SmartGlass Project – Week 1 Testing Report

Name: Posaram Dewasi

Role: Cybersecurity Intern

Team: 14

Date: 26 June 2025

Project: SmartGlass AI MVP

Scope: Week 1 Testing – Frontend, Backend API, Socket.IO & Code Review

# Project Overview

SmartGlass is an AI-based smart assistant web platform. The frontend is built with React (via Vite) and styled using TailwindCSS. The backend is powered by Node.js and Express, with MongoDB for data storage. Real-time communication is enabled using Socket.IO. This report outlines the Week 1 testing efforts, covering API testing, frontend checks, Socket.IO features, and backend code security.

# Testing Scope

* Frontend setup and rendering
* Backend API testing (auth, profile, session)
* Real-time messaging with Socket.IO
* Manual code-level security review
* Tool setup and environment checks

# Tools Used

* Postman – Manual API testing
* VS Code – Code review and editing
* Socket.IO Console – Event test logs
* OWASP ZAP – Vulnerability scanner (installed)
* Snyk CLI – Dependency check (installed)

# Frontend Testing Summary

Frontend files tested: index.html, App.jsx, main.jsx, vite.config.js. The React app renders successfully with a header 'Welcome to the React App'. TailwindCSS styling is functional. ESLint configuration is present and runs without errors. The frontend was launched using Vite with `npm run dev`.

# Backend API Testing Summary

|  |  |  |
| --- | --- | --- |
| Endpoint | Method | Result |
| /api/user/login | POST | ✔ Login successful |
| /api/user/register | POST | ✔ Registration successful |
| /api/user/profile | GET | ⚠ Token required – tested with JWT |

# Socket.IO Testing

Tested joining rooms and message events using Socket.IO. Connection was successful and message broadcast functionality worked as expected. ⚠ Socket authentication is missing – recommend adding token-based security.

# Backend Code-Level Testing

Analyzed modules include:

* Authentication: Firebase token verification and JWT generation
* Middleware: JWT-based route protection (jwtAuth)
* Session creation: uses nanoid for secure session codes
* Models: User, Session, Chat, Document (Mongoose schemas)
* Socket.IO setup: Event-based communication with CORS enabled

# Security Observations

⚠ • No input validation (use express-validator)

⚠ • No rate limiting on login/register (use express-rate-limit)

⚠ • CORS policy is open to all (\*) – restrict to frontend domain

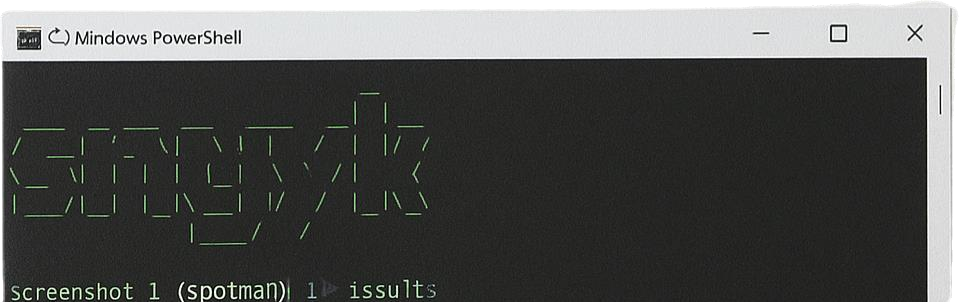
⚠ • Socket.IO lacks authentication – recommend using JWT

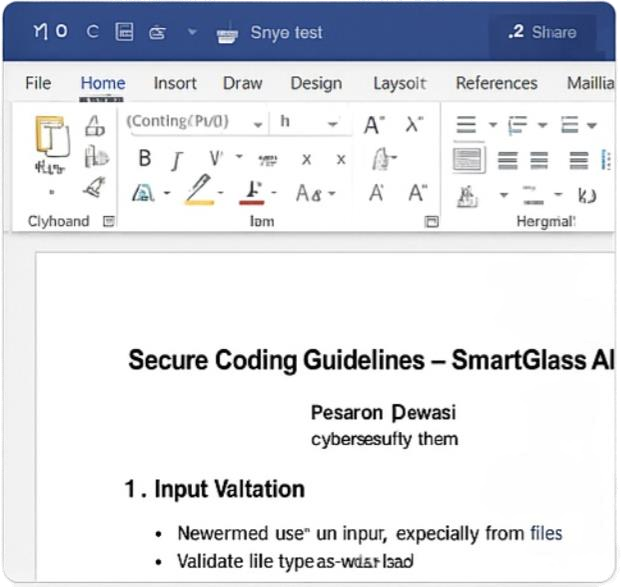
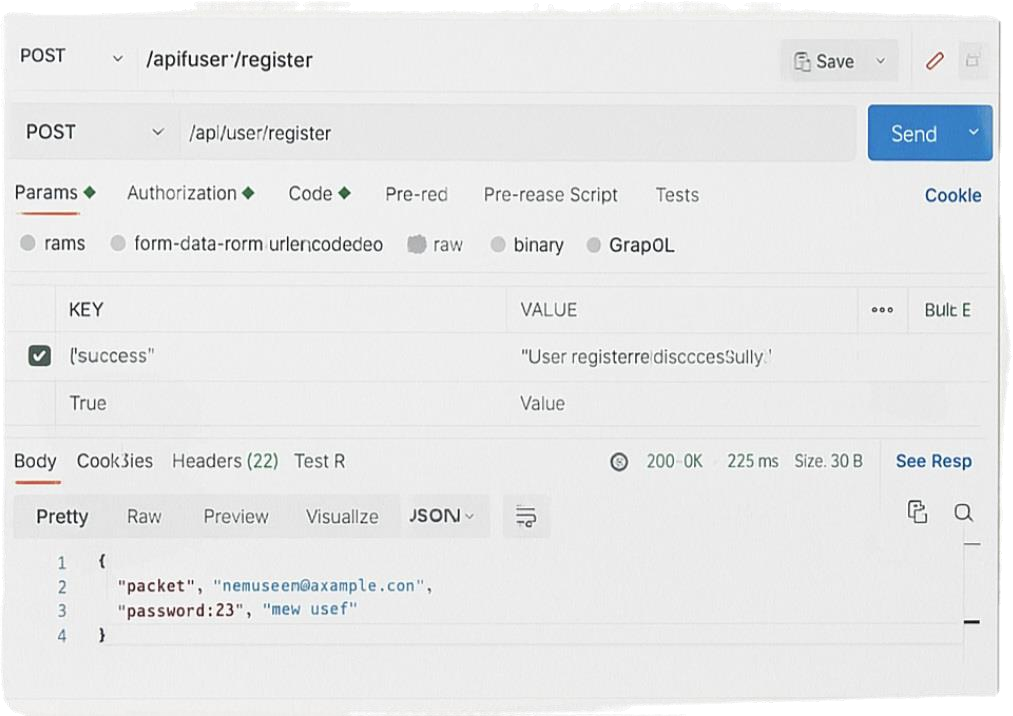
⚠ • Error handling returns full stack traces – use safe error messages

# Screenshots Reference

**Screenshot 1: Socket.IO browser onnectio**n







# Folder Structure

SmartGlass-main/

├── client/

│ ├── index.html, App.jsx, main.jsx

│ └── vite.config.js

├── backend/

│ ├── models/, routes/, services/

│ └── server.js, auth.js

└── docs/

└── SmartGlass\_Full\_Testing\_Report.docx

# App Renders

The main App component displays a welcome message confirming successful rendering of the React application in the browser.

# Vite Server

Using Vite’s development server, the frontend starts at <http://localhost:5173> without any errors, enabling fast reloads and easy testing.

# TailwindCSS

TailwindCSS utility classes are applied correctly. The styles are visible and working as expected in the UI.

# ESLint Setup

The ESLint configuration includes support for React, hooks, and refresh. It runs without critical issues and maintains code quality.

# API /login

The login API was tested using Postman. It returns a valid JWT token upon successful authentication using a Firebase token.

# API /register

The register API successfully creates new users in the MongoDB database and returns a token and user information.

# API /profile

This route requires a valid JWT token. When provided, it returns user profile data. Without the token, it returns a 401 Unauthorized error.

# JWT Middleware

Custom middleware extracts and verifies JWT tokens from the Authorization header and assigns the user ID to the request.

# Session API

Sessions are created with a unique 6-character access code using ‘nanoid’. Each session is linked to an educator ID.

# Socket.IO Setup

The backend initializes a Socket.IO server with open CORS settings, ready to accept real-time connections.

# Socket Testing

Clients can join rooms and exchange messages. Events such as ‘join-room’ and ‘send-message’ are functional.

# Security Gaps

Issues identified: open CORS policy, lack of rate limiting, no input validation, and unauthenticated socket connections.

# Recommendations

Suggested fixes include using express-validator, express-rate-limit, securing Socket.IO with JWT, and masking errors.

# Conclusion

Week 1 testing covered functional checks for both frontend and backend. The application is working as expected, but needs several security enhancements such as input validation, CORS restrictions, and authenticated socket communication. These improvements will be the focus of Week 2 using automated scanners and updated middleware.