Session-Based Authentication with Role Authorization & CSRF Protection

Index

1. Project Overview

This application demonstrates secure session-based authentication, role-based authorization, and CSRF protection in a Node.js + Express backend.

2. Learning Outcomes

- Understand session-based authentication
- Implement role-based access control
- Apply CSRF protection for state-changing requests

3. Scope

In scope: sessions, roles, CSRF.
Out of scope: JWT, OAuth.

4. Tech Stack & Versions

- Node.js
- Express.js
- express-session
- csurf / csrf middleware

5. Folder Structure

```
session-role-app-csrf/
├── app.js
├── routes/
│   ├── auth.js
│   └── protected.js
├── middleware/
│   ├── auth.js
│   └── csrf.js
├── package.json
└── README.md
```

6. High-Level Architecture

```
Browser → Session Cookie → Express → Session Store → Role + CSRF Middleware
```

7. Authentication & Authorization Flow

```
1. Login
2. Session created
```

```
3. Cookie issued
4. Role validated
5. Protected resource accessed
```

8. CSRF Protection Flow

```
1. Token generated
2. Token sent to client
3. Token returned in request
4. Token validated
```

9. API Contract

```
POST /login
POST /logout
GET  /admin
POST /update-profile
```

10. Configuration & Environment Variables

```
SESSION_SECRET
CSRF_SECRET
```

11. How to Run

```
npm install
node app.js
```

12. Common Mistakes

- Disabling CSRF
- Storing sensitive data in session
- Missing role checks

13. Debugging Techniques

- Inspect cookies
- Log session object
- Verify CSRF token header

Appendix A: Source Code

.env

```
PORT=5000
MONGO_URI=mongodb://127.0.0.1:27017/session_role_db
SESSION_SECRET=my_session_secret
```

package.json

```
{
  "name": "session-role-app",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "connect-mongo": "^5.0.0",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-session": "^1.17.3",
    "mongoose": "^8.3.4",
    "csurf": "^1.11.0"
  }
}
```

readme.txt

```
How to run
----------
npm install
npm start

How to test correctly
---------------------
1. Get CSRF token
GET http://localhost:5000/api/csrf-token

2. Login (session-based + CSRF)
POST http://localhost:5000/api/auth/login

Headers:
X-CSRF-Token: <csrfToken>

Body:
{
  "email": "admin@test.com",
  "password": "123456"
}


➡ Session cookie is set automatically
```

```
3. Access protected routes
GET /api/auth/profile
GET /api/auth/admin

(No CSRF header needed for GET, Session cookie is used)

4. Logout (CSRF required)
POST /api/auth/logout

Headers:
X-CSRF-Token: <csrfToken>

Architecture rule to remember
Session Cookie → Authentication
CSRF Token     → Request Integrity
Role Middleware→ Authorization

These are three different security layers.
```

src/middleware/roleAuth.js

```
module.exports = (requiredRole) => {
  return (req, res, next) => {
    if (req.session.user.role !== requiredRole) {
      return res.status(403).json({ message: "Forbidden" });
    }
    next();
  };
};
```

src/middleware/sessionAuth.js

```
module.exports = (req, res, next) => {
  if (!req.session.user) {
    return res.status(401).json({ message: "Not authenticated" });
  }
  next();
};
```

src/models/User.js

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

const userSchema = new mongoose.Schema({
  email: { type: String, unique: true },
  password: String,
  role: {
    type: String,
    enum: ["user", "admin"],
    default: "user"
  }
});
```

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

src/routes/authRoutes.js

```
const express = require("express");
const bcrypt = require("bcryptjs");
const User = require("../models/User");
const sessionAuth = require("../middleware/sessionAuth");
const roleAuth = require("../middleware/roleAuth");

const router = express.Router();

router.post("/signup", async (req, res) => {
  const { email, password, role } = req.body;

  const exists = await User.findOne({ email });
  if (exists) return res.status(400).send("User exists");

  const hashed = await bcrypt.hash(password, 10);

  await User.create({
    email,
    password: hashed,
    role: role || "user"
  });

  res.send("User created");
});

router.post("/login", async (req, res) => {
  const user = await User.findOne({ email: req.body.email });
  if (!user) return res.status(401).send("Invalid credentials");

  const match = await bcrypt.compare(req.body.password, user.password);
  if (!match) return res.status(401).send("Invalid credentials");

  req.session.user = {
    userId: user._id,
    role: user.role
  };

  res.send("Login successful");
});

router.get("/profile", sessionAuth, (req, res) => {
  res.json(req.session.user);
});

router.get("/admin", sessionAuth, roleAuth("admin"), (req, res) => {
  res.send("Welcome Admin");
});

router.post("/logout", (req, res) => {
  req.session.destroy(() => {
```

```
    res.clearCookie("connect.sid");
    res.send("Logged out");
  });
});

module.exports = router;
```

src/server.js

```
const express = require("express");
const mongoose = require("mongoose");
const session = require("express-session");
const MongoStore = require("connect-mongo");
const csrf = require("csurf");
require("dotenv").config();

const authRoutes = require("./routes/authRoutes");

const app = express();
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB connected"));

app.use(
  session({
    name: "connect.sid",
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    store: MongoStore.create({
      mongoUrl: process.env.MONGO_URI
    }),
    cookie: {
      httpOnly: true,
      maxAge: 60 * 60 * 1000
    }
  })
);

// CSRF protection (ignore safe methods)
const csrfProtection = csrf({ ignoreMethods: ["GET", "HEAD", "OPTIONS"] });
app.use(csrfProtection);

// Endpoint to fetch CSRF token
app.get("/api/csrf-token", (req, res) => {
  res.json({ csrfToken: req.csrfToken() });
});

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

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