) {**

**bcrypt.compare(password, user.password, function (err, result) {**

**if (result === true) {**

**res.status(200).json({ status: "valid user" });**

**}**

**});**

**} else {**

**res.status(404).json({ status: "Not valid user" });**

**}**

**} catch (error) {**

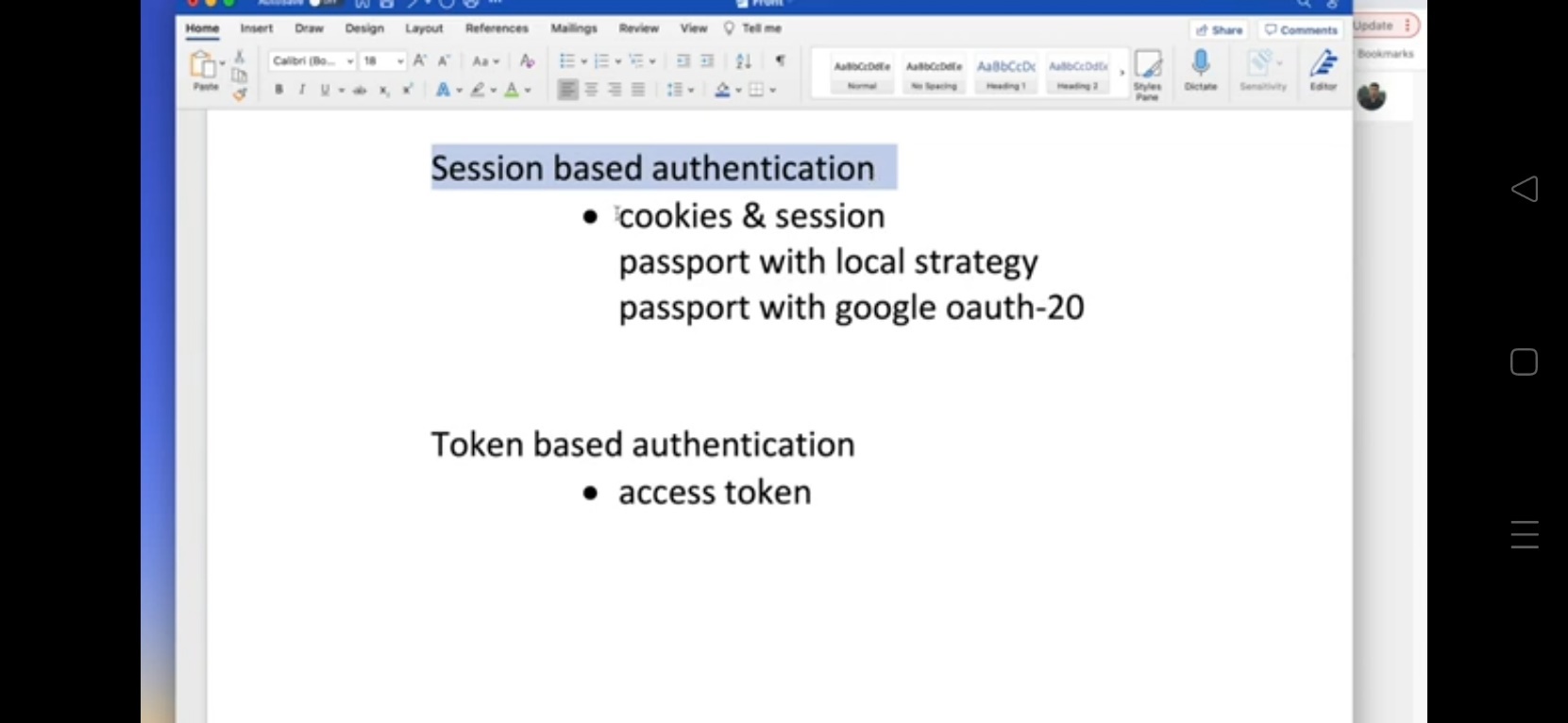
**res.status(500).json(error.message);**

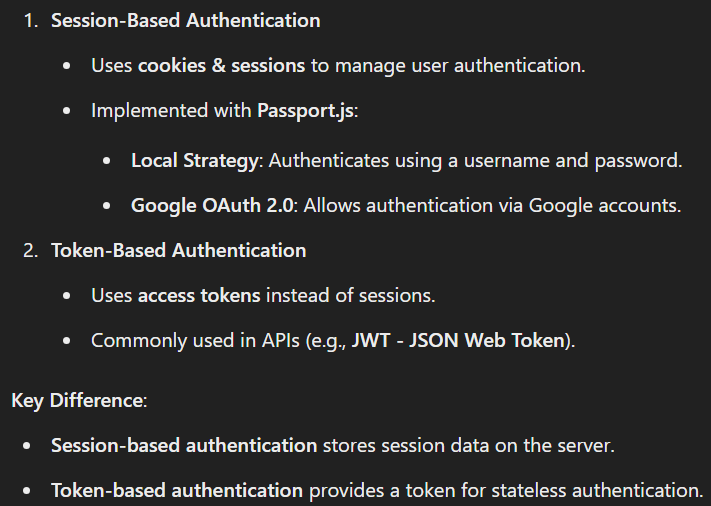
**}**

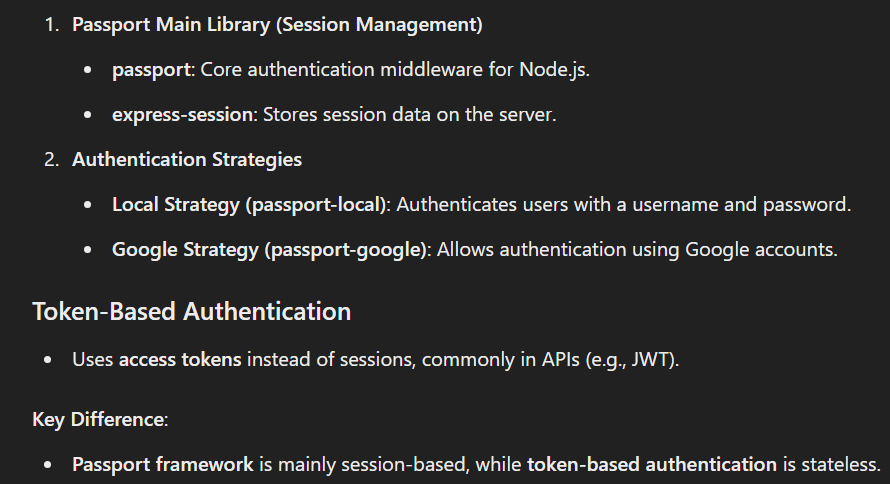
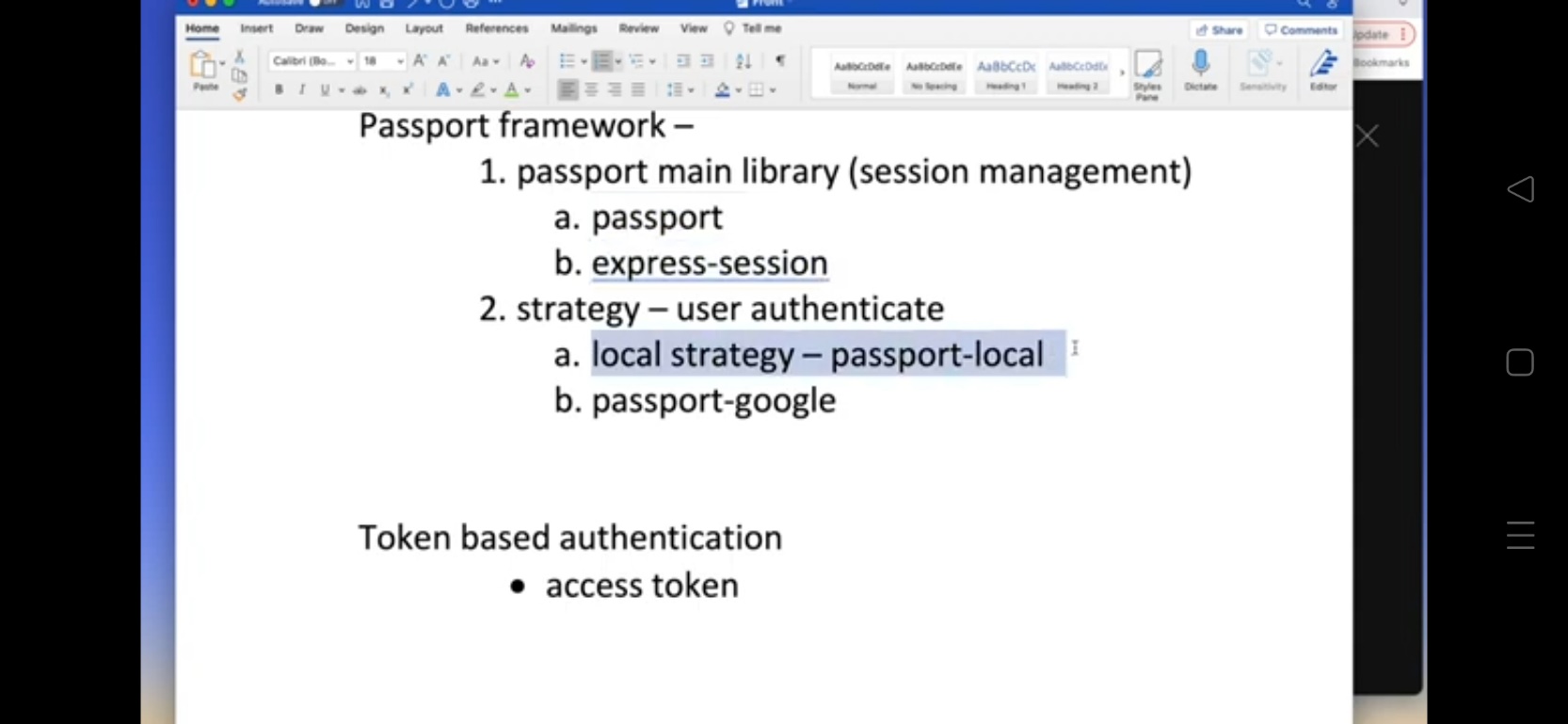
**});**

**```**

**5: Cookies & Session with passport:**

****

****

****

**- passport local strategy**

**- `npm install passport passport-local passport-local-mongoose express-session`**

**- my computer browser -> browse aliexpress (GET Request) -> to aliexpress server -> response the website -> add some items to the cart (post request to the server) -> aliexpress server will response and tell the browser to create a file in my computer for storing my selection -> so when next time we make a get request to the server we send the cookie with the get request -> server will return the cart again**

**- cookie is a text file created by server on a user's device when we visit a website**

**- that stores limited information such as login credentials - username, password; user preferences, cart contents from a web browser session**

**- saving users behaviour**

**- read more about cookies - https://www.trendmicro.com/vinfo/us/security/definition/cookies**

**- types of cookies -> session cookie, presistent cookie, supercookie**

**- login -> save user credentials as cookie for next time authentication -> log out and the session is destroyed**

**- salt and hash is automatically generated by passport-local-mongoose**

**- express session package create the cookie**

**1. passport js framework has 2 separeate libraries**

**- Passport JS Library (main) - maintain session information for user authentication**

**- strategy library - methodology for authenticate an user - passport-local, passport-facebook, passport-oauth2 etc.**

**2. Login process handled by 2 steps: i) session management (Passport.js), ii) authentication (strategy)**

**`npm install passport-local`**

**`npm install passport-facebook`**

**3. for managing session Passport.js library takes help from express-session library**

**`npm install passport express-session`**

**4. source code**

**- bootstrap the project**

**- installing & requiring packages**

**`npm install express nodemon dotenv mongoose ejs cors`**

**- creating server**

**```js**

**//app.js**

**const express = require("express");**

**const cors = require("cors");**

**const ejs = require("ejs");**

**const app = express();**

**app.set("view engine", "ejs");**

**app.use(cors());**

**app.use(express.urlencoded({ extended: true }));**

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

**module.exports = app;**

**//index.js**

**const app = require("./app");**

**const PORT = 4000;**

**app.listen(PORT, () => {**

**console.log(`app is running at http://localhost:${PORT}`);**

**});**

**```**

**- creating routes including try,catch**

**```js**

**// base url**

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

**res.render("index");**

**});**

**// register routes**

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

**res.render("register");**

**});**

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

**try {**

**res.status(201).send("user is registered");**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**// login routes**

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

**res.render("login");**

**});**

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

**try {**

**res.status(201).send("user is logged in");**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**// logout routes**

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

**res.redirect("/");**

**});**

**// profile protected routes**

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

**res.render("profile");**

**});**

**```**

**- creating ejs files**

**- create layout**

**```html**

**<!-- views/layout/header.ejs -->**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

**<meta http-equiv="X-UA-Compatible" content="IE=edge" />**

**<meta**

**name="viewport"**

**content="width=device-width, initial-scale=1.0"**

**/>**

**<title>Document</title>**

**</head>**

**<body>**

**<header>**

**<nav>**

**<a href="/">Home</a>**

**<a href="/register">Register</a>**

**<a href="/login">Login</a>**

**<a href="/profile">Profile</a>**

**<a href="/logout">Logout</a>**

**</nav>**

**</header>**

**</body>**

**</html>**

**<!-- views/layout/footer.ejs -->**

**<footer>**

**<p>copyright by Anisul Islam</p>**

**</footer>**

**</body>**

**</html>**

**```**

**- create pages**

**```html**

**<!-- views/index.ejs -->**

**<%-include("layout/header")%>**

**<main>**

**<h1>Home Page</h1>**

**</main>**

**<%-include("layout/footer")%>**

**<!-- views/register.ejs -->**

**<%-include("layout/header")%>**

**<main>**

**<h1>Register Page</h1>**

**<form action="/register" method="post">**

**<div>**

**<label for="username">username: </label>**

**<input type="text" id="username" name="username" />**

**</div>**

**<br />**

**<div>**

**<label for="password">password: </label>**

**<input type="password" id="password" name="password" />**

**</div>**

**<br />**

**<button type="submit">Register</button>**

**</form>**

**</main>**

**<%-include("layout/footer")%>**

**<!-- views/login.ejs -->**

**<%-include("layout/header")%>**

**<main>**

**<h1>Login Page</h1>**

**<form action="/register" method="post">**

**<div>**

**<label for="username">username: </label>**

**<input type="text" id="username" name="username" />**

**</div>**

**<br />**

**<div>**

**<label for="password">password: </label>**

**<input type="password" id="password" name="password" />**

**</div>**

**<br />**

**<button type="submit">Login</button>**

**</form>**

**</main>**

**<%-include("layout/footer")%>**

**<!-- views/profile.ejs -->**

**<%-include("layout/header")%>**

**<main>**

**<h1>Profile Page</h1>**

**</main>**

**<%-include("layout/footer")%>**

**```**

**- create model and connect to mongodb**

**```js**

**// models/user.model.js**

**const mongoose = require("mongoose");**

**const userSchema = mongoose.Schema({**

**username: {**

**type: String,**

**require: true,**

**unique: true,**

**},**

**password: {**

**type: String,**

**require: true,**

**},**

**});**

**const User = mongoose.model("User", userSchema);**

**module.exports = User;**

**// config/database.js**

**const mongoose = require("mongoose");**

**mongoose**

**.connect("mongodb://localhost:27017/passportDB")**

**.then(() => {**

**console.log("db is connected");**

**})**

**.catch((error) => {**

**console.log(error.message);**

**});**

**// app.js**

**require("./config/database");**

**```**

**- register an user**

**```js**

**//app.js**

**const User = require("./models/user.model");**

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

**try {**

**const user = await User.findOne({ username: req.body.username });**

**if (user) return res.status(400).send("User already exist");**

**const newUser = new User(req.body);**

**await newUser.save();**

**res.status(201).send(newUser);**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**```**

**- encrypt the user password using bcrypt hashing+salting**

**`npm install bcrypt`**

**```js**

**//app.js**

**const bcrypt = require("bcrypt");**

**const saltRounds = 10;**

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

**try {**

**const user = await User.findOne({ username: req.body.username });**

**if (user) return res.status(400).send("User already exist");**

**bcrypt.hash(req.body.password, saltRounds, async (err, hash) => {**

**const newUser = new User({**

**username: req.body.username,**

**password: hash,**

**});**

**await newUser.save();**

**res.redirect("/login");**

**});**

**} catch (error) {**

**res.status(500).send(error.message);**

**}**

**});**

**```**

**- create and add session**

**`npm install passport express-session connect-mongo`**

**```js**

**//Import the main Passport and Express-Session library**

**const passport = require("passport");**

**const session = require("express-session");**

**// for storing session in different collection**

**const MongoStore = require("connect-mongo");**

**// setting middleware**

**app.set("trust proxy", 1); // trust first proxy**

**app.use(**

**session({**

**secret: "keyboard cat",**

**resave: false,**

**saveUninitialized: true,**

**store: MongoStore.create({**

**mongoUrl: "mongodb://localhost:27017/testPassportDB",**

**collectionName: "sessions",**

**}),**

**// cookie: { secure: true },**

**// cookie: { maxAge: 1000 \* 60 \* 60 \* 24 },**

**})**

**);**

**app.use(passport.initialize());**

**// init passport on every route call.**

**app.use(passport.session());**

**// allow passport to use "express-session".**

**```**

**- set passport-local configuration**

**`npm install passport-local`**

**```js**

**// passport.js**

**const User = require("../model/user.model");**

**const passport = require("passport");**

**const bcrypt = require("bcrypt");**

**const LocalStrategy = require("passport-local").Strategy;**

**passport.use(**

**new LocalStrategy(async function (username, password, done) {**

**try {**

**const user = await User.findOne({ username: username });**

**// wrong username**

**if (!user) {**

**return done(null, false, { message: "Incorrect Username" });**

**}**

**// wrong password**

**if (!bcrypt.compare(password, user.password)) {**

**return done(null, false, { message: "Incorrect Password" });**

**}**

**// if user found**

**return done(null, user);**

**} catch (error) {**

**return done(error);**

**}**

**})**

**);**

**// create session id**

**// whenever we login it creares user id inside session**

**passport.serializeUser((user, done) => {**

**done(null, user.id);**

**});**

**// find session info using session id**

**passport.deserializeUser(async (id, done) => {**

**try {**

**const user = await User.findById(id);**

**done(null, user);**

**} catch (error) {**

**done(error, false);**

**}**

**});**

**```**

**- authenticate user using passport-local**

**```js**

**// app.js**

**// login using passport-local strategy**

**const checkLoggedIn = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return res.redirect("/profile");**

**}**

**next();**

**};**

**// login routes**

**app.get("/login", checkLoggedIn, (req, res) => {**

**try {**

**res.render("login");**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**// register routes**

**app.get("/register", checkLoggedIn, (req, res) => {**

**res.render("register");**

**});**

**```**

**- check user is already logged in or not**

**```js**

**const checkAuthenticated = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return next();**

**}**

**res.redirect("/login");**

**};**

**// profile protected routes**

**app.get("/profile", checkAuthenticated, (req, res) => {**

**res.render("profile");**

**});**

**```**

**- logout route setup**

**```js**

**// logout routes**

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

**try {**

**req.logout((err) => {**

**if (err) {**

**return next(err);**

**}**

**res.redirect("/");**

**});**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**```**

**- finally the entire app.js**

**```js**

**const express = require("express");**

**const cors = require("cors");**

**const ejs = require("ejs");**

**const bcrypt = require("bcrypt");**

**const saltRounds = 10;**

**const passport = require("passport");**

**const session = require("express-session");**

**// for storing session in different collection**

**const MongoStore = require("connect-mongo");**

**require("./config/database");**

**require("./config/passport");**

**const User = require("./model/user.model");**

**const app = express();**

**app.set("view engine", "ejs");**

**app.use(cors());**

**app.use(express.urlencoded({ extended: true }));**

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

**// setting middleware**

**app.set("trust proxy", 1); // trust first proxy**

**app.use(**

**session({**

**secret: "keyboard cat",**

**resave: false,**

**saveUninitialized: true,**

**store: MongoStore.create({**

**mongoUrl: "mongodb://localhost:27017/passportDB",**

**collectionName: "sessions",**

**}),**

**// cookie: { maxAge: 1000 \* 60 \* 60 \* 24 },**

**})**

**);**

**app.use(passport.initialize());**

**// init passport on every route call.**

**app.use(passport.session());**

**// allow passport to use "express-session".**

**const checkLoggedIn = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return res.redirect("/profile");**

**}**

**next();**

**};**

**// base url**

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

**res.render("index");**

**});**

**// register routes**

**app.get("/register", checkLoggedIn, (req, res) => {**

**res.render("register");**

**});**

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

**try {**

**const user = await User.findOne({ username: req.body.username });**

**if (user) return res.status(400).send("User already exist");**

**bcrypt.hash(req.body.password, saltRounds, async (err, hash) => {**

**const newUser = new User({**

**username: req.body.username,**

**password: hash,**

**});**

**await newUser.save();**

**res.redirect("/login");**

**});**

**} catch (error) {**

**res.status(500).send(error.message);**

**}**

**});**

**// login routes**

**app.get("/login", checkLoggedIn, (req, res) => {**

**res.render("login");**

**});**

**app.post(**

**"/login",**

**passport.authenticate("local", { successRedirect: "/profile" })**

**);**

**// logout routes**

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

**try {**

**req.logout((err) => {**

**if (err) {**

**return next(err);**

**}**

**res.redirect("/");**

**});**

**} catch (error) {**

**req.status(500).send(error.message);**

**}**

**});**

**const checkAuthenticated = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return next();**

**}**

**res.redirect("/login");**

**};**

**// profile protected routes**

**app.get("/profile", checkAuthenticated, (req, res) => {**

**res.render("profile");**

**});**

**module.exports = app;**

**```**

**## Level 6: Google OAuth with passport session based**

**- setup database name in db and also for session**

**- change in schema**

**```js**

**const mongoose = require("mongoose");**

**const userSchema = mongoose.Schema({**

**username: {**

**type: String,**

**require: true,**

**unique: true,**

**},**

**googleId: {**

**type: String,**

**require: true,**

**},**

**});**

**const User = mongoose.model("User", userSchema);**

**module.exports = User;**

**```**

**- inside login ejs add**

**` <a href="/auth/google">Login with Google</a>`**

**- create dynamic user name in profile page `Welcome <%=username%>`**

**- configure strategy**

**- we need client id, client secret**

**```js**

**// passport.js**

**require("dotenv").config();**

**const User = require("../models/user.model");**

**const passport = require("passport");**

**const GoogleStrategy = require("passport-google-oauth20").Strategy;**

**passport.use(**

**new GoogleStrategy(**

**{**

**clientID: process.env.GOOGLE\_CLIENT\_ID,**

**clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,**

**callbackURL: "http://localhost:5000/auth/google/callback",**

**},**

**function (accessToken, refreshToken, profile, cb) {**

**User.findOne({ googleId: profile.id }, (err, user) => {**

**if (err) return cb(err, null);**

**// not a user; so create a new user with new google id**

**if (!user) {**

**let newUser = new User({**

**googleId: profile.id,**

**username: profile.displayName,**

**});**

**newUser.save();**

**return cb(null, newUser);**

**} else {**

**// if we find an user just return return user**

**return cb(null, user);**

**}**

**});**

**}**

**)**

**);**

**// create session id**

**// whenever we login it creares user id inside session**

**passport.serializeUser((user, done) => {**

**done(null, user.id);**

**});**

**// find session info using session id**

**passport.deserializeUser(async (id, done) => {**

**try {**

**const user = await User.findById(id);**

**done(null, user);**

**} catch (error) {**

**done(error, false);**

**}**

**});**

**// app.js**

**const express = require("express");**

**const cors = require("cors");**

**const ejs = require("ejs");**

**const app = express();**

**require("./config/database");**

**require("dotenv").config();**

**require("./config/passport");**

**const User = require("./models/user.model");**

**const passport = require("passport");**

**const session = require("express-session");**

**const MongoStore = require("connect-mongo");**

**app.set("view engine", "ejs");**

**app.use(cors());**

**app.use(express.urlencoded({ extended: true }));**

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

**app.set("trust proxy", 1); // trust first proxy**

**app.use(**

**session({**

**secret: "keyboard cat",**

**resave: false,**

**saveUninitialized: true,**

**store: MongoStore.create({**

**mongoUrl: process.env.MONGO\_URL,**

**collectionName: "sessions",**

**}),**

**// cookie: { secure: true },**

**})**

**);**

**app.use(passport.initialize());**

**app.use(passport.session());**

**// base url**

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

**res.render("index");**

**});**

**const checkLoggedIn = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return res.redirect("/profile");**

**}**

**next();**

**};**

**// login : get**

**app.get("/login", checkLoggedIn, (req, res) => {**

**res.render("login");**

**});**

**app.get(**

**"/auth/google",**

**passport.authenticate("google", { scope: ["profile"] })**

**);**

**app.get(**

**"/auth/google/callback",**

**passport.authenticate("google", {**

**failureRedirect: "/login",**

**successRedirect: "/profile",**

**}),**

**function (req, res) {**

**// Successful authentication, redirect home.**

**res.redirect("/");**

**}**

**);**

**const checkAuthenticated = (req, res, next) => {**

**if (req.isAuthenticated()) {**

**return next();**

**}**

**res.redirect("/login");**

**};**

**// profile protected route**

**app.get("/profile", checkAuthenticated, (req, res) => {**

**res.render("profile", { username: req.user.username });**

**});**

**// logout route**

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

**try {**

**req.logout((err) => {**

**if (err) {**

**return next(err);**

**}**

**res.redirect("/");**

**});**

**} catch (error) {**

**res.status(500).send(error.message);**

**}**

**});**

**module.exports = app;**

**```**

**## Level 7: passport-jwt (token based)**

**- how token based works**

**- user register using username, password to the server -> server creates a token for the user -> so next time when user make any request server give access by validating the given token**

**- folder and file structure**

**- server**

**- models**

**- user.model.js**

**- config**

**- app.js**

**- index.js**

**- .env**

**- .gitignore**

**- initialize npm and install package**

**`npm init -y && npm install express nodemon cors dotenv bcrypt mongoose`**

**- test**