**# 2.2 JWT Login (jsonwebtoken)**

  const jwt = require('jsonwebtoken');

  // On successful login

  const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT\_SECRET, {

  expiresIn: '1h',

  });

**# 2.3 Express Rate Limiter**

  // In server.js

  const rateLimit = require('express-rate-limit');

  const limiter = rateLimit({

  windowMs: 15 \* 60 \* 1000, // 15 minutes

  max: 100, // limit each IP

  message: 'Too many requests, try again later.',

  });

  app.use('/api/', limiter);

**# 3. GOOGLE SSO SETUP**

**# 3.1 .env file**

  GOOGLE\_CLIENT\_ID=your\_client\_id\_here

  GOOGLE\_CLIENT\_SECRET=your\_client\_secret\_here

  JWT\_SECRET=your\_jwt\_secret

**# 3.2 config/passport.js**

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

  const passport = require('passport');

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

  passport.use(new GoogleStrategy({

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

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

  callbackURL: '/auth/google/callback',

  }, async (accessToken, refreshToken, profile, done) => {

  const user = await User.findOrCreate({ where: { googleId: profile.id } });

  return done(null, user[0]);

  }));

  passport.serializeUser((user, done) => done(null, user.id));

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

  const user = await User.findByPk(id);

  done(null, user);

  });

**# 3.3 Route**

  // In routes/auth.js

  router.get('/google', passport.authenticate('google', { scope: ['profile', 'email'] }));

  router.get('/google/callback',

  passport.authenticate('google', { failureRedirect: '/login' }),

  (req, res) => {

    res.redirect('/dashboard');

  });

**# 4. INPUT VALIDATION & SANITIZATION**

  // In routes/auth.js

  const { body, validationResult } = require('express-validator');

  router.post('/register',

    body('email').isEmail(),

    body('password').isLength({ min: 6 }),

    (req, res) => {

     const errors = validationResult(req);

     if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

     // Proceed with registration

   }

  );

**# 5. HELMET + CSP HEADERS**

// In server.js

const helmet = require('helmet');

app.use(helmet());

app.use(

  helmet.contentSecurityPolicy({

    directives: {

      defaultSrc: ["'self'"],

      scriptSrc: ["'self'", "https://apis.google.com"],

      styleSrc: ["'self'", "'unsafe-inline'"],

    },

  })

);

**# 6.RBAC MIDDLEWARE**

// middleware/role.js

module.exports = function (roles) {

  return (req, res, next) => {

    if (!roles.includes(req.user.role)) return res.status(403).json({ error: 'Access Denied' });

    next();

  };

};

**# 7. SESSION & COOKIE SETTINGS**

const cookieParser = require('cookie-parser');

app.use(cookieParser());

res.cookie('token', token, {

  httpOnly: true,

  secure: true,

  sameSite: 'strict',

  maxAge: 3600000,

});

**# 8. LOGGING WITH WINSTON + MORGAN**

const morgan = require('morgan');

const winston = require('winston');

const logger = winston.createLogger({

  transports: [

    new winston.transports.File({ filename: 'combined.log' }),

  ],

});

app.use(morgan('combined', {

  stream: {

    write: (message) => logger.info(message.trim()),

  },

}));

**# 9. DATABASE SECURITY**

// Sequelize ORM already uses parameterized queries internally

// Secure: Sequelize uses parameterized queries, preventing SQL injection

await User.findOne({ where: { email: req.body.email } });

---------------------------------------------------------------------------------------------------------------------------------------------------

**# Test Authentication Using Postman**