Documentation Report — Day 22: Forms, Database, Authentication

server.js — Main Application Entry Point

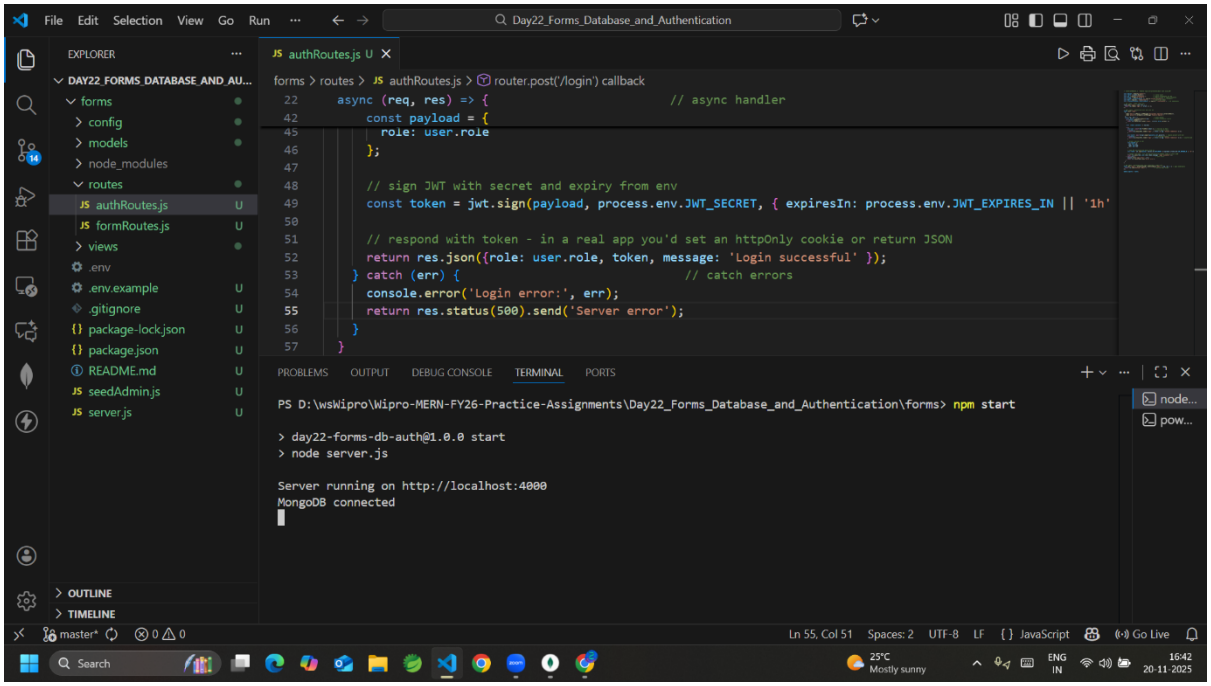
This is the central file that initializes the Express server, configures middleware, loads routes, connects to MongoDB, and starts the application.
It also sets EJS as the view engine and enables body parsing, sessions, and static files.

Key Code Snippet:

```
// server.js : main server file, start point

const express = require('express');
const mongoose = require('mongoose');          // import mongoose for DB
const dotenv = require('dotenv');              // import dotenv to load .env
const bodyParser = require('body-parser');
const path = require('path');

dotenv.config();                               // load environment variables
from .env

const app = express();                          // create express app

// view engine setup
app.set('view engine', 'ejs');                  // set EJS as the template
engine
app.set('views', path.join(__dirname, 'views'));// set views directory path

// middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

// connect to MongoDB
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB error:', err));

// routes
app.get("/", (req, res) => {
  res.render("home");
});


app.use('/', require('./routes/formRoutes'));    // mount form routes at root
```

```
app.use('/', require('./routes/authRoutes'));    // mount auth routes at root

// seed admin script route (optional, not for production)
// this route is not created here; use seedAdmin.js to create admin user


const PORT = process.env.PORT || 4000;
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

Output:



models/User.js — User Schema (Mongoose)

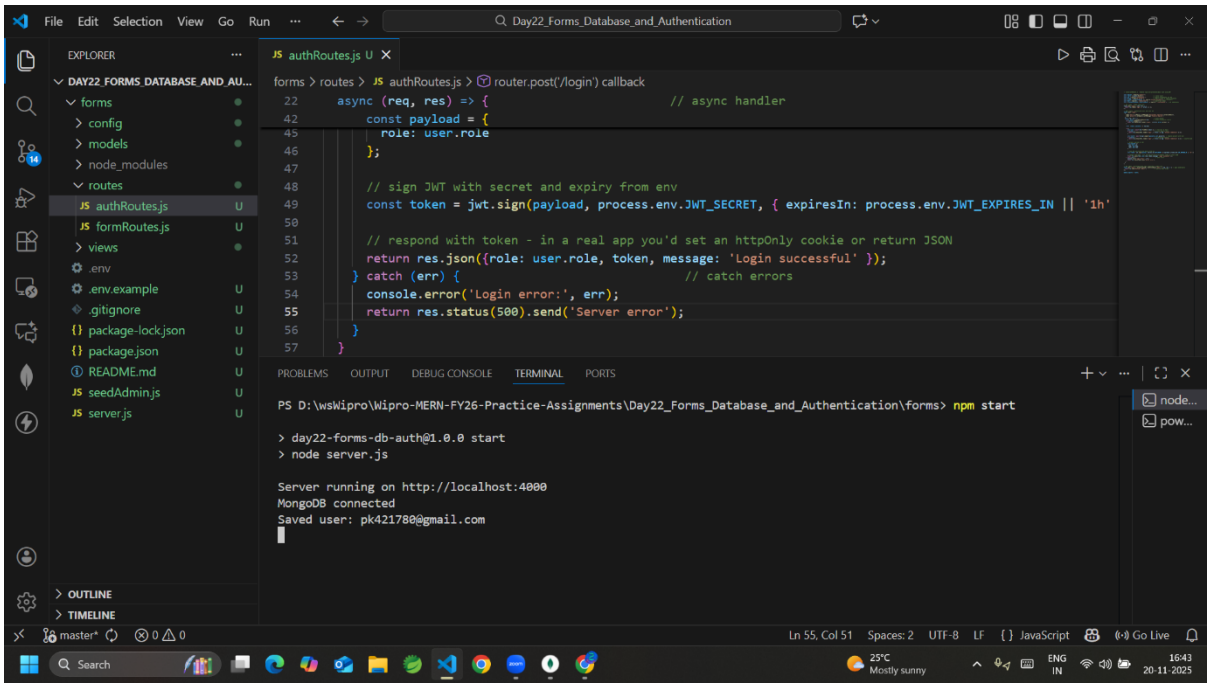
Defines the MongoDB user model including fields like name, email, hashed password, and role.
Mongoose handles schema validation and communicates with MongoDB.

Key Code Snippet:

```
// models/User.js : User Mongoose model with schema and comments

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({        // define a new schema
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['user', 'admin'], default: 'user' }
}, { timestamps: true });
```

```
module.exports = mongoose.model('User', userSchema);
```

Output:



routes/formRoutes.js — Form Handling + ValidationKey Code Snippet:

Handles the registration form route, validates input using express-validator, hashes passwords, and saves new users to MongoDB.
Renders EJS pages for registration.

Key Code Snippet:

```
// routes/formRoutes.js : Handles registration form and submission

const express = require('express');
const router = express.Router();                    // create router instance
const { body, validationResult } = require('express-validator'); // import
validators
const bcrypt = require('bcrypt');                   // import bcrypt for hashing
const User = require('../models/User');             // import User model

// GET /register - render registration form
router.get('/register', (req, res) => {
  return res.render('register', { errors: [] });
});

// POST /register - handle registration form submission
router.post('/register',
  [
    body('name').trim().notEmpty().withMessage('Name is required'),
    body('email').isEmail().withMessage('Valid email
required').normalizeEmail(),
```

```
    body('password').isLength({ min: 5 }).withMessage('Password must be at
least 5 characters')
  ],
  async (req, res) => {                          // async handler for DB
operations
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(422).render('register', { errors: errors.array() });
    }

    const { name, email, password } = req.body;

    try {
      const existing = await User.findOne({ email }); // check if user exists
      if (existing) {
        return res.status(409).render('register', { errors: [{ msg: 'Email
already in use' }] });
      }

      const saltRounds = 10;                      // bcrypt salt rounds
      const hashed = await bcrypt.hash(password, saltRounds); // hash the
password

      const newUser = new User({                  // create user document
        name,
        email,
        password: hashed,
        role: 'user'
      });

      await newUser.save();                       // save to MongoDB
      console.log('Saved user:', newUser.email);

      // Respond with success message as requested
      return res.send(`Registration successful for ${name}`);
    } catch (err) {                               // catch DB / other errors
      console.error('Registration error:', err);
      return res.status(500).send('Server error');
    }
  }
);

module.exports = router;
```

Output:

routes/authRoutes.js — Login, JWT, Protected Routes

Handles login and JWT creation. Valid credentials generate a token containing user id + role. Also defines the `/admin` protected route accessible only to admin users.

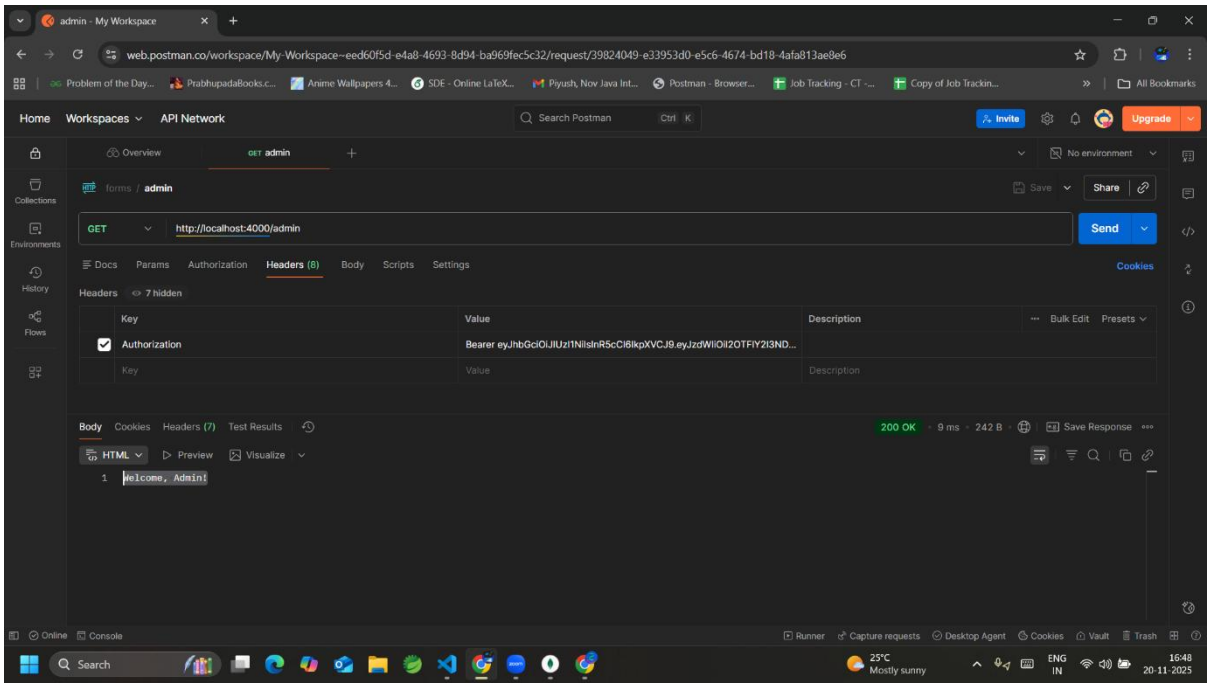
Key Code Snippet:

```
// routes/authRoutes.js : Handles login and protected admin route using JWT

const express = require('express');
const router = express.Router();                // create router
const jwt = require('jsonwebtoken');            // import jsonwebtoken for JWT
const bcrypt = require('bcrypt');               // import bcrypt to compare
passwords
const { body, validationResult } = require('express-validator'); // validators
const User = require('../models/User');         // import User model
const { authenticateJWT, authorizeRole } = require('../config/auth'); // JWT
middlewares

// GET /login - render login form
router.get('/login', (req, res) => {
  return res.render('login', { errors: [] });
});

// POST /login - authenticate user and issue JWT
router.post('/login',
  [
    body('email').isEmail().withMessage('Valid email
required').normalizeEmail(),
    body('password').notEmpty().withMessage('Password required')
  ],
```

```
  async (req, res) => {                          // async handler
    const errors = validationResult(req);       // collect validation results
    if (!errors.isEmpty()) {
      return res.status(422).render('login', { errors: errors.array() });
    }

    const { email, password } = req.body;

    try {
      const user = await User.findOne({ email }); // find user by email
      if (!user) {                               // if user doesn't exist
        return res.status(401).render('login', { errors: [{ msg: 'Invalid
credentials' }] });
      }

      const match = await bcrypt.compare(password, user.password); // compare
password with hash
      if (!match) {                              // if password mismatch
        return res.status(401).render('login', { errors: [{ msg: 'Invalid
credentials' }] }); // unauthorized
      }

      // prepare payload for JWT
      const payload = {
        sub: user._id,
        name: user.name,
        role: user.role
      };

      // sign JWT with secret and expiry from env
      const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn:
process.env.JWT_EXPIRES_IN || '1h' });

      // respond with token - in a real app you'd set an httpOnly cookie or
return JSON
      return res.json({role: user.role, token, message: 'Login successful' });
    } catch (err) {                              // catch errors
      console.error('Login error:', err);
      return res.status(500).send('Server error');
    }
  }
);

// GET /admin - protected route, only accessible to admin role
router.get('/admin', authenticateJWT, authorizeRole('admin'), (req, res) => {
// apply middlewares
  return res.send('Welcome, Admin!');           // required admin output
});
```

```
module.exports = router;
```

Output:



config/auth.js — JWT Verification + RBAC

Contains JWT verification middleware that checks the bearer token and attaches user details to `req.user`.
Also includes role-based authorization middleware to restrict admin routes.

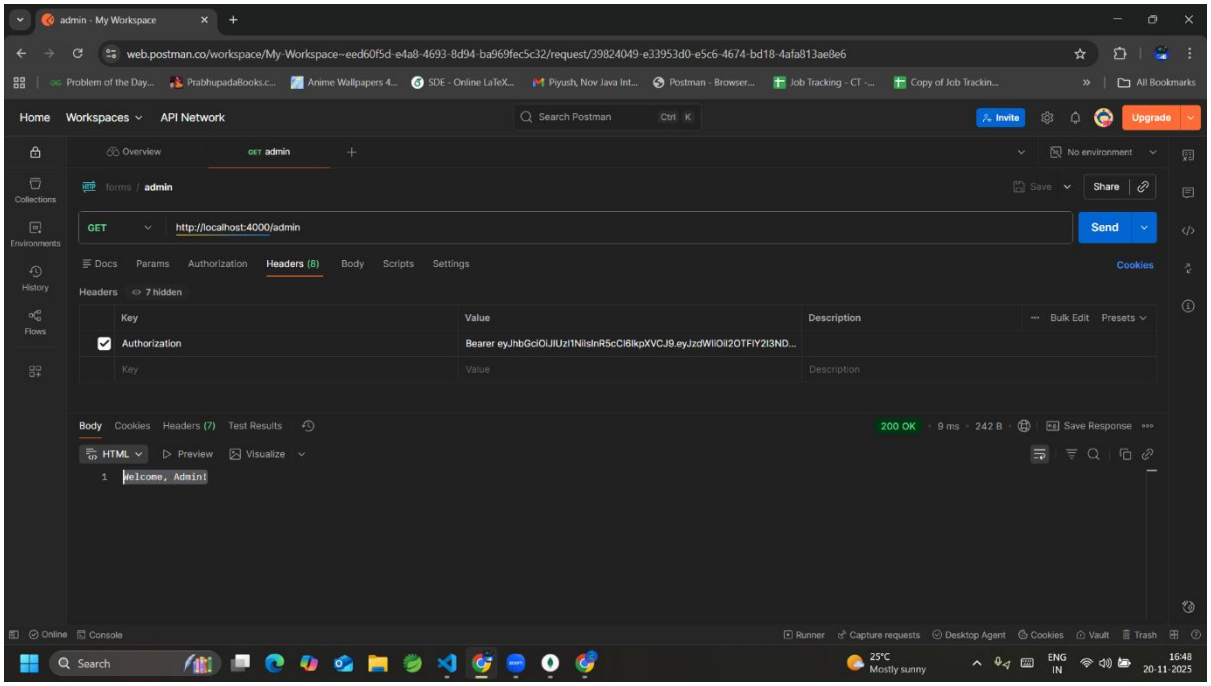
Key Code Snippet:

```
// config/auth.js : JWT authentication and RBAC middleware

const jwt = require('jsonwebtoken');
const User = require('../models/User');

// authenticateJWT middleware verifies Bearer token and attaches user info to
req.user
function authenticateJWT(req, res, next) {      // middleware signature
  const authHeader = req.headers['authorization'] ||
req.headers['Authorization']; // read auth header

  if (!authHeader || !authHeader.startsWith('Bearer ')) { // if no bearer
token
    return res.status(401).send('Access Denied');
  }

  const token = authHeader.split(' ')[1];        // extract token part

  try {
```

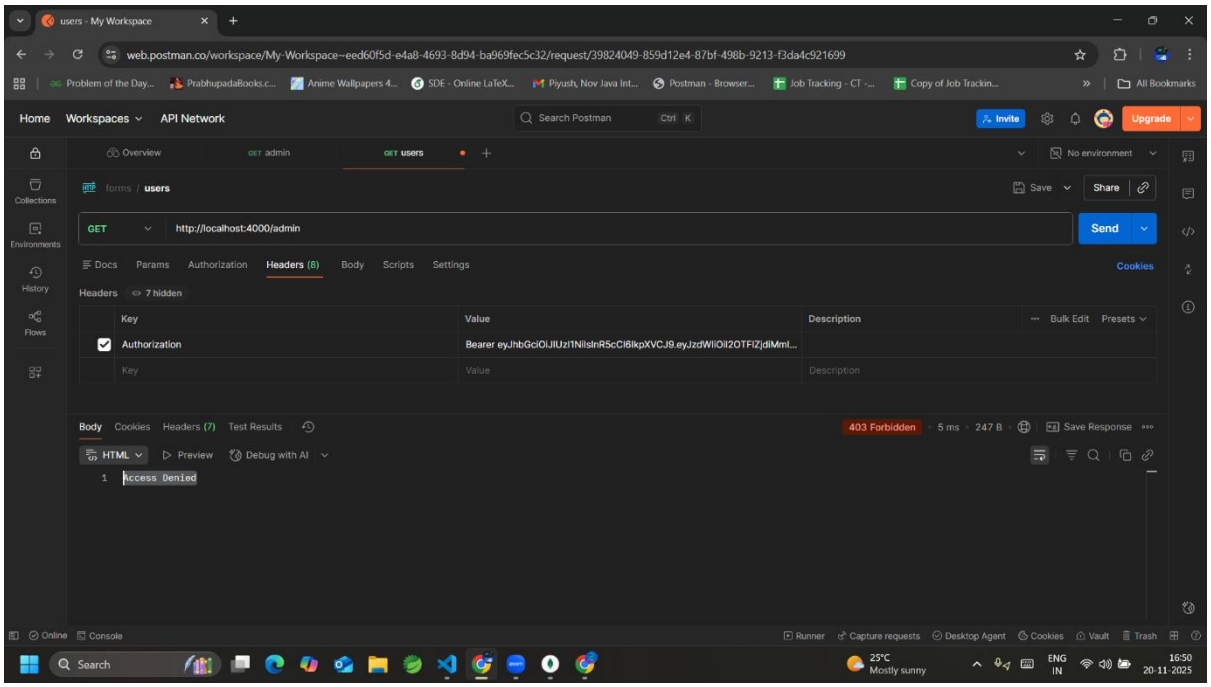
```
    const decoded = jwt.verify(token, process.env.JWT_SECRET); // verify token
signature
    // attach decoded payload to request for downstream handlers
    req.user = {
      id: decoded.sub,
      name: decoded.name,
      role: decoded.role
    };
    return next();
  } catch (err) {
    console.error('JWT error:', err);
    return res.status(401).send('Access Denied');
  }
}

// authorizeRole middleware ensures the user has required role(s)
function authorizeRole(requiredRole) {
  return function (req, res, next) {
    if (!req.user) {                            // if user not attached
      return res.status(401).send('Access Denied');
    }
    if (req.user.role !== requiredRole) {       // if roles don't match
      return res.status(403).send('Access Denied');
    }
    return next();
  };
}

module.exports = { authenticateJWT, authorizeRole };
```

Output:

[Authorized]



[Unauthorized]



views/register.ejs & login.ejs — Bootstrap Styled

EJS templates used to render frontend pages for registration and login.
Bootstrap is used for clean UI, responsive layout, and modern styling.

Key Code Snippet:

[login.ejs]

```
<!-- views/login.ejs - login form with Bootstrap styling -->
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset="utf-8" />
  <title>Login</title>

  <!-- Bootstrap CSS CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" />
</head>

<body class="bg-light">

  <div class="container d-flex justify-content-center align-items-center"
style="min-height: 100vh;">
    <div class="card shadow-lg p-4" style="width: 400px;">

      <h3 class="text-center mb-4">Login</h3>

      <!-- Display Validation Errors -->
      <% if (errors && errors.length) { %>
        <div class="alert alert-danger">
          <ul class="mb-0">
            <% errors.forEach(function(err){ %>
              <li>
                <%= err.msg %>
              </li>
              <% }) %>
          </ul>
        </div>
        <% } %>

          <!-- Login Form -->
          <form action="/login" method="POST">

            <div class="mb-3">
              <label class="form-label">Email</label>
              <input type="email" name="email" class="form-control"
placeholder="example@gmail.com" required />
            </div>

            <div class="mb-3">
              <label class="form-label">Password</label>
              <input type="password" name="password" class="form-control"
placeholder="Enter password" required />
            </div>

            <button type="submit" class="btn btn-primary w-100">
              Login
```

```
            </button>
          </form>

    </div>
  </div>

  <!-- Bootstrap JS -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min
.js"></script>
</body>

</html>
```

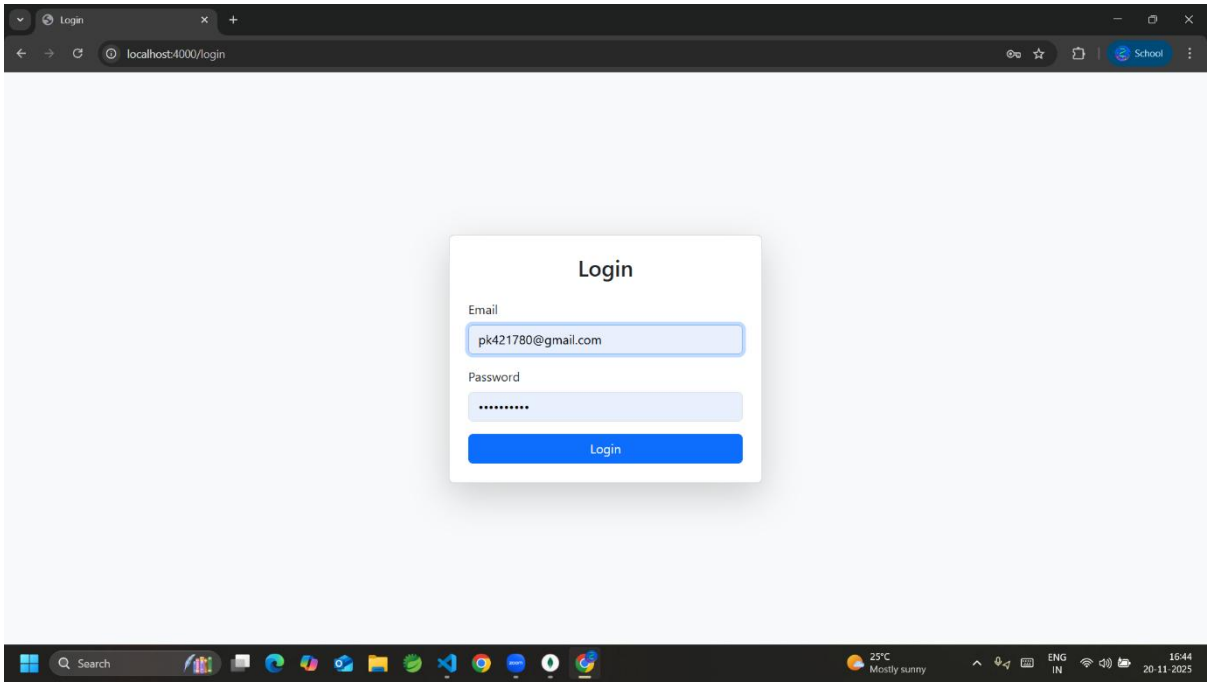
[register.ejs]

```
<!-- views/register.ejs - registration form with Bootstrap -->
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Register</title>

  <!-- Bootstrap CSS CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" />
</head>

<body class="bg-light">

  <div class="container d-flex justify-content-center align-items-center"
style="min-height: 100vh;">
    <div class="card shadow-lg p-4" style="width: 450px;">

      <h3 class="text-center mb-4">Create an Account</h3>

      <!-- Show validation errors -->
      <% if (errors && errors.length) { %>
        <div class="alert alert-danger">
          <ul class="mb-0">
            <% errors.forEach(function(err){ %>
              <li>
                <%= err.msg %>
              </li>
              <% }) %>
          </ul>
        </div>
        <% } %>
```

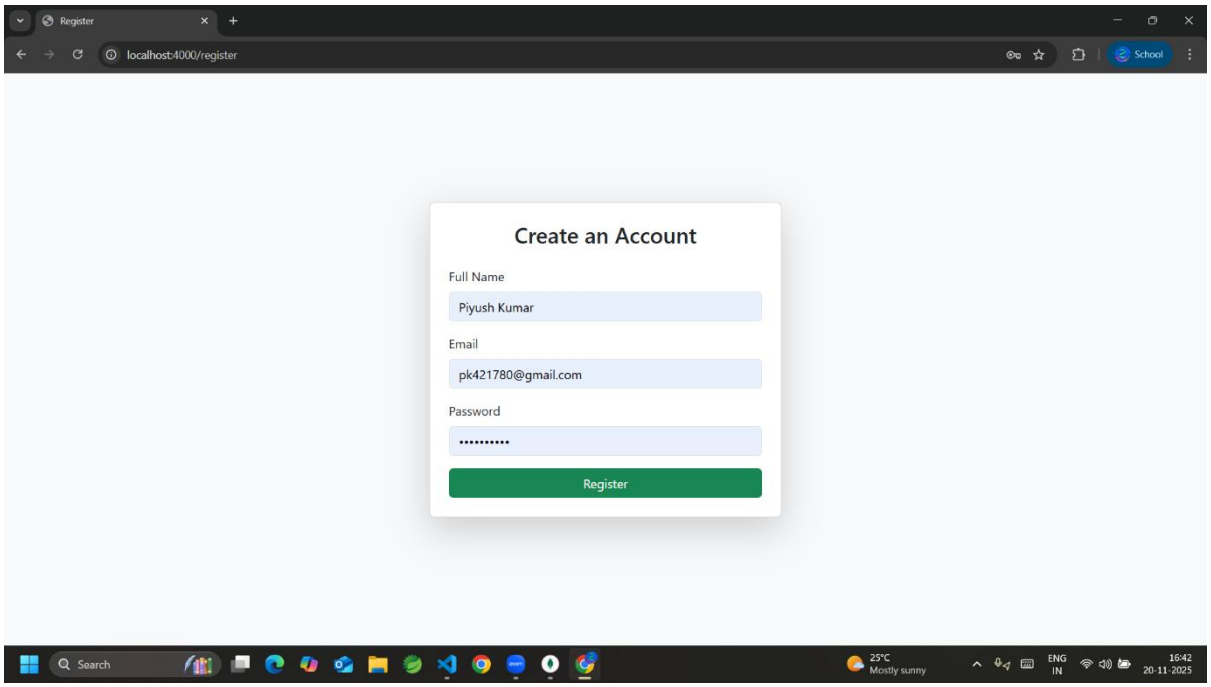
```
          <!-- Registration Form -->
          <form action="/register" method="POST">

            <div class="mb-3">
              <label class="form-label">Full Name</label>
              <input type="text" name="name" class="form-control"
placeholder="John Doe" required />
            </div>

            <div class="mb-3">
              <label class="form-label">Email</label>
              <input type="email" name="email" class="form-control"
placeholder="example@gmail.com" required />
            </div>

            <div class="mb-3">
              <label class="form-label">Password</label>
              <input type="password" name="password" class="form-control"
placeholder="Enter a strong password"
                 required />
            </div>

            <button type="submit" class="btn btn-success w-
100">Register</button>
          </form>

    </div>
  </div>

  <!-- Bootstrap JS -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min
.js"></script>
</body>

</html>
```

Output:

[Login Form]



[Registration Form]



seedAdmin.js — Create Default Admin User

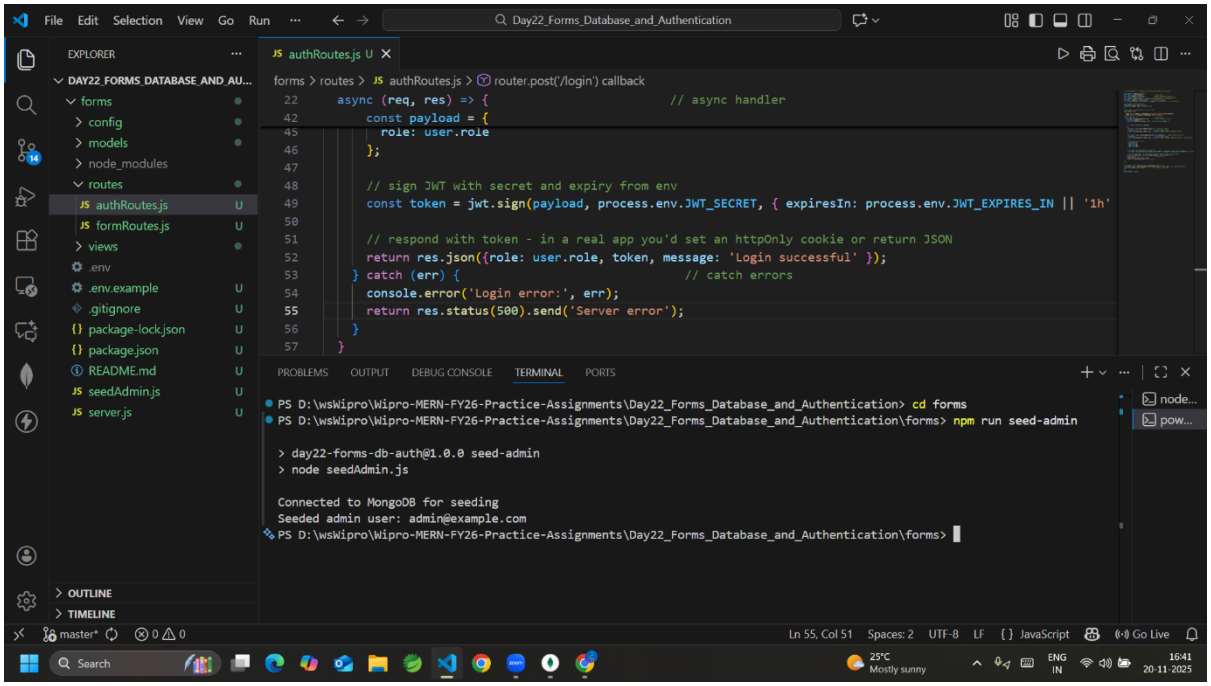
A standalone Node script that inserts an admin user (admin@example.com) into the database with a hashed password.
Useful if no admin user exists for login.

Key Code Snippet:

```
// seedAdmin.js - helper script to create an admin user from command line
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const bcrypt = require('bcrypt');
```

```
const User = require('./models/User');

dotenv.config();

async function seed() {                          // async seeding function
  try {
    await mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true,
useUnifiedTopology: true }); // connect
    console.log('Connected to MongoDB for seeding');

    const email = 'admin@example.com';
    const existing = await User.findOne({ email }); // check existing
    if (existing) {
      existing.role = 'admin';
      await existing.save();
      console.log('Updated existing user to admin:', email);
      process.exit(0);
    }

    const hashed = await bcrypt.hash('adminpass', 10); // hash default
password
    const admin = new User({                        // create admin user
      name: 'Admin User',
      email,
      password: hashed,
      role: 'admin'
    });

    await admin.save();                             // save to DB
    console.log('Seeded admin user:', email);
    process.exit(0);
  } catch (err) {
    console.error('Seeding error:', err);
    process.exit(1);
  }
}

seed();
```

Output:


File Edit Selection View Go Run
Day22_Forms_Database_and_Authentication
EXPLORER
DAY22_FORMS_DATABASE_AND_AU...
forms
config
models
node_modules
routes
authRoutes.js
formRoutes.js
views
.env
.env.example
.gitignore
package-lock.json
package.json
README.md
seedAdmin.js
server.js
authRoutes.js U
forms > routes > authRoutes.js > router.post('/login') callback
22 async (req, res) => { // async handler
42 const payload = {
45 role: user.role
46 };
47
48 // sign JWT with secret and expiry from env
49 const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: process.env.JWT_EXPIRES_IN || '1h'
50
51 // respond with token - in a real app you'd set an httpOnly cookie or return JSON
52 return res.json({role: user.role, token, message: 'Login successful' });
53 } catch (err) { // catch errors
54 console.error('Login error:', err);
55 return res.status(500).send('Server error');
56 }
57 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\wsWipro\Wipro-MERN-FY26-Practice-Assignments\Day22_Forms_Database_and_Authentication> cd forms
PS D:\wsWipro\Wipro-MERN-FY26-Practice-Assignments\Day22_Forms_Database_and_Authentication\forms> npm run seed-admin
> day22-forms-db-auth@1.0.0 seed-admin
> node seedAdmin.js
Connected to MongoDB for seeding
Seeded admin user: admin@example.com
PS D:\wsWipro\Wipro-MERN-FY26-Practice-Assignments\Day22_Forms_Database_and_Authentication\forms>
node...
pow...
OUTLINE
TIMELINE
master*
Ln 55, Col 51 Spaces: 2 UTF-8 LF { } JavaScript Go Live
Search
25°C
Mostly sunny
ENG
IN
16:41
20-11-2025