SSA SECURITY FUNDAMENTALS AND ASSESSMENT REPORT

THE UNSECURE PWA COMPANY

UNSECURE PWA Security Report

<Assessor Details>

Document details

Assessment

Date	30/04/2025
Control classification	Choose an item.
Definition	Choose an item.
Deployment model	Choose an item.
Version	V1.0

Prepared by examplecompany

Address	123 example rd	
Assessor name	Jackson Mackey	
Assessor qualifications	none	
Contact email		

Prepared for The Unsecure PWA Company

Address	123 example ave	
Contact name		
Contact email		

Revision history

Version	**Date**	**Description**	**Author**
1.0	30/04/2025	report finished	Jackson

Adapted from Australian Signals Directorate's Australian Cyber Security Centre (2022).

Contents

1. Executive summary

1.1 Briefly outline the purpose of the report.

The purpose of this report is to outline what the client, The Unsecure PWA Company, wants in terms of information security improvements to their system. Then to provide a brief analysis on what areas they need improvement on, then demonstrate the measures implemented to prevent future and current vulnerailties.

1.2 Highlight key findings and recommendations regarding security practices in software development.

Key findings from the AUS task on the Unsecure PWA highlight common security lapses in software development, including inadequate input validation, missing HTTPS implementation, poor authentication controls, and insecure storage of user data. These issues expose applications to risks such as XSS, data breaches, and session hijacking. To mitigate these threats, developers should adopt secure coding practices, enforce HTTPS, validate and sanitize all inputs, implement strong authentication mechanisms, and follow the principle of least privilege. Regular security testing and adherence to standards like OWASP guidelines are essential for building secure applications.

2. Introduction

2.1 Overview of ‘The Unsecure PWA Company’.

The client, ‘The Unsecure PWA Company’, has engaged us as a software engineering security specialist to provide expert advice on the security and privacy of their application.

This progressive web app is currently in the testing and debugging phase of the software development lifecycle and can be accessed here: Unsecure PWA.

2.2 Importance of secure software architecture in today’s digital landscape.

- Reduced risk of data breaches and unauthorised access.
- Enhanced trust and credibility with users.
- Lower maintenance costs associated with fixing security issues.

Table 21 – latest trends impacting the internet and cybersecurity

Definition	Characteristics	Example
Data protection is the process of protecting sensitive information from damage, loss or corruption.	Confidentiality - Data is only disclosed to those its owned by Integrity - Data isnt modified unintentionally or maliciously Availability - Data is easily and always available to correct users	Data security Access controls Storage requirements
Cyber attacks are acts performed by individuals with harmful intent, whose goal is to steal data, cause damage or to disrupt computing systems	Include malware, social engineering, man in the middle, denial of service, and injection attacks	Common examples of cyber attacks: malware, phishing, man in the middle, denial of service, SQL injection, zero day exploit, domain name system tunneling.
Static application security testing (SAST), are white box tools to inspect source code	A developer's approach - testers have access to underlying framework, design and implementation of vulnerabilities early in the development cycle without executing the program. Code is analysed more quickly than humans, and real-time feedback, with graphical representations show the exact location of vulnerabilities and problem code.	Some examples of SAST tools include, Aikido security, SAST, checkmarx, gitlab, sonar.
Dynamic application security testing (DAST)These tools are black box tools to	A hacker approach, no knowledge of internals. Testing highlights authentication and server	Common examples of cyber attacks: malware, phishing, man in the middle, denial of service, SQL injection, zero day exploit, domain

Definition	Characteristics	Example
products during operation and provide feedback on compliance and generally security.	configuration issues and is language dependent.	name system tunneling.
Vulnerability assessment is a systematic review of security weaknesses in an information system. It evaluates if the system is susceptible to any known vulnerabilities, assigns severity levels to those vulnerabilities, and recommends remediation or mitigation, if and whenever needed.	The security scanning process consists of four steps: testing, analysis, assessment and remediation.	SQL injection, XSS and other code injection attacks. Escalation of privileges due to faulty authentication mechanisms. Insecure defaults – software that ships with insecure settings, such as a guessable admin passwords.
Penetration testing, is a simulated cyber attack against your computer system to check for exploitable vulnerabilities.	Pen testing can involve the attempted breaching of any number of application systems, (e.g., application protocol interfaces (APIs), frontend/backend servers) to uncover vulnerabilities	injection attacks, XSS, SQL injection
API security is a software intermediary that allows your applications to communicate with one another	● Implement Access Control ● Implement Data Classification ● Validate Parameters ● Encrypt API Requests and Responses ● Continuous Security	Improper API security involves ● Broken Object-Level Authorization ● Broken User Authentication ● Excessive Data Exposure ● data breaches
Cross-site scripting (XSS) is one of the most common methods used	XSS allows execution of JavaScript code, so the damage that can be done	Escape Dynamic Content It consists of replacing significant characters with the HTML entity

Definition	Characteristics	Example
to attack websites. It allows a malicious user to execute arbitrary chunks of JavaScript when other users visit your site. It is the most common publicly reported vulnerability	by an attacker depends on the sensitivity of the data being handled by your site. SOme things you can do are ● Spreading worms on social media sites ● Session Hijacking ● Identity theft ● Denial of service attack and website vandalism ● Theft of sensitive data ● Financial Fraud	encoding. Escaping editable content means it will never be treated as executable code by the browser, effectively closing the door for most attacks. USe HTML.escape to prevent, along with sanitising and validating data. Allowlist Values If a particular dynamic data item can only take a handful of valid values, the best practice is to restrict the values in the data store, and have your rendering logic only permit known good values. For instance, instead of asking a user to type in their country of residence, have them select from a drop-down list. Implement a Content-Security Policy Browsers support Content-Security Policies that allow the author of a web-page to control where JavaScript (and other resources) can be loaded and executed from. XSS attacks rely on the attacker being able to run malicious scripts on a user's web page - either by injecting inline <script> tags somewhere within the <html> tag of a page, or by tricking the browser into loading the JavaScript from a malicious third-party domain.By setting a content security policy in the response header, you can tell the browser to never execute inline JavaScript, and to lock down which

Definition	Characteristics	Example
		domains can host JavaScript for a page.
Cross-site request forgery (CSRF) is where vulnerabilities can be used to trick a user's browser into performing an unwanted action on your site.	Any function that your users can perform deliberately is something they can be tricked into performing inadvertently, with the most malign cases involving attackers spreading themselves as a worm. They have also been used to ● Steal confidential data ● Spread worms on social media ● Install malware on mobile phones	Representation State Transfer (REST) ● A series of design principles that assign certain types of actions to different HTTP verbs. By following a REST-ful design, it will keep your code clean and help your site scale. Moreover, REST insists that GET rvb requests are used only to view resources. Keeping your GET requests side effect free will limit the harm that can be done by malicious URLS. Anti-Forgery Tokens ● Even when edit actions are restricted to non-GET requests, you are not entirely protected. POST requests can still be sent to your site from scripts and pages hosted on other domains. In order to ensure you only handle valid HTTP requests, you need to include a secret and unique token with each HTTP response, and have the server verify that token when it's passed again using the POST method. Each time

Definition	Characteristics	Example
		your server renders a page that performs sensitive actions, it should write out an anti-forgery token in a hidden HTML form field. This token must be included with form submissions, or AJAX calls.

3. Benefits of developing secure software

3.1 Data protection

3.1.1 Explanation of the significance of protecting sensitive data.

Table 22 – protecting sensitive data

Issue	Explanation
Confidentiality	Confidentiality is the secure protection of data from unauthorised access, only permitting select users access to said data. This is significant in protecting sensitive data because it ensures that sensitive data is protected from potentially malicious unauthorised access which helps maintain the confidentiality of personal, financial and other sensitive information which could jeopardise the client and company. An example of this is medical data, which have to be protected to prevent ransomware attacks, etc. which hold the company liable and damage their reputation but also impact the client/owner of said data.
Integrity	Integrity is how trustworthy, reliable, accurate and complete the data is to its original state. Any unauthorised modifications to data reduce its integrity. Maintaining the integrity of data is significant in protecting sensitive data as it ensures that the data is reliable and accurate making sure that the use of the data isn't misleading. This is often used maliciously to influence the decisions of the victim of attack.
Availability	Protecting the availability of sensitive data is done by ensuring that data is easily and readily available when needed. This is significant in protecting sensitive data as it allows users to access their data without disruptions, as sometimes access to this data may be critical/urgent. This involves protection against attacks such as denial-of-service (DOS) attacks.
Compliance with regulations	developing secure software helps organisations comply with data protection regulations and standards. This avoids possible legal trouble and liability issues. Some of these

Issue	Explanation
	regulations include the Privacy Act 1988. In dustry specific regulations should also be considered such as health insurance probability and account act (HIPAA) for healthcare related services or payment card industry data security standard (PCI DSS) for payment.
Protection against cyber threats	Secure software reduces the risk of cyber threats such as malware, ransomware, phishing attacks and data breaches. It helps safeguard sensitive information and prevents financial losses and reputational damage.
Customer trust	building software with robust security measures enhances customer trust. Users are more likely to engage with a platform that prioritize their data protection, leading to increased customer loyalty and satisfaction
Business continuity	secure software contributes to overall resilience of a business by reducing the likelihood of data loss or service disruption due to security incidents. this ensures continuity of operation and minimises potential financial and operational impacts

3.1.2 Strategies employed for data encryption and storage.

Data encryption is the process of converting data into a code to prevent unauthorised access, while secure data storage involves ensuring that data stored isn't manipulated or stolen. This is done by:

- Hashing and salting sensitive data like passwords to prevent leaking user passwords and rainbow table attacks
- Have secure backups of data on different servers with different providers to ensure that attacks don't affect all data, meaning reliable backs can be restored
- Using different providers (e.g AWS and Google cloud) ensures that an architecture attack doesn't compromise the data integrity.

3.2 Minimising cyber attacks and vulnerabilities

Minimizing cyber attacks and vulnerabilities is crucial for ensuring the security of systems, networks, and data. By identifying and addressing vulnerabilities proactively, adopting security best practices, and fostering a culture of security awareness, organizations can reduce the risk of cyber attacks, limit potential damage. This ensures the integrity, confidentiality, and availability of their systems and data.

3.2.1 Benefits of minimising common cyber threats and vulnerabilities.

Table 23 – minimising common cyber threats and vulnerabilities

Issue	Explanation
Prevent data breaches	Secure software development prevents data breaches, and the loss/leaking of sensitive information. This can be information like passwords and credit card information which can be harmful to both the PWA and user if leaked. Preventing data breaches is done by implementing: ● Authorisation ● Sanitising inputs ● QA testing all system before rolling out to ensure no exploits This is beneficial as the Unsecure PWA company will be liable for a lot of breaches if proper measures to prevent breaches aren't taken. This also prevents reputational loss that may prevent users from using/trusting the PWA.
Mitigate against cyber threats	Reduces the likelihood of cyber attacks such as malware infections, ransomware incidents, phishing scams and other malicious activities that disrupt operations and compromise data integrity. This is beneficial as fixing cyber threats

Issue	Explanation
	AFTER they've occurred often incurs more costs and maintenance than implementing mitigation before the deployment of the PWA. Leading to reduced costs.
Protect customer trust	Secure software builds confidence among users and stakeholders by demonstrating a commitment to safeguarding their data and privacy. This enhances trust in the software and the organisation, leading to increased customer loyalty and positive brand reputation.
Comply with regulations	Developing secure software ensures compliance with data protection regulations and industry standards, such as GDPR (general data protection regulation), HIPAA (health insurance portability and accountability act), PCI DSS (payment card industry data security standard). Meeting these requirements helps avoid legal penalties and demonstrates a commitment to data security and privacy best practices, ultimately helping the longevity, reputation and finances of the PWA.
Business continuity	By reducing the risk of downtime, financial, and reputational damage, secure software contributes to business continuity and operational stability.
Cost savings	Addressing security vulnerabilities during software development life cycle is more cost effective than dealing with security incidents and breaches after deployment. Investing in secure software development upfront can help save resources that would otherwise be spent on radiation effects and damage control
Competitive advantage	Developing software sets organisations apart from competitors by demonstrating a commitment to cybersecurity and data protection. This can be a differentiating factor for customers and partners who prioritise security when choosing software solutions. Furthermore, software that is secure typically is of higher quality and has distinct features (in ui, etc.) this is often noticed by the consumer, further increasing trust.

Discussion on preventative measures implemented.

The preventative measures implemented ensure that a range of vulnerabilities are mitigated. These solutions are tailored to the scope and design of the Unsecure PWA and include:

- Session management which ensures that proper authorisation is maintained, preventing malicious to pretend to be another, authorised user.
- Hashing and salting passwords to prevent attackers from directly reading sensitive data in the case that they gain access to the database
- Sanitising input fields to prevent attacks such as SQL and XSS
- Integrating Google Authenticator's two factor authentication to add an extra layer of security which further prevents improper authorisation

These preventative measures follow many standards such as those outlines by OWASP and are effective at protecting the Unsecure PWA

4. Fundamental software development steps to develop secure code

4.1 Requirements definition for client: 'The Unsecure PWA Company'

Table 24 – requirements definition for client

Requirement	Description
Objective	To increase the security of an already made progressive web application (PWA) provided by the client and improve the functionality of the commenting feature. This is done by implementing security features that prevent common vulnerabilities and align with industry standards and regulations surrounding these features. The system must enhance user security by: ● having a secure login page with 2 factor authentication ● store sensitive information securely and encrypted ● ensure users are only able to delete/edit their own comments ● prevent common vulnerabilities such as injection attacks and session hijacking
Tools and environment	The client's requirements for the tools and environments I must use are: Visual Studio Code (VS code) with SQ Lite, Python Flask and Bcrypt, and access to Google 2FA
Functional requirements	The function requirements are to: Create a login page that accepts a username and password and provides access only to authorized users, have 2 factor authentication upon login as an additional layer of security, authenticate users and ensure that only authorized users can access protected features based on their roles, user credentials must be securely stored and protected against unauthorized access, Sanitise inputs to prevent SQL injections, cross site scripting and other attacks, display user comments along with their usernames, allow users to edit and delete comments if authorised, and conduct security testing before and after implementation of security features to identify vulnerabilities

Requirement	Description
	and measure their success with documented results showing reduced vulnerabilities.
Non-functional requirements	The non-functional requirements include: ● improving the overall security of the PWA by implementing industry standard security measures ● ensuring that the security measures do not compromise the user experience or ergonomics of the PWA ● ensuring that there is no significant impact on performance from the implementation of the security measures ● Ensuring that the security measures are scalable, meaning as the PWA grows in scope and users, the security measures are still effective
Testing and validation	Testing and Validation involves ensuring that the requirements are actually met. This will be done through conducting penetration testing (using programs such as OWASP ZAP) to identify the security vulnerabilities which will then be addressed and resolved, then after solutions have been implemented penetration testing will be performed again to ensure the solutions work. However other testing and validation methods will also be implemented (SAST, etc.) to provide an extra layer of validation.
Documentation and presentation	document any security enhancements made and include the details of these security measures and algorithms through things such as code snippets. I am then to prepare a presentation on the security improvements made, including before and after comparisons of the testing and validation results
Timeline	The project will have an established timeline for each phase of the project, including: development, testing, validation, and documentation. There must be milestones also set for conducting pen tests, implementing security measures and finalising the documentation along with the presentation.

Outline the process for gathering security requirements.

The requirements definition in software development is the process of determining the criteria for success defined by the client. The process of gathering security requirements involves heavy collaboration and communication with the client. Requirements should be the what and not the how, meaning requirements can be understood without technical knowledge. This should be done by:

1. analysing the system and understanding the current state of the unsecured PWA to identify security vulnerabilities that need to be addressed
2. surveying and communicating with clients, end users, regulatory bodies, etc. to identify the priorities of the project (e.g privacy for users, compliance for legal).
3. define the specific security requirements, such as a secure login page with 2 factor authentication, etc. which can be done through interviews with the client, reviewing industry standards (such as OWASP Top 10).
4. Review the requirements with stakeholders to ensure they are understood and are what they want.

Each requirement shouldn’t be vague (especially functional requirements) so acronyms like SMART can be used to ensure that each requirement is: Specific, Measurable, Achievable, Relevant, and Timed. Following this process helps ensure the final product is both secure and aligned with stakeholder expectations.

4.2 Determining specifications

Table 25 – determining specifications

Specification	Description
Login page development	I am to develop a login page for the Insecure PWA that Includes a Username and Password field. ● Inputs will be sanitised and validated to prevent injection attacks. ● Secure session management will be established after successful login. ● Login forms will be protected from Cross-Site Request Forgery using CSRF tokens. ● Two-Factor Authentication will be triggered after successful credential validation Access the the PWA should not be allowed until all the criteria is met. If the credentials are not valid, an appropriate error message should be displayed that does not specify if it is the username or password that is incorrect.
Password hashing	User credentials will be stored securely by hashing and salting the passwords. This will be done by using Bcrypt. ● a cost factor of 12 will be used as it is a good balance between security and performance ● Each password will be automatically salted by Bcrypt.

Specification	Description
	The salt is a random value added to the password before hashing to ensure that identical passwords result in different hash outputs, this prevents rainbow table attacks. ● All hashing will be performed server side so that the hashing algorithm used is unknown ● During authorisation when logging in, the server will hash the inputted password and compare the hashed result with the user's hashed password stored in the database. This ensures brute force attacks are not effective and that in the case of malicious access to the database, passwords are not reversible.
Input field sanitisation	validate and sanitise inputs to remove potentially malicious or harmful data entered into all input fields of the PWA (comments, username, password, etc.). This will be done by: ● Using the python html.escape() function to convert special characters (like <,>,”,etc.) to HTML-safe equivalents ● Using input validation where each input field will be validated against the expected formats (e.g email). This will be done server and client sided ● Common malicious characters will be not allowed to add additional protection ● parameterised queries will be used when executing SQL code to ensure any inputs are not interpreted as code, preventing SQL injection attacks This prevents a range of vulnerabilities such as Cross Site Scripting and SQL injection.
Two-factor authentication (2FA)	implement 2 factor authentication onto the login system to add an extra layer to its security. This will be done by: ● using a TOTP code generated by google authenticator to verify user logins. This is accessed using a QR code to ensure a smooth user experience ● Once the user inputs the code the server will verify

Specification	Description
	that it is correct. ● Users will be limited to 5 TOTP attempts within a 10 minute window. This rate limiting further prevents brute force attacks This implementation follows OWASP Authentication Best Practices and ensures a secure login system.
Penetration testing	Penetration testing will be conducted to identify and measure the effectiveness of vulnerabilities and their implemented solutions in the PWA. This is done by: ● Conducting an initial penetration test using OWASP ZAP which is a Dynamic Application Security Test meaning it doesn't have access to the code. ● The vulnerabilities will then be properly documented and addressed ● Another penetration test using OWASP ZAP will then be performed to confirm that previously identified vulnerabilities have been effectively mitigated and that no new vulnerabilities have been introduced. All testing will follow OWASP guidelines, this process ensures that any vulnerabilities are appropriately identified and handled.
Documentation and presentation	● Document all security measures implemented into the Unsecure PWA (include all measures and algorithms). This documentation will follow the template provided and follow the standard of the example reports provided. ● prepare a presentation using powerpoint on the before-and-after security improvements, pen testing results, and the reasoning behind the security measures by following standards and other templates of similar reports and documentation.
Security tools and technologies	The security tools and technologies I will use are: ● Visual studio code as the IDE ● SQLite3 for databases ● Python Flask for server side logic

Specification	Description
	● Bcrypt for the hashing function ● Google 2fa for the TOTP ● OWASP ZAP for dynamic application security testing Which are all up to date and effective technologies.
Timeline and milestones	A clear timeline with measurable milestones for each stage will be created. These stages include: development, testing, validation and documentation. deadlines are to be set for implementing security measures, conducting pen testing, and finalising documentation and presentation.
Scalability and performance	All security enhancements must not have any significant impacts on the performance of the PWA. This can be measured through various tools such as the performance tab on the inspect element of the Chrome Dev Tool. Furthermore, these security features must be scalable with an increase of users and growth of the PWA.
Compliance and best practices	All security measures must align with industry practices and compliance requirements including those related to data protection and privacy (such as the Privacy Act 1988). Furthermore, coding standards and security guidelines must be adhered to in order to maintain the integrity and security of the PWA, one of these is OWASP Top 10.

Discuss the specifications that enhance security.

While all of the specifications enhance the security of the Unsecure PWA in some sense by implementing features that comply with security standards, the specifications that exclusively enhance security are:

Password hashing (and salting) which ensures that sensitive data is encrypted and cannot be reversed and prevents brute force attacks. This step is vital as it is a very common standard of information security and must be implemented. This works by using a one way hashing algorithm. Another security feature that enhances security is input sanitisation. This is very important and common practice in information security. This prevents people exiting the input parameters and being able to run commands on the system. This can allow people to use SQL to do things like delete the entire database, hence why input sanitisation is so important. Another reason input sanitisation is important to implement is because it prevents cross site scripting where malicious users execute arbitrary chunks of JavaScript when other users visit your site. It is the most common publicly reported vulnerability and hence why its so important to prevent. Another specification that enhances the security of the PWA is 2 factor authentication. Microsoft reported that 99.9% of automated account attacks could be blocked by enabling any form of 2FA, meaning that it is a very effective security measure to implement into the PWA. While these are the only specifications that directly enhance security, making them the most critical, the other specifications have elements that indirectly enhance security. Such as documentation and presentation, this allows people to in the future go back to the documentation and go over the surety features implemented. This is very important in ensuring a future proof cyber security of the PWA that the client understands, making it effective in enhancing security of the PWA. Penetration testing also indirectly enhances the security of the PWA as it allows for the identifying of vulnerabilities which is necessary in fixing them, furthermore it allows for the effectiveness of the solutions to the vulnerabilities to be measured, again making it effective in enhancing the security of the PWA. compliances and best practices is effective as it holds the security measures implemented accountable, ensuring they are up to date and adequate.

While several specifications in the development of the PWA contribute to overall security, those that focus on password hashing, input sanitization, and 2FA are the primary security enhancements. These measures address the most common vulnerabilities and provide critical layers of protection. Additionally, documentation, pen testing, and compliance play a vital role in ensuring that security practices are implemented effectively, reviewed regularly, and maintained over time.

4.3 Design

Key design principles for secure architecture.

When designing a secure system there are many key principles to be considered to ensure there is minimal risk of vulnerability inherent to the system. While there isn’t any official list of key design principles, there are key design principles common across many secure software standards (such as OWASP ASVS) that should be present in every secure PWA, these are:

- **Defense in Depth**, which instructs that secure architecture should not rely on any one security mechanism to keep the system safe and have a range of measures in place to ensure security. An example of this is 2 factor authentication as it's an additional layer of security to the login system.
- **Least Privilege**, which instructs that the minimum level of authorisation/access rights should be given, and for only as long as it's needed. Furthermore it also says to remove any features that aren’t needed (e.g FTP and SSH should be removed if they aren't needed)
- **Separation of Duties**, which instructs that there should be no single point of control, meaning that multiple bad actors will be needed to compromise the system.
- **Secure by Design**, which instructs that Secure software must be secure by design and not be implemented as an afterthought. This means security shouldn’t be considered after the installation of the software but rather during every stage of the development cycle (requirements, specifications, etc.).

- **Economy of Mechanism**, which instructs that while the system should be designed to be secure it should also be simple, meaning that security measures aren't complex or difficult to perform for users. This prevents them circumventing the systems in place (e.g if the login system requires a new really long password to be made every month then users will end up doing things like ComplexPassword#1, then ComplexPassword#2, … because the system is too complex)

These key design principles for software architecture are the strategies used to create a secure system. These are critical to secure software and hence must be always considered.

4.4 Development

Best practices for writing secure code.

Table 26 – design and development phases

Phases	Description
Requirement analysis	An understanding of the Unsecure PWA must be established and the security vulnerabilities that are to be addressed. I must then define specific security requirements such as implementing password hashing, input sanitisation, two-factor authentication, and pen testing.
Design phase	Design the PWA with security features in mind. Include the login page, password hashing mechanism, input sanitisation process, and 2FA integration. Properly plan the features and their implementation by creating wireframes and sketches to visualise the layout and user interface elements of the secure PWA
Development phase	The development phase is where the planned features and requirements are made. Develop the required features, this is done by: ● Developing a login page that has the required features set out by the client, including: a login page with secure authentication which is done using python flask and SQL lite ● ensuring that passwords are stored securely in the database by hashing them with Bcrypt ● Ensure that all input fields are properly sanitised to prevent SQL injection ans XSS attacks ● Add Google Authenticator for two factor authentication to the login process to ensure the system has defence in depth

Phases	Description
Testing phase	The testing phase is essential in developing secure code to identify and fix vulnerabilities and errors and weaknesses in the software before deployment, ensuring a higher level of security, this will be done by: ● Conducting an initial pen testing to identify vulnerabilities in the unsecured PWA and address critical security issues ● Testing the implementation of password hashing, input sanitisation, and 2FA for functionality and security compliance ● Performing user acceptance testing to ensure a seamless user experience while maintaining robust security measures
Security assessment and validation	This stage is essential in writing secure code as it ensures/verifies that the solutions implemented actually resolve any vulnerabilities and to what extent, this is done by: ● Validating the effectiveness of the security enhancements through post-implementation pen testing and security assessments ● Addressing any vulnerabilities discovered during testing and ensure that all security measures are functioning as intended
Documentation and presentation	This stage is where everything done is documented to ensure all processes are held accountable and easy to understand for the client. This is done by: ● Documenting the security design and development process, including detailed specifications, implementation steps, and security measures ● Presenting the security enhancements, pen testing results, and the rationale behind the chosen security features
Deployment and monitoring	This stage is where the finalised solution/program is released. This software should be adequate secure by this point, although it should still be monitored to check its running properly and is not vulnerable. This stage is done by: ● Deploying the secure PWA to a web server or hosting environment, ensuring that all security measures are

Phases	Description
	operational ● Implementing monitoring tools to track security metrics, detect potential security incidents, and ensure continuous security of the PWA

4.5 Integration

Security considerations during integration.

Integration is the testing of the program as a whole, with all modules together. This allows for the testing of the interaction between modules allowing for the assessment of the system as a whole.

Table 27 – integration

Steps	Description
Assessment of existing system	During integration, the system being integrated into should be heavily considered. Any vulnerabilities in the system before integration should be properly identified and measured. This is done by: ● Identifying the current vulnerabilities in the PWA ● Reviewing the current security measures in place and measuring their accuracy ● Evaluating the system architecture to understand potential security risks ● Running tools like OWASP ZAP to help identify vulnerabilities in the system through penetration testing ● Evaluating how improvements may be integrated into the existing system This ensures a proper understanding of the system the solution is being integrated into
Requirements alignment	Ensure that the solutions implemented, when integrated, align with the requirements set out by the client. This can be done by: ● Surveying/presenting the integrated system to the client ● Ensuring the system meets regulatory requirements (e.g Privacy Act 1988) WHEN integrated ● Check that each requirement set out in the requirements

Steps	Description
	definition is covered WHEN integrated ● Check that the integrated system prevents the vulnerabilities identified as requirements using pen tests, etc. During integration, while all requirements should aim to be implemented, the most critical should be prioritised as they are most vital to the security of the PWA
Modular development approach	All functions should be designed as modular, meaning each function of the code is separated and can be developed, tested, and maintained separately. This makes it very reusable and easily replaceable. Each module should be integrated after verifying that: ● it is independently secure ● Performs the task it is supposed to perform to the required extent ● each module can be tested independently ● each module follows the security practices and standards independently ● each function is documented Then only after this is verified they should be integrated together, then their capacity to maintain these same security standards should be evaluated using tools such as OWASP ZAP
API integration	During the integration stage seamlessly connect APIs to enhance system functionality and enable efficient data exchange. When integrating APIs into the Unsecure PWA, ensure that they are integrated securely, this is done by: ● Following RESTful design principles for consistency, scalability, and ease of use ● Encrypting API requests and responses using SSL/TLS via HTTPS to protect data in transit ● Validating and sanitizing all input and output data to protect against injection attacks ● Using whitelists to specify which parameters are allowed in requests ● Using authentication like OAuth 2.0 and JWT to ensure only authorised access ● Validating and sanitize all input and output data to prevent

Steps	Description
	injection attacks ● Monitoring and logging API activity to check for attacks ● setting up alerts for suspicious activity
Database integration	Integrating the database is essentially connecting it to the rest of the PWA. This means making sure the frontend/backend can access and modify data when needed, secure database integration involves: ● Securely connect the database to the backend, avoiding exposure of database paths or files ● Controlling access to the database file, ensuring only the server has access to it ● Using parameterized queries in the SQL code to prevent SQL injection ● opening, using, and closing connections safely to avoid data corruption ● Storing the SQLite file in a secure location that isn't accessible from the browser or frontend ● Setting up database logging to ensure activity can be monitored for malicious activity These processes and measures ensure that the connection of the database to the rest of the PWA is done securely and avoids the risk of data loss of vulnerabilities
User interface integration	Secure user interface (ui) integration involves connecting the user interface to the backend systems, APIs, databases, etc. in a way that prevents malicious attacks or manipulations of the system. This is done by: ● Ensure Secure Session Management ● perform UI Testing to ensure the API interacts with PWA properly ● UI should not directly integrate backend logic ● displaying correct and descriptive errors ● Hide sensitive data such as passwords with asterixis (*) ● Ensure that the UI does not handle backend logic like authentication ● integrate client side validation for user convenience, and leave server side validation for security

Steps	Description
Testing and validation	Integrating texting and validation involves testing individual modules at a unit and whole system level. This is done by using tools such as OWASP ZAP to first: ● initially test each individual module and ensure they are secure ● Integrate each module using a secure integration plan ● use the same tools to test the security of the PWA and compare it to the initial results These tests should measure each individual module integrated (e.g fuzz testing for API)
Deployment and monitoring	integrating deployment and monitoring securely into the Unsecure PWA involves ensuring all modules and requirements are deployed to the PWA which is securely hosted on a web server, the PWA should then have monitoring tools integrated, these are tools such as logging tools, OWASP ZAP, user metrics, etc. This ensures that each individual module is deployed and monitored securely as a complete program.

4.6 Testing and debugging

Methods for ensuring security through testing.

Testing and debugging allows us to find security flaws and vulnerabilities before we release our code to the public, thus reducing the risks. Testing and debugging helps prevent unwanted or unexpected operations in our final code thus improving security. Methods to ensure security through testing include:

- Static application security testing (SAST), also known as white box tools, is a method where the source code or binaries are inspected for vulnerabilities. SAST is good because it finds vulnerabilities earlier which makes the issues cheaper to fix, analyzes 100% of the code, doesn't need to execute the program, points out the exact code location of vulnerabilities, and can be automated. However, SAST is language dependant meaning the tools used to perform SAST have to be tailored to the programs language (e.g python)
- Dynamic application security testing (DAST), also known as black box testing, involves testing the program while its running without access to the code and points out any vulnerabilities found. DAST is good because it Highlights authentication and server configuration issues, is language independent which was a drawback of SAST making these tools good in tandem, tests the whole system, checks memory consumption and resources meaning it can test the performance of the security measures implemented, tests common attacks such as SQL injection and XSS. DAST does have drawbacks being that it produces a large amount of false positives and does not point to the exact location of the issues.
- Penetration testing involves simulating a cyber attack against your own system to highlight vulnerabilities such as unsanitized inputs that are susceptible to code injection attacks. Pen testing typically utilizes DAST to scan for vulnerabilities, after vulnerabilities are discovered it simulates an actual attack such as SQL injection to

evaluate the extent of the damage it could cause. There is then typically a score given on how vulnerable the system is and a report outlining what those vulnerabilities are.

- Vulnerability assessments are a systematic review of weaknesses in the system. This involves assessing the vulnerability of many different components such as critical servers, authentication systems, databases, etc. It involves four steps being: testing, analysis, assessment and remediation. This assessment is unique in the fact that it involves things like a risk assessment, analysing the severity of the vulnerabilities which allows for priorities to be assigned to them and the potential damage outlined.
- Code review, a method where the code is reviewed from flaws, readability and scalability. This can be often done manually.
- API testing is a method to test the API component of the system, this method prevents unauthorized access, data leaks, injection attacks, and other vulnerabilities by ensuring APIs function securely and correctly under various conditions. There are a range of different API tests including: load testing which ensures that the API isnt vulnerable to DOS attacks, Fuzz testing which tests its ability to handle invalid inputs, etc. while there are many tests for just one component of the system (API), it is a very critical component so much be thoroughly tested.

While these are all valuable methods for ensuring security through testing, the greatest amount of security is through using these methods together as they all have their own unique weaknesses and strengths.

4.7 Installation

Security measures taken during installation.

Installation is the deployment of a system for use by the consumer. The steps involved in the installation of secure software include verifying system requirements, configuring settings for security controls, installing the software and conducting post-installation checks for security. Security measures that should be taken during installation include:

- Changing default credentials to prevent unauthorised access
- Disabling unnecessary services and ports to limit the possibilities for vulnerabilties
- firewalls and intrusion detection systems
- Ensuring all dependencies and architecture are up to date
- ensuring users and services have the minimum permissions needed
- Using secure communication protocols like HTTPS
- Setting up audit logging and monitoring tools to track potentially malicious activity
- Performing post installation security tests with previously mentioned methods such as OWASP ZAP

4.8 Maintenance

Strategies for ongoing security monitoring and updates.

Maintenance is the process of keeping software operational, updated and secure over its lifespan. As a part of this there should be ongoing security monitoring and updates of the program. Strategies to ensure this include:

- Ensuring all software, architecture and dependencies are up to date
- Implement monitoring tools to track security metrics and detect potential security incidents
- Performing network monitoring to send out alerts if any issues regarding the availability or functionality of network services arise
- Regular code reviews and vulnerability assessments to ensure that the program is still secure
- Privileged user monitoring to ensure there aren't any administrators with malicious intent.
- Preventing alert fatigue by only having critical alerts marked as urgent

5. Influence of end users on secure design features

5.1 Analysis of how user capabilities and experiences affect security features.

Table 28 – user capability and experience

Issue	Explanation
User-friendly authentication	Authentication should be user-friendly as this follows the economy of mechanism key design principle. Authentication should be user friendly because if it is too complex and difficult then users will find ways to circumvent the security measures, reducing the effectiveness of security features implemented. An example of this is if the login system requires a password that requires at least 8 characters, a symbol, and number that must be changed every month then users will end up doing things like ComplexPassword#1, then ComplexPassword#2 the next month, etc… because the system is too complex. This reduces the security as if an attacker gets access to the past passwords of authenticated users, then they could just use the amount of months it's been since that password to get their current password. Additionally user capabilities must be considered such as color blindness, meaning the 2fa shouldnt have a color based captcha. mAs a result, authentication security features are designed to be user friendly, ensuring that users don't circumvent the features because they are too complex or confusing for them.
Role-based access control	Role-based access controls should match the user's capabilities and experience. Role-based access control is a system where sets of permissions are grouped into roles, and users are assigned to roles, instead of assigning individual permissions directly to users. Role-based access control allows for easier management of access control, as permissions are managed per role rather than per user. This makes it less common for inappropriate access to be given accidentally. Proper assignment of roles is important because it follows the key design principle of least privilege, ensuring that users only have the minimum permissions they need. This strengthens security

Issue	Explanation
	by reducing the risk of accidental misuse, preventing insider threats, and limiting the potential damage if a user's account is compromised. Therefore, user capabilities and experiences directly affect the security of role-based access control systems, as roles must be carefully matched to a user's actual skills and responsibilities to maintain effective security
Training and awareness	The training and awareness of users directly affects security features effectiveness. Users need proper training on how to use security features to ensure they are not misusing security tools and features which could compromise the security of the system. User awareness (which is often trained) helps users recognise attacks, strengthening the security of the system. A key awareness feature is surrounding social engineering as these attacks focus on tricking/manipulating the users. An example of a common social engineering attack is dropping usbs contaminated with malware near the office of the target, which then due to a lack of awareness, users may plug them in to compromising devices out of curiosity. As a result, proper training and awareness of users is vital in ensuring the effectiveness of security features and preventing socially engineered attacks
User feedback and testing	User feedback strengthens security features by ensuring that users aren’t frustrated or confused by security measures, preventing them from circumventing them and allowing developers to design more user-friendly security features. User testing ensures that users follow security features in an effective way by testing the capabilities of the users, through exercises such as simulated phishing attacks. This affects security features by making sure they align with the users capabilities and experience/knowledge of systems
Simplified security controls	Security controls are anything that minimise security risks and include things such as physical controls like a lock to the server room and virtual things such as a firewall. Simplifying security controls ensures that users are capable of following and properly understanding the security measures in place, reducing

Issue	Explanation
	mistakes and the risk of users circumventing protections. Furthermore, simple security controls are easier to update and change meaning developers can quickly adhere to user feedback and testing which further strengthens the security of the system

5.2 Importance of user feedback in the design process. ABOUT NOT JUST SECURITY

User feedback is very important especially in the design process as incorporating user feedback before the development phase means that features that consider user feedback are not an afterthought, reducing costs and the quality of features implemented based on user feedback. Furthermore, user feedback in the design process is important because it:

- Improves the user experience as it allows them to give feedback on usability, performance issues and feature requests which helps developers enhance the software to meet user expectations, improving
- Ensures inclusion of disabled users as users with disabilities can give feedback on non-inclusive features, for example a user with color blindness may suggest a color accessibility option, improving the software by increasing the range of users who can use it
- .End users can also provide a unique perspective on security features that should be tweaked or implemented, especially if a security feature is too complex and may be circumvented by the users, ensuring an economy of mechanism and therefore improving the security of the PWA before the development of it.
- End users provide feedback on ui feel, ux, etc. which overall contributes to customer trust. If the PWA is designed to create customer trust through measures like a clean and industry standard ui then users are more likely to use the PWA, increasing the revenue of the PWA. This is especially important during the design phase as going back and making these improvements after development would be costly.

Overall, user feedback during the design process is important as it not only improves the product and its security but also saves costs from having to implement curtain features after development.

6. Developing secure code

6.1 Fundamental Software Design Security Concepts

Explanation of key concepts:

Table 29 – fundamental software design security concepts

Concept	Explanation
Confidentiality	Secure Software ensures that sensitive data is protected from any unauthorised access. This maintains the confidentiality of potentially sensitive information such as medical, financial and personal data.
Integrity	Secure software maintains the integrity of data by maintaining the accuracy and reliability of data. This is done by preventing any unauthorised modifications to data through proper security measures.
Availability	Availability in secure software design refers to data that is always accessible to its owner. Secure software helps maintain the availability of data by preventing attacks such as Distributed Denial of Service (DDOS) attacks through proper security measures.
Authentication	Authentication in secure software design is the process of verifying the identity of a user of a system. This ensures that malicious users are not able to impersonate other users, ensuring access to data is restricted to the correct users.
Authorisation	Authorisation in secure software design is the process of determining the actions a user is permitted to perform. This controls what resources, data, and actions the user can access/perform based on their identity. This prevents malicious users performing actions they aren't intended to do.
Accountability	Accountability in secure software design in the process of holding individuals responsible for their actions and decisions within a system. This deters malicious activity and allows for the investigation after attacks. This is done

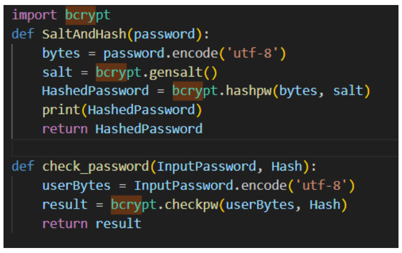
Concept	Explanation
	through features such as logging, ensuring actions can be traced back to individuals.

7. Security features incorporated into software

7.1 Overview of security features implemented.

Table 30 – security features

Security feature	Application

Data protection	● User passwords were securely stored by hashing and salting them with Bcrypt to protect user passwords stored in the database.
	○ 
	`import bcrypt`
	`def SaltAndHash(password):`
	`    bytes = password.encode('utf-8')`
	`    salt = bcrypt.gensalt()`
	`    HashedPassword = bcrypt.hashpw(bytes, salt)`
	`    print(HashedPassword)`
	`    return HashedPassword`
	
	`def check_password(InputPassword, Hash):`
	`    userBytes = InputPassword.encode('utf-8')`
	`    result = bcrypt.checkpw(userBytes, Hash)`
	`    return result`
	this is a snippet of what hashes and checks the hashed passwords
	● Data minimisation was ensured by collecting only necessary user information and implementing data retention policies to limit data storage and potential sensitive information that can be stolen
	● Sensitive data was encrypted both in transit (while it travels across the network) and at rest (while stored) using encryption protocols like TLS to prevent unauthorized access, interception, or tampering of data

Security measures	● Input validation and sanitisation techniques were implemented to prevent common vulnerabilities like SQL injections and cross-site scripting. ● parameterised queries and prepared statements were implemented to protect against SQL injection attacks on the database. ● Secure coding practices were used to mitigate security risks, such as input validation, output encoding, and error handling. ● Use secure apis by following REST-ful design principles and securely transferring and logging requests/data
Privacy protection	● Develop a privacy policy that clearly outlines how user data is collected, used and protected within the PWA.

Security feature	Application
	• Obtain user consent for data processing activities and provide transparency on how user data is handled. • Implement privacy-enhancing technologies (PETs) like anonymisation and pseudonymisation to protect user privacy.
Regulatory compliance	• Align security measures with regulatory requirements such as the Privacy Act 1988 in Australia to ensure compliance with data protection laws. • Implement mechanisms to facilitate user data rights, such as the right to access, rectify and delete personal data. • Conduct regular security audits and assessments to ensure compliance with industry standards and regulations.
User authentication and authorisation	• Implement strong user authentication mechanisms, such as multi-factor authentication (2FA) with Google Authenticator, to enhance login security. • Enforce access controls and authorisation mechanisms to restrict user privileges based on roles and permissions. • Monitor and log user access and activities to maintain accountability and traceability. • Maintain users authentication using secure session management

8. Security by design approach

8.1 Contribution of cryptography and sandboxing

Explain the role of cryptography in data security to the client, ‘The Unsecure PWA company’.

Cryptography is the practice of using codes and ciphers to protect sensitive information. It plays a crucial role in data security for The Unsecure PWA Company by ensuring that even if unauthorized access occurs, the data remains secure and unreadable. By encrypting sensitive data, cryptography protects against data breaches and ensures that confidential information remains private.

For The Unsecure PWA Company, cryptography ensures data confidentiality, integrity, and authenticity.

- Confidentiality is maintained by ensuring that only authorized parties can access the data.
- Integrity ensures that data hasn’t been altered or tampered with during transmission.
- Authenticity ensures that the data and the users are genuine and trustworthy.

Additionally, implementing end-to-end encryption is crucial for securing communication between users and your application, preventing eavesdropping and tampering by third parties. This encryption ensures that messages or data exchanged cannot be intercepted or modified while in transit, securing the entire communication process.

Explain how sandboxing limits exposure to vulnerabilities.

Sandboxing involves isolating applications or processes from the rest of the system, effectively limiting the impact of any security breaches or vulnerabilities. By providing a controlled environment for running untrusted code or applications, sandboxing helps mitigate the risk of malicious software accessing sensitive data or compromising system integrity. This technique reduces exposure and ensures that even if an application is compromised, the damage is contained within the sandboxed environment, helping prevent malicious actions from spreading to other parts of the software or system.

Summarise security by design for the client.

Security by design is a key design principle for secure software. When systems are made using Security by Design, they are designed to be inherently secure, this means that security controls are implemented as a core part of the design process rather than being implemented as an afterthought. This improves the security of the system as security features are intentionally designed with the product, also reducing costs as security controls do not have to be implemented after installation.

8.2 Privacy by design approach

Provide the client with the principles of proactive privacy measures.

Privacy measures should be considered from all stages of development, this means that the software for the Insecure PWA Company should incorporate a privacy by design approach that has proactive privacy measures. There are 7 key principles of privacy by design being:

1. Proactive not reactive, preventative not remedial, meaning that potential risks should be anticipated and privacy-invasive events should be prevented before they occur
2. Privacy as a default setting, meaning that the privacy of personal information should be protected by default
3. Privacy embedded into design, meaning that privacy should be a core function of the system
4. positive-sum not zero-sum, meaning that privacy and security should be a trade off.
5. full lifecycle protection, means that information should be protected throughout its use. This means process personal information securely and then destroy it securely when its no longer needed
6. Visibility and transparency, means that people would be fully aware of the personal information being collected, and for what purpose
7. keep it user centric, means that the interests of users should be the top priority for the privacy of their data. This is done by having strong privacy defaults.

Explain the importance of embedding privacy into software design to the client.

Embedding privacy into software design is important for the Unsecure PWA Company because it ensures that they meet community expectations around how public agencies handle personal information. Embedding privacy into the design process ensures that the system is designed to appropriately handle sensitive and personal information without violating the trust of the consumer, increasing business reputation and ensuring that regulations aren’t breached. Furthermore, this also ensures that privacy is not an afterthought which saves costs from having to implement privacy measures after development. It also allows for more seamless integration of privacy practices, ensuring that security measures work effectively without disrupting the user experience or the overall functionality of the system.

9. Testing and evaluating security and resilience

9.1 Provide the client, 'The Unsecure PWA Company', with methods for identifying vulnerabilities and creating resilience.

Table 31 – methods for identifying vulnerabilities and creating resilience

Method	Description
Hardening systems	System hardening reduces the attack surface and potential vulnerabilities by ensuring the system has prevented unnecessary risks. This involves implementing secure software principles, timely patches/updates, proper security configurations, and closing unused ports. For the Unsecure PWA this minimises potential threats that could impact the continuity and data security of their product.
Handling breaches	Handling breaches is responding to breaches in the security system in a controlled and planned way that is effective and reducing damages and recovering from cyber incidents. This can be done through preparing an incident report plan, detecting and containing breaches quickly, and communicating clearly the issues with stakeholders. Systems should also be continuously monitored to allow for accountability and investigation. Handling breaches will maintains consumer trust and reduces damages to the Unsecure PWA Company if a breach were to occur.
Disaster recovery plan	A disaster recovery plan outlines the steps that should be take in the case of a disaster, which is things like AWS going down, etc. These things disrupt the continuity of the Unsecure PWA and can impact profits and reputation as the availability of data is reduced. To minimise the damages of a disaster a disaster recovery plan should be made that includes: ● recovery objectives ● backup services (like Google cloud) ● risk assessment ● business impact analysis This protects the Unsecure PWA Company by ensuring

Method	Description
	business continuity, reduces data loss, and strengthening trust in the company's ability to manage cyber threats.

10. Strategies for managing security of programming code

10.1 Code review

Explain to the client the process for conducting thorough code reviews.

A code review is the process of evaluating the code's quality, security and compliance with coding standards. This can be measured through things like: checking that the naming conventions are consistent with the rest of the code (Snake case, camel case, etc.), the code is readable with logical comments, and the code follows the security standards and regulations (e.g encrypts sensitive data), and so on.

10.2 Testing methods

Explain SAST and DAST and the importance of vulnerability assessments and penetration testing to the client.

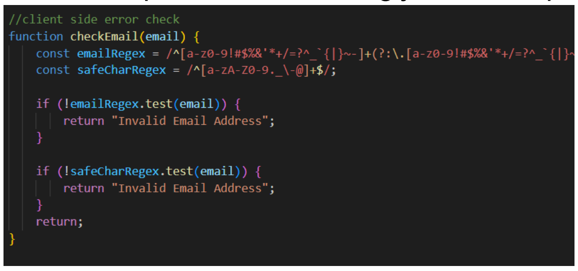
Table 32 – testing methods

Strategy	Explanation
Code review	A code review is the process of evaluating the code's quality, security and compliance with coding standards. This is important to the Unsecure PWA Company to ensure the maintainability of their code, reducing the costs of future improvements.
Static application security testing (SAST)	Static application security testing (SAST), also known as white box tools, is a method where the source code or binaries are inspected for vulnerabilities. It is important for The Unsecure PWA Company to use SAST because it finds vulnerabilities earlier which makes the issues cheaper to fix, analyzes 100% of the code, doesn't need to execute the program, points out the exact code location of vulnerabilities, and can be automated. However, SAST is language dependant meaning the tools used to perform SAST have to be tailored to the programs language (e.g python)
Dynamic application	Dynamic application security testing (DAST), also known as black box testing, involves testing the program while its running without access to the code and points out any vulnerabilities found. It is important that the Unsecure PWA Company use DAST because it highlights authentication

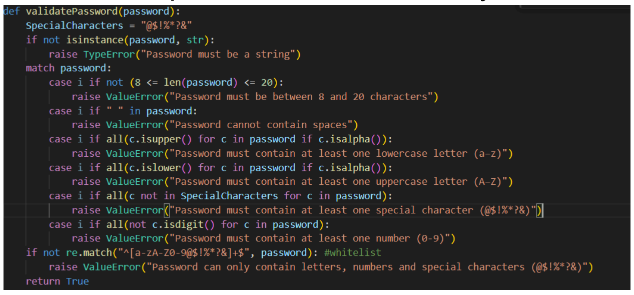
Strategy	Explanation
security testing (DAST)	and server configuration issues, is language independent which was a drawback of SAST making these tools good in tandem, tests the whole system, checks memory consumption and resources meaning it can test the performance of the security measures implemented, tests common attacks such as SQL injection and XSS. DAST does have drawbacks being that it produces a large amount of false positives and does not point to the exact location of the issues, so it is still important for the Unsecure PWA Company to use more than just DAST.
Vulnerability assessment	Vulnerability assessments are a systematic review of weaknesses in the system. This involves assessing the vulnerability of many different components such as critical servers, authentication systems, databases, etc. It involves four steps being: testing, analysis, assessment and remediation. This is important for the Unsecure PWA Company as it allows for preemptive measures to be put in place and is a very organised and systematic evaluation of the system. Furthermore, it involves things like a risk assessment, analysing the severity of the vulnerabilities which allows for priorities to be assigned to them and the potential damage outlined.
Penetration testing	Penetration testing involves simulating a cyber attack against your own system to highlight vulnerabilities such as unsanitized inputs that are susceptible to code injection attacks. This is important to The Unsecure PWA Company because on top of finding vulnerabilities, it simulates an actual attack such as SQL injection to evaluate the extent of the damage it could cause. There is then typically a score given on how vulnerable the system is and a report outlining what those vulnerabilities are which provides valuable, measurable metrics on the vulnerability of the systems.

11. Defensive data input handling

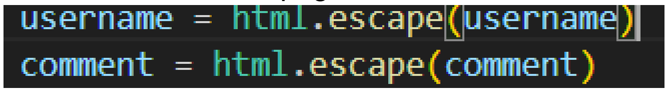
Provide the client with code snippets that demonstrate best practices for input validation, sanitisation, and error handling.

- Initial client-side input validation using js to have quicker responses
    - 

```
//client side error check
function checkEmail(email) {
    const emailRegex = /^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}
    const safeCharRegex = /^[a-zA-Z0-9._\-@]+$/;

    if (!emailRegex.test(email)) {
        return "Invalid Email Address";
    }

    if (!safeCharRegex.test(email)) {
        return "Invalid Email Address";
    }
    return;
}
```

- Then server sided input validation for security from malicious activirty
    - 

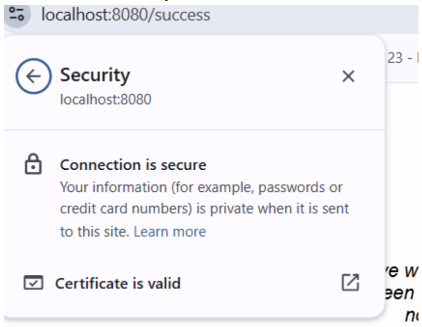
```
def validatePassword(password):
    SpecialCharacters = "@$!%*?&"
    if not isinstance(password, str):
        raise TypeError("Password must be a string")
    match password:
        case i if not (8 <= len(password) <= 20):
            raise ValueError("Password must be between 8 and 20 characters")
        case i if " " in password:
            raise ValueError("Password cannot contain spaces")
        case i if all(c.isupper() for c in password if c.isalpha()):
            raise ValueError("Password must contain at least one lowercase letter (a-z)")
        case i if all(c.islower() for c in password if c.isalpha()):
            raise ValueError("Password must contain at least one uppercase letter (A-Z)")
        case i if all(c not in SpecialCharacters for c in password):
            raise ValueError("Password must contain at least one special character (@$!%*?&)")
        case i if all(not c.isdigit() for c in password):
            raise ValueError("Password must contain at least one number (0-9)")
    if not re.match("^[a-zA-Z0-9@$!%*?&]+$", password): #whitelist
        raise ValueError("Password can only contain letters, numbers and special characters (@$!%*?&)")
    return True
```

- using html.escape() for any user input shown on website like the username and comment when on comment page
    - 

```
username = html.escape(username)
comment = html.escape(comment)
```

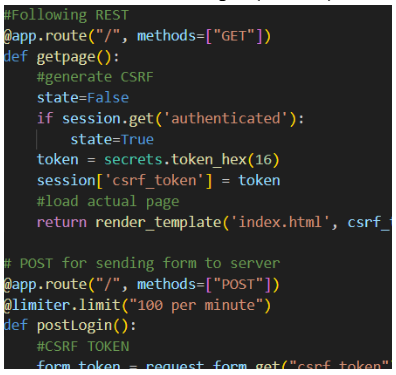
12. Safe API development

Provide the client with a brief explanation of how you created secure APIs to reduce vulnerabilities.

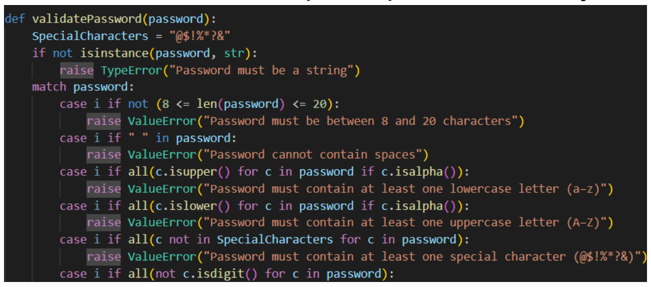
- For safe API development i got a self signed SSL certificate that was encrypted with aes (an advanced encryption algorithm) to encrypt any api traffic going from client to server, this prevented sensitive data from being intercepted
    - 
      localhost:8080/success

      Security
      localhost:8080

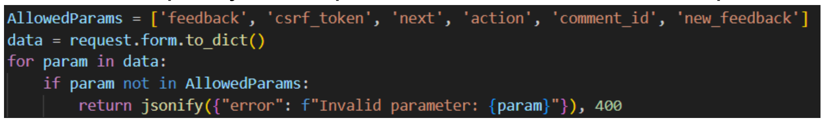
      Connection is secure
      Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

      Certificate is valid
- I followed REST-ful design principles
    - 

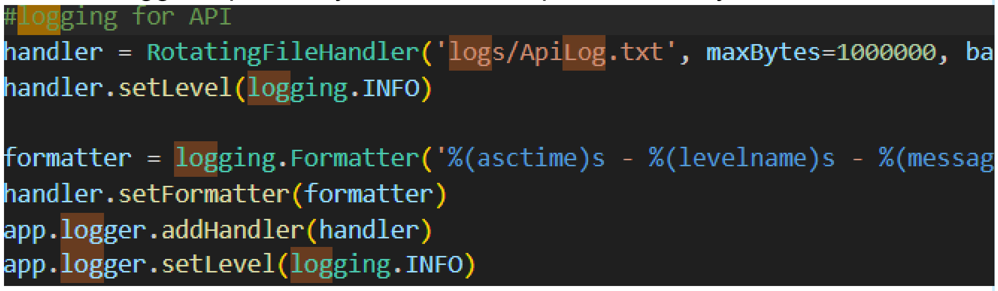
```
#Following REST
@app.route("/", methods=["GET"])
def getpage():
    #generate CSRF
    state=False
    if session.get('authenticated'):
        state=True
    token = secrets.token_hex(16)
    session['csrf_token'] = token
    #load actual page
    return render_template('index.html', csrf_

# POST for sending form to server
@app.route("/", methods=["POST"])
@limiter.limit("100 per minute")
def postLogin():
    #CSRF TOKEN
    form_token = request.form.get("csrf_token"
```

- I validated and sanitised all output to prevent SQL injection and XSS attacks
    - 

```
def validatePassword(password):
    SpecialCharacters = "@$!%*?&"
    if not isinstance(password, str):
        raise TypeError("Password must be a string")
    match password:
        case i if not (8 <= len(password) <= 20):
            raise ValueError("Password must be between 8 and 20 characters")
        case i if " " in password:
            raise ValueError("Password cannot contain spaces")
        case i if all(c.isupper() for c in password if c.isalpha()):
            raise ValueError("Password must contain at least one lowercase letter (a-z)")
        case i if all(c.islower() for c in password if c.isalpha()):
            raise ValueError("Password must contain at least one uppercase letter (A-Z)")
        case i if all(c not in SpecialCharacters for c in password):
            raise ValueError("Password must contain at least one special character (@$!%*?&)")
        case i if all(not c.isdigit() for c in password):
```

- I used whitelists to specify which parameters are allowed in requests
    - 

```
AllowedParams = ['feedback', 'csrf_token', 'next', 'action', 'comment_id', 'new_feedback']
data = request.form.to_dict()
for param in data:
    if param not in AllowedParams:
        return jsonify({"error": f"Invalid parameter: {param}"}), 400
```

- I monitored and logged api activity to monitor suspicious activity
    - 

```
#logging for API
handler = RotatingFileHandler('logs/ApiLog.txt', maxBytes=1000000, ba
handler.setLevel(logging.INFO)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(messag
handler.setFormatter(formatter)
app.logger.addHandler(handler)
app.logger.setLevel(logging.INFO)
```

13. Efficient execution for the user

Inform the client and users of the strategies for effective memory, session, and exception management.

Table 33 – strategies for effective memory, session, and exception management

Strategy	Explanation
Memory management	Proper memory management prevents information leaking to attackers which could give them information on possible vulnerabilities. Strategies to ensure secure memory management include: ● Turning off HTTP headers ● Avoid paths ending in file extensions ● Sanitise error messages ● sanitise source code that leaves information like server names ● scrub potentially sensitive meta data
Session management	A session is an interaction between the user and the website. It maintains the user data throughout the session on the site, adapting to their personal preferences and maintaining the logged in status for the user. If an attacker gets access to a session ID they can impersonate a user. This can be prevented by: ● Not Passing Session IDs in GET/POST Variables ● Regenerating the Session ID at Authentication ● Accept Only Server-Generated Session IDs ● Timeout and Replace Old Session IDs ● Require a New Session When Visiting From Suspicious Referrers
Exception management	Exception management is handling and addressing exceptions or errors that occur during software execution, preventing crashes, provide guidance to what went wrong and ensuring the continuity of the software. This can be done by:

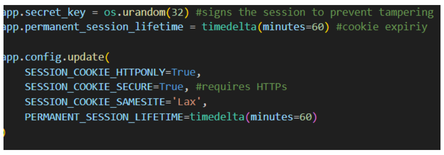
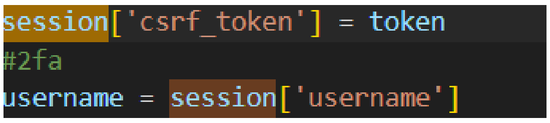
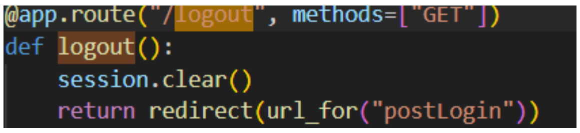
Strategy	Explanation
	● catching errors ● setting informative error messages and codes

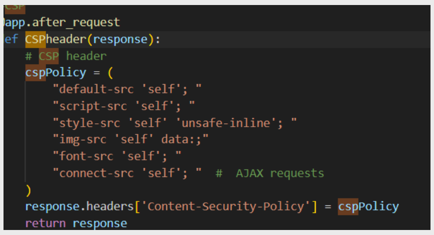
14. Secure code for user action controls

Provide the client and users with snippets of code to explain how you have protected against.

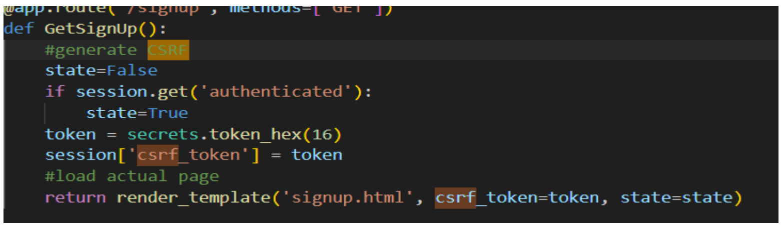
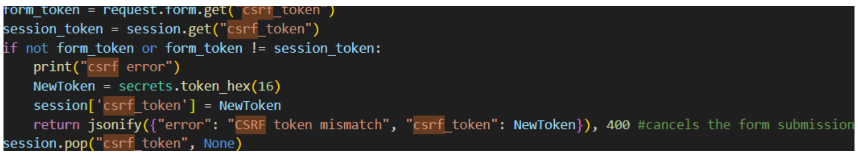
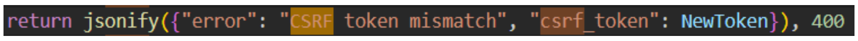
Table 34 – secure code samples

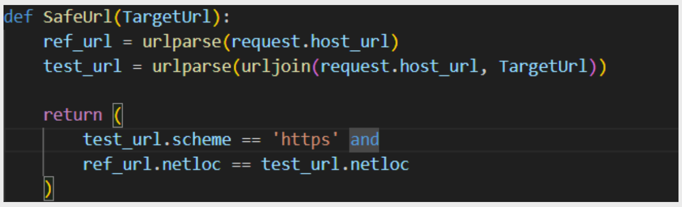
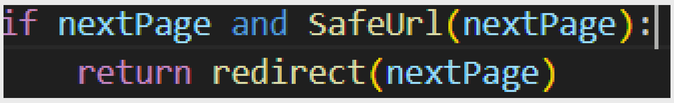
Vulnerability	Code sample

Broken authentication	● I protected against broken authentication by usng propper session management configured so it
	○ Not Passing Session IDs in GET/POST Variables
	○ Regenerating the Session ID at Authentication
	○ Accept Only Server-Generated Session IDs
	○ Timeout and Replace Old Session IDs
	○ Require a New Session When Visiting From Suspicious Referrers
	● 
	`app.secret_key = os.urandom(32) #signs the session to prevent tampering`
	`app.permanent_session_lifetime = timedelta(minutes=60) #cookie expiriy`
	
	`app.config.update(`
	`    SESSION_COOKIE_HTTPONLY=True,`
	`    SESSION_COOKIE_SECURE=True, #requires HTTPs`
	`    SESSION_COOKIE_SAMESITE='Lax',`
	`    PERMANENT_SESSION_LIFETIME=timedelta(minutes=60)`
	`)`
	● 
	`session['csrf_token'] = token`
	`#2fa`
	`username = session['username']`
	● I would also invalidate these sessions on log out
	○ 
	`@app.route("/logout", methods=["GET"])`
	`def logout():`
	`    session.clear()`
	`    return redirect(url_for("postLogin"))`

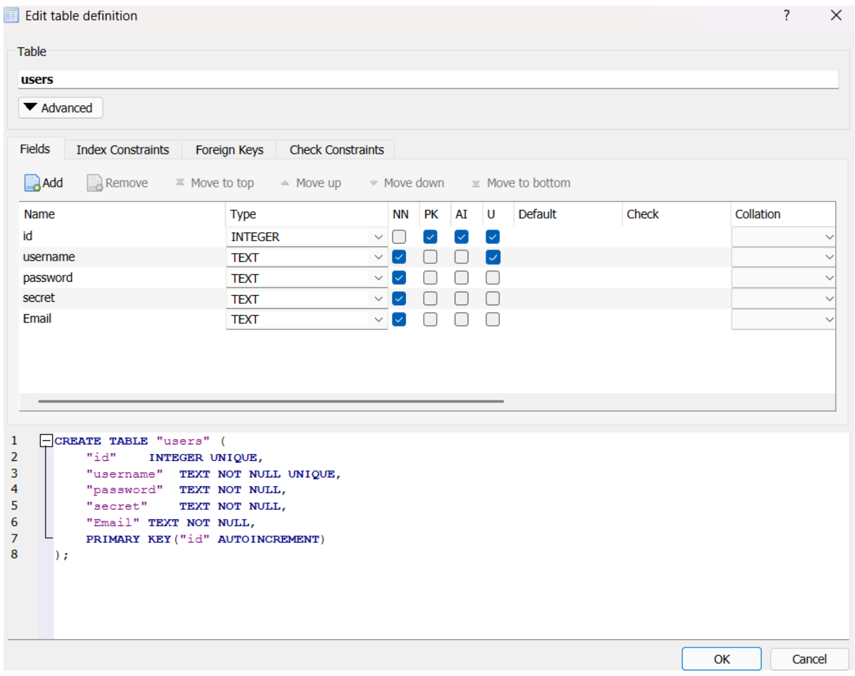
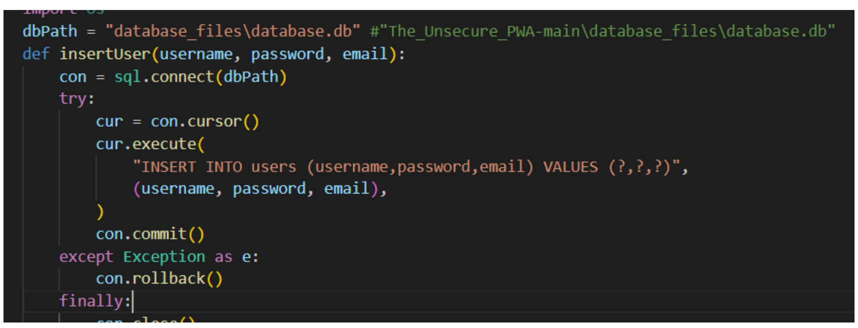
Cross-site scripting (XSS)	I protected against XSS using a CSP which limits which scripts and resources the browser is allowed to load and execute
	
	`app.after_request`
	`ef CSPheader(response):`
	`    # CSP header`
	`    cspPolicy = (`
	`        "default-src 'self'; "`
	`        "script-src 'self'; "`
	`        "style-src 'self' 'unsafe-inline'; "`
	`        "img-src 'self' data:;"`
	`        "font-src 'self'; "`
	`        "connect-src 'self'; "  #  AJAX requests`
	`    )`
	`    response.headers['Content-Security-Policy'] = cspPolicy`
	`    return response`

Vulnerability	Code sample

CSRF	I protected against Cross site request forgery by using CSRF tokens which are essentially tokens to perform an action once, meaning a link cant be used to repeat an action
	● I would initially assing the CSRF token in the GET request
	
	`@app.route('/signup', methods=['GET'])`
	`def GetSignUp():`
	`    #generate CSRF`
	`    state=False`
	`    if session.get('authenticated'):`
	`        state=True`
	`    token = secrets.token_hex(16)`
	`    session['csrf_token'] = token`
	`    #load actual page`
	`    return render_template('signup.html', csrf_token=token, state=state)`
	● I would then verify it and get rid of it, only re assigning it if another GET request was made or a form needed resubmission
	
	`form_token = request.form.get("csrf_token")`
	`session_token = session.get("csrf_token")`
	`if not form_token or form_token != session_token:`
	`    print("csrf error")`
	`    NewToken = secrets.token_hex(16)`
	`    session['csrf_token'] = NewToken`
	`    return jsonify({"error": "CSRF token mismatch", "csrf_token": NewToken}), 400 #cancels the form submission`
	`session.pop("csrf_token", None)`
	
	`return jsonify({"error": "CSRF token mismatch", "csrf_token": NewToken}), 400`

Invalid forwarding and redirecting	I prevented invalid forwarding and redirecting by only trusting internal sites
	● 
	`def SafeUrl(TargetUrl):`
	`    ref_url = urlparse(request.host_url)`
	`    test_url = urlparse(urljoin(request.host_url, TargetUrl))`
	
	`    return (`
	`        test_url.scheme == 'https' and`
	`        ref_url.netloc == test_url.netloc`
	`    )`
	● This tests if the url is safe
	I would then allow the next request if it is safe
	
	`if nextPage and SafeUrl(nextPage):`
	`    return redirect(nextPage)`

Race conditions	To prevent against race conditions i configured my SQL table to only allow unique usernames via DB browser. This hard coded the unique limit and was an extra layer onto of the sign in check that I would do to check for duplicate usernames, preventing race conditions creating multiple accounts of the same username

Vulnerability	Code sample
	 

Edit table definition

Table

users

Advanced

Fields | Index Constraints | Foreign Keys | Check Constraints

Add | Remove | Move to top | Move up | Move down | Move to bottom

Name	Type	NN	PK	AI	U	Default	Check	Collation
id	INTEGER	☐	✓	✓	✓			
username	TEXT	✓	☐	☐	✓			
password	TEXT	✓	☐	☐	☐			
secret	TEXT	✓	☐	☐	☐			
Email	TEXT	✓	☐	☐	☐			

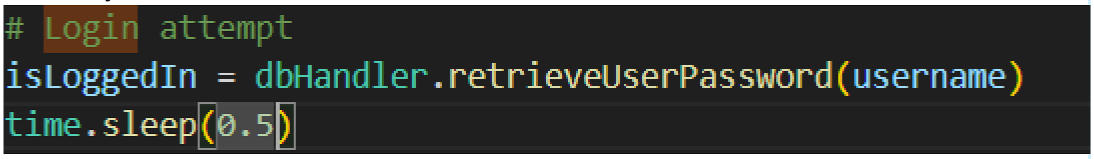
```
CREATE TABLE "users" (
    "id"    INTEGER UNIQUE,
    "username"  TEXT NOT NULL UNIQUE,
    "password"  TEXT NOT NULL,
    "secret"    TEXT NOT NULL,
    "Email" TEXT NOT NULL,
    PRIMARY KEY("id" AUTOINCREMENT)
);
```

OK | Cancel

```
dbPath = "database_files\database.db" #"The_Unsecure_PWA-main\database_files\database.db"
def insertUser(username, password, email):
    con = sql.connect(dbPath)
    try:
        cur = con.cursor()
        cur.execute(
            "INSERT INTO users (username,password,email) VALUES (?,?,?)",
            (username, password, email),
        )
        con.commit()
    except Exception as e:
        con.rollback()
    finally:
```

15. Protecting against file attacks

Provide the client and users with snippets of code that demonstrate how to safeguard user files from side channel attacks.

- To prevent side channel attacks I delayed logins to give attackers less information on the system
- 

```
# Login attempt
isLoggedIn = dbHandler.retrieveUserPassword(username)
time.sleep(0.5)
```

16. Conclusion

Summarise key findings and recommendations for enhancing security in ‘The Unsecure PWA Company’, including the enterprise benefits and how secure practices improve products and influence future developments.

Key findings I have found extend beyond the vulnerabilities such as XSS & SQL injection attacks, plain text passwords, no 2fa, ect. The most important findings are the real world implications of the findings. The business impacts and eventual higher costs of running a vulnerable system and the ethical and legal liabilities that come with it. As a result I believe the improvements I made along with the strategies for future protection have significant enterprise value and should be referenced back in future developments by the Unsecure PWA Company.

17. References

List of resources and references used in the report.

https://www.wiz.io/academy/security-by-design https://www.imperva.com/learn/data-security/privileged-user-monitoring/

https://www.imperva.com/learn/data-security/ueba-user-and-entity-behavior-analytics/

https://www.imperva.com/learn/data-security/access-control-list-acl/

https://www.imperva.com/learn/application-security/api-security/

https://www.imperva.com/learn/application-security/penetration-testing/

https://www.imperva.com/learn/application-security/threat-intelligence/

https://www.imperva.com/learn/data-security/data-protection/

https://www.imperva.com/learn/application-security/cyber-security-threats/

https://www.ipc.nsw.gov.au/fact-sheet-privacy-design

18. Security report appendix

Any additional information or supporting documents.