# Secure Software Design Project

# CY-321


## Project Phase 3:

## System Architecture and Secure Design

## Project Team

Taha Juzar-2022585

Ahmer Ayaz-2022070

Bashir Ahmed-2022646

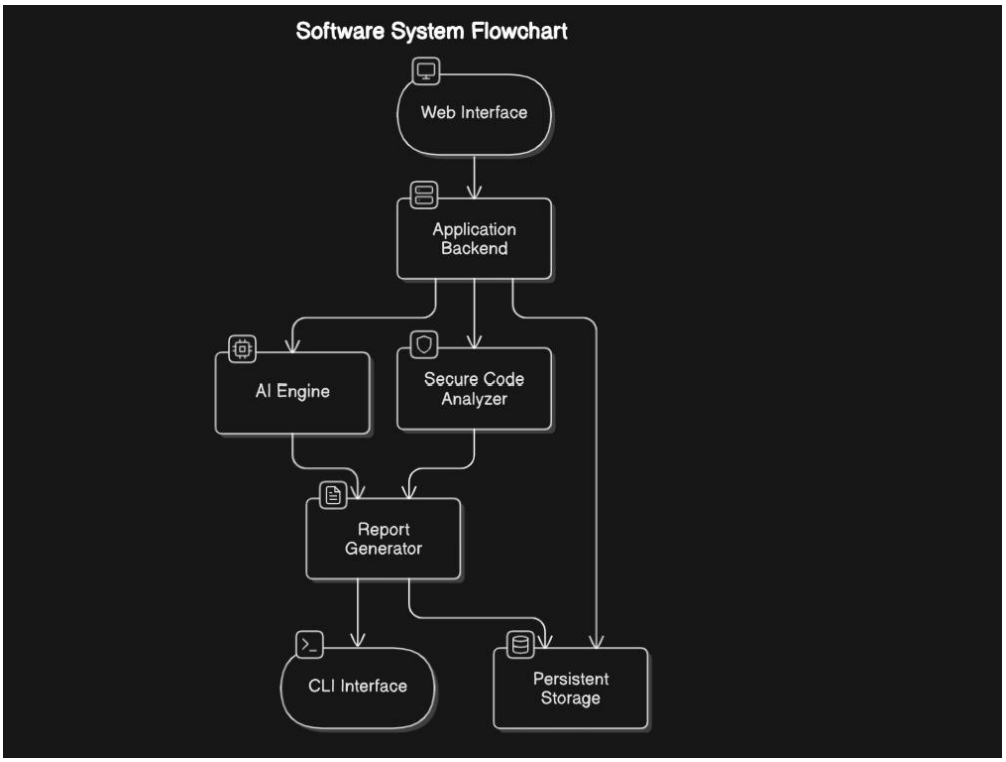
Younus Khan-2022455

## Submitted to: Dr Zubair Ahmed

## Code Sentinel

The advanced security solution code Sentinel relies on AI to realize its mission of improving codebase security through vulnerability detection and secure coding standards regulation. The system applies artificial intelligence for extensive security evaluations and identifies dangers to stop possible threats before attackers can exploit them. Secure applications can be built efficiently through builders' access to detailed reports and practical recommendation features within this tool.

## Architecture Diagram of Code Sentinel:




The above architecture diagram illustrates a dual-interface system where users interact via a Web UI or CLI, both routed through a backend API layer. The backend connects to the AI Engine for ML-based vulnerability detection and a Secure Code Analyzer for static analysis. Results are processed by the Report Generator and stored securely in a persistent database.

## Security Controls

1. **Authentication**

- Web UI & CLI Access:
  OAuth2-based authentication for web users.
  API keys or token-based authentication for CLI interactions.

- Role-Based Access Control (RBAC):
  Different roles like admin, developer, and viewer control what each user can access or edit.

2. **Encryption**

- Data-in-Transit:
  HTTPS/TLS for secure communication between frontend, backend, and storage.

- Data-at-Rest:
  AES-256 encryption for storing reports, user credentials, and sensitive data in the database.

- Token Storage:
  Use of hashed and salted tokens (e.g., bcrypt) for authentication credentials.

3. **Access Control**

- File Upload Restrictions:
  Scan uploaded code for malware and restrict executable files.

- Code Execution Sandbox:
  If live testing is enabled, run code in isolated containers (e.g., Docker) with resource limits.

- Audit Logs:
  Maintain logs for every user action (e.g., scans, logins, downloads) to monitor suspicious activity.

## Security Design Measures

1. **OWASP Compliance**

Vulnerability detection aligned with OWASP Top 10 (SQLi, XSS, Insecure Deserialization, etc.).
Regular updates to vulnerability signatures and detection logic.

2. **Static Code Analysis Security**

- Detects:
  Insecure functions/APIs
  Missing input validations
  Hardcoded secrets
  Suggest secure alternatives with references to CWE and NIST standards.

3. **Secure Defaults**
   CLI/Web app restricts full file system access.
   Safe default configuration values (e.g., minimal privileges, disabled debugging in production).

4. **Input Validation**
   All inputs (CLI parameters, web inputs) are sanitized to prevent injection attacks.

5. **Dependency Checking**
   Third-party libraries are checked for known vulnerabilities using tools like pip-audit or Safety.

6. **Continuous Monitoring**
   Future support for integration with GitHub Actions or CI/CD pipelines to run security checks automatically.