Assignment Guidance and Front Sheet

This sheet is to be populated by the Module Tutor, checked by the Programme Team, and uploaded to Moodle for students to fill in their ID and submit with their assessment.

Student ID or IDs for group work	U2136249

Module Title & Code	*WM240, Cyber Context of Software Engineering*
Module Owner	*Hassan Raza*
Module Tutor	*Hassan Raza*
Module Marker	*Hassan Raza*
Assesment type	*Coursework*
Date Set	*Cw2 – 13-3-23*
Submission Date (excluding extensions)	*Cw2 – 19-5-2023*
Marks return date (excluding extensions)	*Cw2 - 18-06-2023*
Weighting of mark	*Cw2 - 50%*

Assessment Detail	*See individual specifications*
Additional details	*See individual specifications*
Module learning outcomes (numbered)	*1) Apply cyber security good practice to various phases of the software engineering lifecycle* *2) Critically reflect on the development of a software project* *3) Demonstrate the understanding and application of relevant software development frameworks to a given software development scenario*

Learning outcomes assessed in this assessment (numbered)	*1, 2, 3: cw2*
Marking guidelines	*See individual specifications*
Submission guidance	*See individual specifications*
Academic Guidance	*support in timetable lab sessions*
Resubmission details	*A combination of CW1 and CW2 with a new case study i.e., paras 2, 3, 4 in the Introduction will be replaced by a new case study*
Late submission details	If work is submitted late, penalties will be applied at the rate of **5 marks per University working day** after the due date, up to a **maximum of 10 working days** late. After this period the mark for the work will be reduced to 0 (which is the maximum penalty). "Late" means **after the submission deadline time as well as the date** – work submitted after the given time even on the same day is counted as 1 day late.

2A: Addressing Vulnerabilities Process

During the development phase of the global-finance digital (GFD) platform, I performed security testing methods to identify and analyse potential vulnerabilities within the GFD platform. I utilised Static Application Security (SAST) and Web Application Scanning (WAS), providing me with an in-depth analysis of the application's security, which allowed me to ensure that the platform is not only functional but also secured from potential threats.

Eight significant security issues were identified, these are detailed in Figure 1:

No.	Alert Type	Risk
1.	Cross-Site Scripting (XSS)	High
2.	Absence of Anti-CSRF Tokens	Medium
3.	CSP: Wildcard Directive	Medium
4.	Content Security Policy (CSP) Header Not Set	Medium
5.	Missing Anti-clickjacking Header	Medium
6.	Cross-Domain JavaScript Source File Inclusion	Low
7.	Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low
8.	X-Content-Type-Options Header Missing	Low
9.	Information Disclosure - Suspicious Comments	Informational

Figure 1 - Security Vulnerabilities Identified in the GFD platform.

I utilised Synk for SAST analysis and OWSAP ZAP for a more general WAS scan. When using the SAST analysis tool, I was able to discover a high-risk vulnerability that was missing from the WAS scans.

Figure 2 - XSS Vulnerability Present in GFD platform

Figure 2, indicates an existent Cross-site Scripting (XSS) vulnerability, classified as high-risk. vulnerability present in the GFD platform. This is because in line 39, the input from the request URL that flows into "send" function, where it is used to render an HTML page to the user is not validated properly. This could allow a threat actor – malicious actor with intent to cause harm or pose a threat (CSRC, 2013), to trick the application to accept a request as originated from a trusted source to perform malicious actions by injecting arbitrary code that are otherwise is blocked by the browser's Same Origin Policy (CWE, 2006). To prevent this type of attack, line 39 should be modified to sanitise the data input in the HTTP request before reflecting it back, to ensure all data is validated

before echoing anything back to the user. A potential fix could be: “res.send({ files: req.files, body: req.body });”, these changes will redirect invalid requests preventing malicious code to be injected (PortSwigger, 2017).

Figure 3 - CSRF Vulnerability found in GFD platform.

Figure 3 shows a **Cross-Site Request Forgery (CSRF)** vulnerability that was found on both analyses. CSRF occurs when an attacker takes advantage of a user’s authenticated credentials such as a browser cookie to impersonate that trusted user and perform unauthorised actions (MITRE, 2009). In this case, the application has CSRF protection disabled, which could allow attackers to execute requests on a user’s behalf, this is because the web application is not set up to differentiate between legitimate and malicious requests. To prevent this attack, code could be added to implement hidden tokens that are checked to confirm state-change requests and never assume session identifiers as a mean to validate legitimate request (Kirsten, 2023).

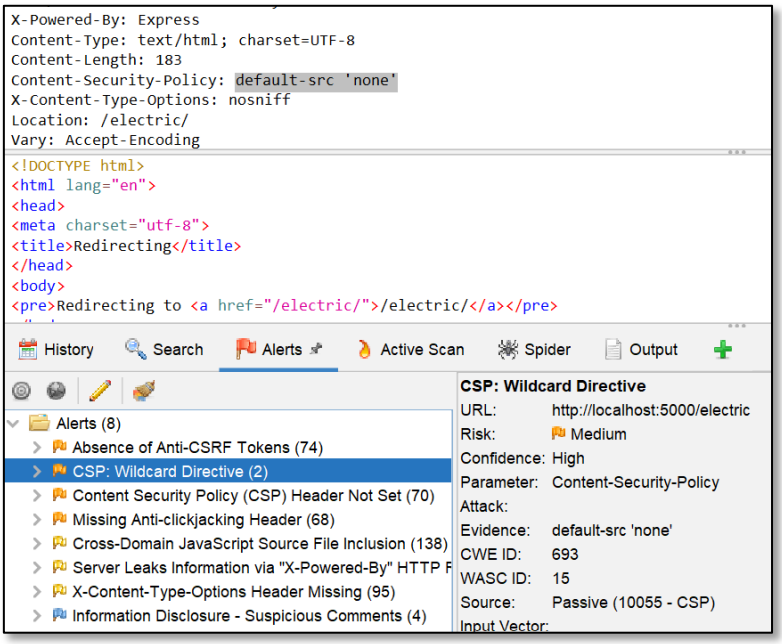

Figure 4 - CSP Vulnerability found in GFD platform.

Figure 4 shows a **Content Security Policy (CSP)** wildcard Directive vulnerability. The CSP is an added layer of security that detects and mitigates XSS , it provides a set of standard HTTP headers that allows admins to declare approved sources of contents that browsers should only be allowed to load (CWE, 2014). To prevent this the application should implement anti-CSRF tokens to verify that

the user is intentionally making a request (Mozilla, 2019). For example, to allow content from a trusted domain and all its submains the following code can be added to the header:

"Content-Security-Policy: default-src 'self' example.com *.example.com"

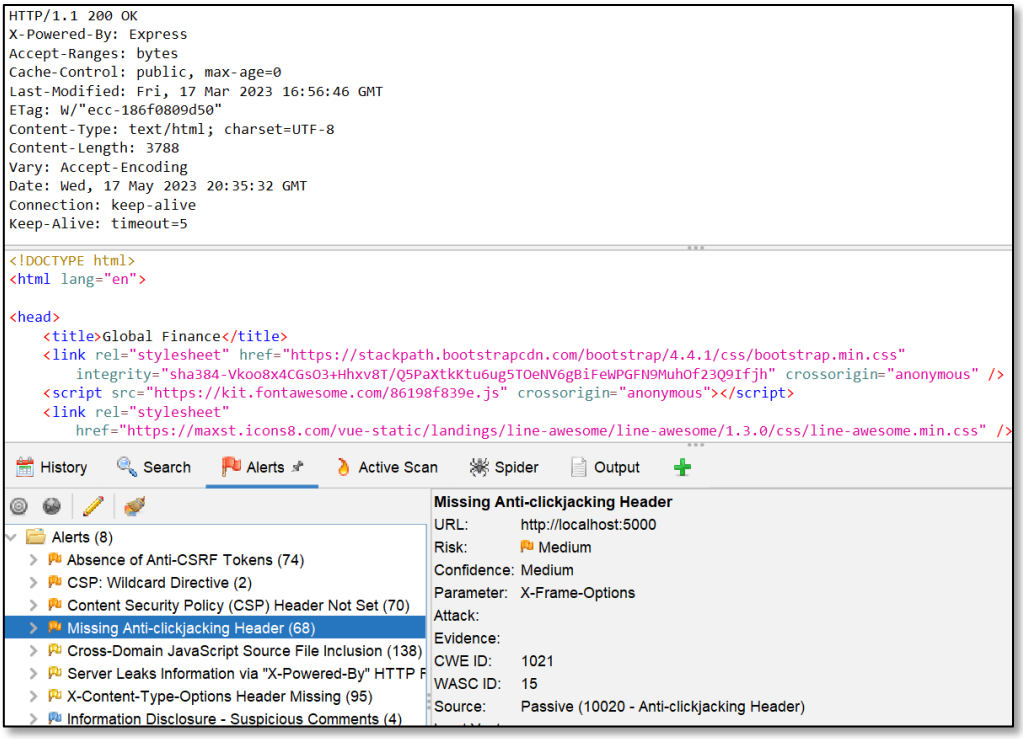

Figure 5 - Missing Anti-ClickJacking Header in GFD platform

Missing Anti-clickjacking Header also known for "UI redress attack", when an attacker uses multiple opaque UI layers to trick a user into clicking a button or link (Rydstedt, 2022). Figure 5 illustrates that the web application does not include X-Frame Options to protect against "ClickJacking" attacks. To resolve this issue GFD platform must implement X-Frame-Options HTTP response headers across all pages, to instruct the browser to not display a page within a frame (CWE, 2019). This can be done by sending proper CSP frame-ancestors directive response headers that instruct the browser to not allow other domains, other options include setting authentication cookies with "SameSite=Strict" attributes.

Other less significant vulnerabilities were found including the Disclosure of information though comments. This risk issue could lead to the exposure of sensitive information by looking at the source code. To solve this, it is critical to perform a review of all comments in web application and remove any that could disclose information about the system which could be used for malicious purposes.

2B: Azure instance Implementation

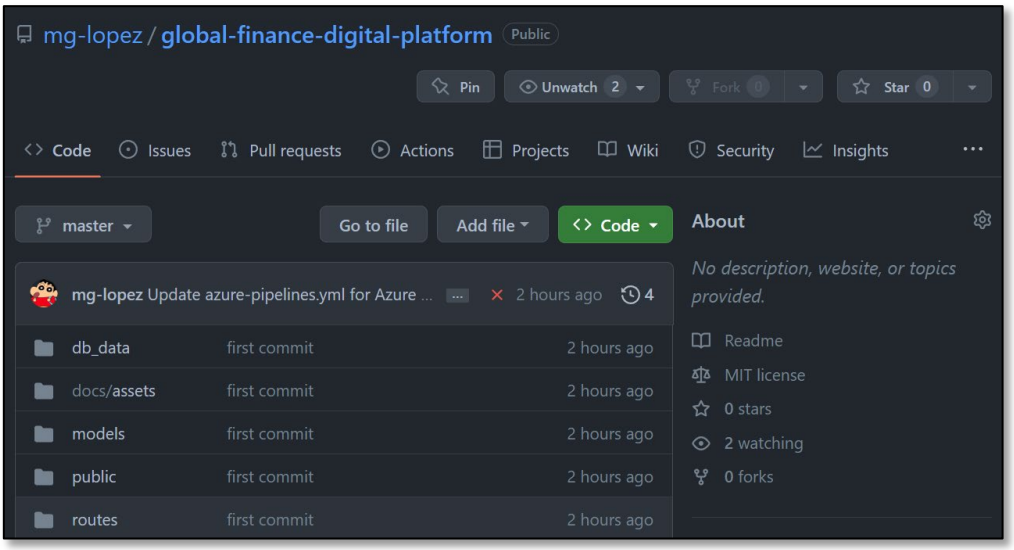

Figure 6 - GitHub Repository of GFD Platform

To automate the software development process and ensure continuous integration and continuous delivery (CI/CD), I have implemented a CI/CD using Azure DevOps.

I began by creating a GitHub repository (https://github.com/mg-lopez/gfinances) where all the code resides. Then I initiated an Azure pipeline by creating a file named “azure-pipelines.yml” and stored it under the root directory of the repository.

```
steps:
- task: NodeTool@0
  inputs:
    versionSpec: '10.x'
  displayName: 'Install Node.js'

- script: |
    npm install
    npm run build --if-present
    npm run test --if-present
  displayName: 'npm install, build and test'
- task: ArchiveFiles@2
  displayName: 'Archive files'
  inputs:
    rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
    includeRootFolder: false
    archiveType: zip
    archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
    replaceExistingArchive: true

- upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
  artifact: drop
```

Figure 7 – Contents of "azure-pipelines.yml” file

Figure 7 shows the contents of the azure pipeline, it first installs the latest version of Node.js which is the engine in which the GFD platform was developed. It follows by installing and building all the required dependencies, once that is completed, it then comprises all the files into a .ZIP file to improve the overall performance when automating the systems updates as creates less overhead for the server.

```yaml
- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  condition: succeeded()
  jobs:
  - deployment: Deploy
    displayName: Deploy
    environment: $(environmentName)
    pool:
      vmImage: $(vmImageName)
    strategy:
      runOnce:
        deploy:
          steps:
          - task: AzureWebApp@1
            displayName: 'Azure Web App Deploy: gfinance'
            inputs:
              azureSubscription: $(azureSubscription)
              appType: webAppLinux
              appName: $(webAppName)
              runtimeStack: 'NODE|10.10'
              package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
              startUpCommand: 'npm run start'
```

Figure 8 - Deploy GFD platform to Azure Web App Service

Figure 8, indicates that once all the files are compressed, the pipeline will create an Azure Web App service which will be used to host all the files from the GitHub repository to the Azure Cloud so that it can be used to deploy GFD platform to the wider internet.

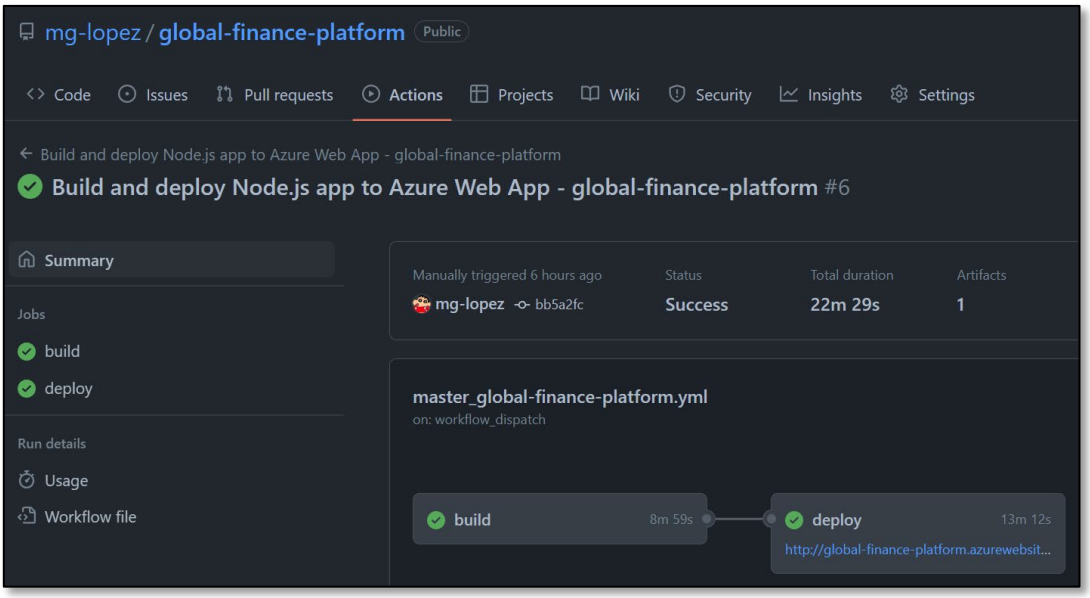

Figure 9 - GFD platform developed using GitHub & Azure DevOps

After all of this is done, I proceed to deploy the pipeline, which was configured to trigger a build and deploy the application whenever a commit is pushed to the GitHub repository. Figure 9, illustrates the first build of the application located at the following URL: https://gfinance.azurewebsites.net/home.

2C: Reflections

Upon reflection, this project has served as a helpful journey into the world of software development, security testing and automation. It has introduced me a newfound appreciation for the complexities inherent in the field, and the critical importance of robust security measures.

Addressing these vulnerabilities was a challenging but rewarding process. It has provided me with a practical insight into potential security threats that a real-world application might face, as well as strategies to mitigate such threats. It has also been a humbling experience to recognise the vulnerabilities in my own code, reminding me of the constant need for vigilance in design and approach choices.

The task of setting a CI/CD pipeline in Azure DevOps gave me a first-hand experience to the power of automation in software development, it was fascinating to see how automated processes could streamline software updates, eliminating all the manual intervention.

Despite the fixes of the vulnerabilities listed above, I am cognisant for further improvements. For instance, a more user-friendly interface could enhance the website's design. In addition, the incorporation of more exhaustive testing measures such as Dynamic Application Security Testing, could further reinforce the identification of potential vulnerabilities.

On the design of the website, I considered several improvements:

1. **Streamlining Features and Buttons**: the prototype has an extensive set of features, which may be overwhelming for some users (Hotjar, 2019). Therefore, I am planning to streamline some of the features and buttons, to prioritise the most impactful ones, making the platform much easier to use.
2. **Consistency of Look and Feel:** Moving forward, I plan to ensure a consistent UI design across the entire web application. Maintaining the same look, feel and navigation ensures that users feel comfortable browsing the platform (de la riva, 2018).
3. **Regular Performance and User Experience Testing:** it is essential that the platform is regularly tested to ensure it meets user needs. This could involve running web monitoring test for availability, speed and security, as well as testing new concepts and aesthetic choices from customer feedback (Roose, 2021).

In conclusion, this project has been a profound learning experience, providing me with practical exposure to critical aspects of DevOps development methodology. The hand-on experience has equipped me with a deep understanding and a solid foundation of software development, valuable for my future advancements in the industry.

References

CSRC (2013). *threat actor - Glossary | CSRC*. [online] csrc.nist.gov. Available at: https://csrc.nist.gov/glossary/term/threat_actor.

CWE (2006). *CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (4.1)*. [online] cwe.mitre.org. Available at: https://cwe.mitre.org/data/definitions/79.html.

CWE (2014). *CWE - CWE-693: Protection Mechanism Failure (4.3)*. [online] cwe.mitre.org. Available at: https://cwe.mitre.org/data/definitions/693.html.

CWE (2019). *CWE - CWE-1021: Improper Restriction of Rendered UI Layers or Frames (4.4)*. [online] cwe.mitre.org. Available at: https://cwe.mitre.org/data/definitions/1021.html.

de la riva, maria (2018). *Why Consistency Is So Incredibly Important In UI Design*. [online] Careerfoundry.com. Available at: https://careerfoundry.com/en/blog/ui-design/the-importance-of-consistency-in-ui-design/.

Hotjar (2019). *8 Web App Design Best Practices to Improve UX*. [online] www.hotjar.com. Available at: https://www.hotjar.com/web-app-design/best-practices/ [Accessed 19 May 2023].

Kirsten (2023). *Cross Site Request Forgery (CSRF) | OWASP*. [online] owasp.org. Available at: https://owasp.org/www-community/attacks/csrf.

MITRE (2009). *CWE - CWE-352: Cross-Site Request Forgery (CSRF) (3.4.1)*. [online] Mitre.org. Available at: https://cwe.mitre.org/data/definitions/352.html.

Mozilla (2019). *Content Security Policy (CSP)*. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

PortSwigger (2017). *What is cross-site scripting (XSS) and how to prevent it?* [online] Portswigger.net. Available at: https://portswigger.net/web-security/cross-site-scripting.

Roose, J. (2021). *How to Conduct Usability Testing in Six Steps*. [online] Toptal Design Blog. Available at: https://www.toptal.com/designers/ux-consultants/how-to-conduct-usability-testing-in-6-steps.

Rydstedt, G. (2022). *Clickjacking | OWASP*. [online] owasp.org. Available at: https://owasp.org/www-community/attacks/Clickjacking.

Synk (2018). *DOM Based XSS | Tutorial & Examples*. [online] Snyk Learn. Available at: https://learn.snyk.io/lessons/dom-based-xss/javascript/ [Accessed 18 May 2023].

Synopsys (2019). *What Is SAST and How Does Static Code Analysis Work? | Synopsys*. [online] www.synopsys.com. Available at: https://www.synopsys.com/glossary/what-is-sast.html#:~:text=Static%20application%20security%20testing%20(SAST.