**Security Policy – SmartGlass Project**

**Name: Posaram Dewasi**

**Team: 14**

**Role: Cybersecurity Intern**

**Duration: Week 4 (18 July 2025)**

**##Overview**

This document outlines the security measures implemented in the SmartGlass project. It serves as a guideline to ensure the application is secure, compliant with best practices, and protected against common web vulnerabilities.

**## Authentication & Authorization**

* All private API routes are protected using JWT tokens.
* jwtAuth middleware verifies token validity before allowing access.
* Role-based access control ensures only educators can create sessions or upload content.

**## Secure Configuration**

* Sensitive data (JWT\_SECRET, Firebase credentials, DB URI) is stored in .env files.
* .env is listed in .gitignore to prevent accidental GitHub uploads.
* Environment-specific variables are not hardcoded in codebase.

**## Input Validation**

* User inputs (email, name, session title) are validated before processing.
* Invalid inputs return appropriate 400 Bad Request responses.
* Future plan: Use express-validator for stronger field validation.

**## CORS & HTTP Headers**

* CORS configured to allow only frontend domain http://localhost:5173.
* helmet middleware added to enforce secure HTTP headers.

**## Error Handling**

* API errors are returned in a clean, non-revealing format.
* Internal server or DB errors are logged but not shown to users.

**## Session and Token Management**

* JWT tokens are signed using strong secrets from .env.
* Tokens expire after 7 days for safety.
* Token expiry is checked on each private route.

**## 🔍 Monitoring and Logging**

* All login and upload actions are logged via console.log (basic monitoring).
* Repeated failed logins trigger warning logs.
* Suggestion: Implement winston or file-based logging for production.

**## Vulnerability Scanning**

* OWASP ZAP used to scan backend for open routes and exposed headers.
* Snyk CLI used to detect dependency vulnerabilities in package.json.
* Postman tests used to simulate XSS, SQLi, and auth bypass attempts.

**## Recommendations**

* + Use express-validator to sanitize inputs.
  + Add rate limiting on login/upload routes.
  + Implement email verification during signup.
  + Move logs to external secure file or monitoring system.
  + Rotate secrets monthly and enforce HTTPS in production

**## References**

* OWASP Top 10
* JWT Auth Docs
* Node.js Security Checklist
* Helmet.js and CORS Docs