***This essay is written entirely in English, as the integration of Hebrew alongside extensive English terminology would compromise clarity and textual consistency.***

***Executive Summary:***

This report outlines the security assessment process, combining automated ZAP scans with manual code reviews to identify key vulnerabilities. These issues were carefully analyzed, explained, and addressed through targeted fixes supported by code samples. Building on these solutions, the report provides recommendations for secure development practices that strengthen protection against common threats. Together, these steps form a continuous process that not only resolves present risks but also reinforces long-term resilience. In doing so, the assessment ensures that applications remain aligned with core security principles. Ultimately, this approach upholds the cyber **C.I.A** triad of confidentiality, integrity, and availability.

***Testing Methodology:***  
The application was tested using a multi-layered security approach:

1. **Automated Code Scanning:** Using **Zed Attack Proxy (ZAP)** to identify common vulnerabilities in both code and server communication.
2. **Manual Review:** Analyzing the code and application for vulnerabilities according to the **OWASP Top Ten**, including input validation, authentication logic, and security best practices.
3. **Penetration Testing:** Verifying that detected vulnerabilities were exploitable and ensuring they are addressed appropriately.

***Detected Vulnerabilities***

**1. Cryptographic Failures**

* **Description:** Passwords are transmitted to the server without encryption, and there is no secure verification against stored data. Sensitive information, such as passwords, is handled in plaintext.
* **Risk Level:** Critical
* **Impact Scope:** Entire application
* **Detection Method:** Manual review – OWASP Top Ten
* **Implications:** Exposure of passwords may allow unauthorized account access, identity theft, or misuse of sensitive information.

**2. SQL Injection**

* **Description:** User input is not properly validated or sanitized, allowing special characters to be inserted, potentially enabling malicious SQL queries.
* **Risk Level:** Critical
* **Impact Scope:** All input fields
* **Detection Method:** Manual review – OWASP Top Ten
* **Implications:** Exploiting this vulnerability could result in modification, deletion, or theft of database information.

**3. Security Logging and Monitoring Failures**

* **Description:** No centralized logging system exists to track critical events such as successful or failed login attempts, or important user actions like post creation or deletion.
* **Risk Level:** High
* **Impact Scope:** Server-side
* **Implications:** Lack of monitoring prevents timely detection of attacks or malicious activity.

**4. Content Security Policy (CSP) Header Not Set**

* **Description:** CSP headers are not configured to restrict which sources the browser is allowed to load resources from.
* **Risk Level:** Medium
* **Impact Scope:** Client-side
* **Detection Method:** Automated scan – ZAP
* **Implications:** The application is exposed to XSS attacks and potential content injection from third-party sites.

**5. Cross-Site Scripting (XSS)**

* **Description:** Lack of input sanitization, particularly when displaying content using innerHTML, allows injection of malicious scripts.
* **Risk Level:** High
* **Impact Scope:** Client-side
* **Detection Method:** Manual and automated scans
* **Implications:** XSS attacks can steal cookies, access user data, or perform actions on behalf of the user without their consent.

**6. Missing Anti-Clickjacking Header**

* **Description:** Headers such as X-Frame-Options or Content-Security-Policy: frame-ancestors are missing, leaving the application vulnerable to clickjacking.
* **Risk Level:** Medium
* **Impact Scope:** Client-side
* **Detection Method:** Automated scan – ZAP
* **Implications:** Clickjacking attacks could trick users into performing unintended actions on the site.

***Appendix***:

1 – Cryptographic Failures :   
Before :   
תמונה שמכילה טקסט, צילום מסך, גופן, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.  
After :   
תמונה שמכילה טקסט, צילום מסך, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.  
  
2.SQL injections :   
Before :

תמונה שמכילה טקסט, צילום מסך, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

After :   
תמונה שמכילה טקסט, צילום מסך, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

This is in regard to the login /sign up , here we can clearly see an improvement – users are no longer able to use special characters in their names , ones that could pose threat to SQL queries.   
The same applies to the Posts payloads contents , But with a bit more freedom , such as dots , and slashes , as can be see in the before :  
  
תמונה שמכילה טקסט, צילום מסך

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

And this is the posts boundaries after:   
תמונה שמכילה טקסט, צילום מסך, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.  
תמונה שמכילה טקסט, צילום מסך, גופן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.  
Here we can clearly see a change , here we validate our post content , to prevent such attempts.

3. **Security Logging and Monitoring Failures :**

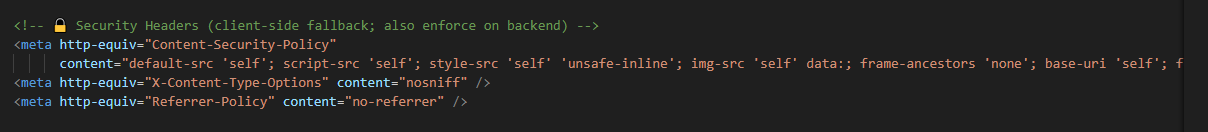
I have implemented a logging and monitoring system that creates a log for every time a user tries to sign up , log in , and create a post , each log has a time stamp , and a success factor , meaning it also logs unsuccessful attempts of logging in / signing up , etc.   
This is how it looks like :

תמונה שמכילה טקסט, צילום מסך, תצוגה, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

4. **Content Security Policy (CSP) Header Not Set**

Added headers to the HTML files.

**5. Cross-Site Scripting (XSS)**

We can patch the cross site scripting vulnerability by writing code in such a way , that users would not be able to inject scripts into our browsers , we can do that by regulating our input boxes (as was done with the SQL injections) and to encode our website like so that it will not recognize code as a code , but rather as plain text. We can also add the CSP headers to make it easier for us to make sure we aren’t getting tricked.  
To use plain text instead of code , we would just use the “textContent” attribute in java script , instead of the “InnerHTML”.  
as can be seen here in the repaired result :   
תמונה שמכילה טקסט, צילום מסך, תוכנה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.

6. **Missing Anti-Clickjacking Header  
  
We can fix this vulnerability by just adding the CSP headers to our front pages , as we did in the fourth section.**

***Recommendations :***

I would like to take the initiative and talk about some recommendations that I could implement to this code , these recommendations would make the application over all more secure , easier to maintain , and make it easier for us in the future to deal with exploiters.  
  
1. Transitioning to HTTPS , currently the app only works on HTTP , if we were to migrate to https it would greatly advance our security , by enabling data encryption over the web.  
  
2. Enable MFA , by enabling the MFA we could ensure that even if we have a data leak , users would be significantly less effected by it.

3.Transitioning to a real database , in the current state of the project we are using plain files in order to save data , these files are local and can be messed up with by the downloaders of the application , which means its susceptible to being changed and modified by anyone who downloads it , which causes issues down the line.

4.Installing a WAF – an over-the-web firewall could be of great help to ensure that suspicious calls from over the web don’t enter our website , and as a result we are able to ease the traffic on the server.

5.Manage dependencies – its important to know what dependencies you are using , and to keep track of their development , to make sure that when a vulnerability in them is discovered , and an update comes out , you immediately install it to patch it , or if an update is yet to come out , you disable them. As well as making sure they are not copyrighted of course.