Software Engineering Stage 6 (Year 12) – teacher support resource

**Secure software architecture**

# Teacher support resource

**Teacher note:** this resource has been designed to facilitate the ready conversion into a student booklet by removing the answers within the response windows. Teacher notes can be deleted before distributing to students. This booklet should be submitted as the documentation component of the assessment task in part or in whole.

Student name:

Class:

Teacher:

Contents

[Teacher support resource 1](#_Toc183426576)

[Unit overview 4](#_Toc183426577)

[T eaching advice on Secure software architecture 10](#_Toc183426578)

[Task description 11](#_Toc183426579)

[Submission details 13](#_Toc183426580)

[Component A 13](#_Toc183426581)

[Component B 13](#_Toc183426582)

[Steps to success 14](#_Toc183426583)

[What is the teacher looking for? 17](#_Toc183426584)

[Teaching advice 18](#_Toc183426585)

[Marking guidelines 19](#_Toc183426586)

[Student-facing rubric 21](#_Toc183426587)

[Glossary 24](#_Toc183426588)

[NESA glossary keywords 31](#_Toc183426589)

[Secure software architecture pre-test 32](#_Toc183426590)

[Section 1 – Secure software architecture 32](#_Toc183426591)

[Section 2 – designing software 33](#_Toc183426592)

[Secure software architecture 37](#_Toc183426593)

[Designing software 38](#_Toc183426594)

[Developing secure code 67](#_Toc183426595)

[Design, develop and implement code considering efficient execution for the user 103](#_Toc183426596)

[Design, develop and implement secure code to minimise vulnerabilities in user action controls 106](#_Toc183426597)

[Impact of safe and secure software development 122](#_Toc183426598)

[Appendix 1 – security report 128](#_Toc183426599)

[1. Executive summary 131](#_Toc183426600)

[2. Introduction 132](#_Toc183426601)

[3. Benefits of developing secure software 134](#_Toc183426602)

[4. Fundamental software development steps to develop secure code 136](#_Toc183426603)

[5. Influence of end users on secure design features 141](#_Toc183426604)

[6. Developing secure code 142](#_Toc183426605)

[7. Security features incorporated into software 143](#_Toc183426606)

[8. Security by design approach 144](#_Toc183426607)

[9. Testing and evaluating security and resilience 146](#_Toc183426608)

[10. Strategies for managing security of programming code 147](#_Toc183426609)

[11. Defensive data input handling 148](#_Toc183426610)

[12. Safe API development 148](#_Toc183426611)

[13. Efficient execution for the user 149](#_Toc183426612)

[14. Secure code for user action controls 149](#_Toc183426613)

[15. Protecting against file attacks 150](#_Toc183426614)

[16. Conclusion 150](#_Toc183426615)

[17. References 150](#_Toc183426616)

[18. Security report appendix 151](#_Toc183426617)

[Appendix 2 – enhancing security of a progressive web app (PWA) with security measures 152](#_Toc183426618)

[Appendix 3 – how to guide – 2FA using Google Authenticator and Flask 156](#_Toc183426619)

[Appendix 4 – secure Flask Bootstrap PWA template 162](#_Toc183426620)

[Advice for developers 164](#_Toc183426621)

[How to guide 165](#_Toc183426622)

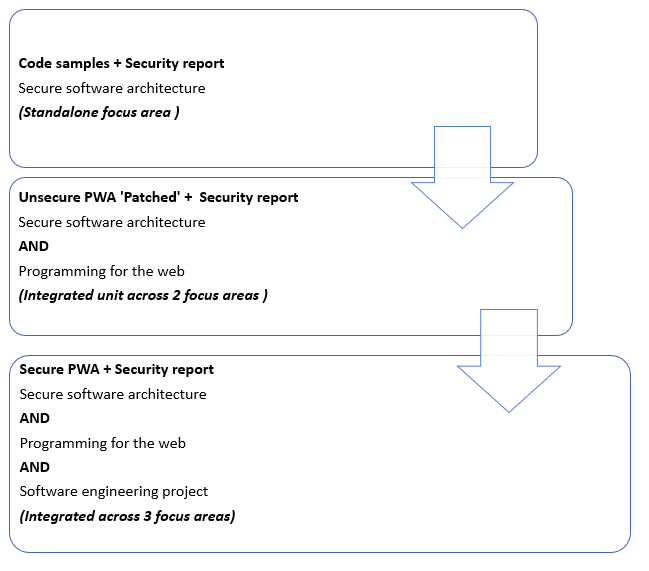
[References 167](#_Toc183426623)

[Legislation 172](#_Toc183426624)

# Unit overview

Options are provided to differentiate content depending upon students’ experience. At a minimum, students complete the security report template for the scenario ([Appendix 1](#_Appendix_1_:)). Students attempt coding exercises to secure the progressive web app (PWA) ([Appendix 2](#_Appendix_2:_Enhancing)). Alternatively, teachers and students may choose to develop a secure app from scratch using Python and the Flask framework ([Appendix 3](#_Appendix_3:_How) and [Appendix 4](#_Appendix_4:_Secure)). Assessment tasks that accompany these integrated unit projects include: Steps to success, Marking guidelines and Student facing rubrics. They are available in the **Secure Software** folder under the **Files** tab of the [Software Engineering channel](https://schoolsnsw.sharepoint.com/:f:/r/sites/TASNSWStatewideStaffroom/Shared%20Documents/13.%20%F0%9F%A7%91%E2%80%8D%F0%9F%92%BB%20Software%20Engineering%2011-12/Secure%20Software%20Architecture?csf=1&web=1&e=6nkPE3) via the TAS Statewide staffroom.

Figure 1 – standalone to integrated focus areas



This unit guides students through the Secure software architecturefocus area. The content for this focus area has been designed to be delivered as theory required for and integrated into testing and securing the progressive web application (PWA). Students are provided with the following scenario:

Your client, ‘The Unsecure PWA Company’, has engaged you 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 progressive web app (PWA)](https://github.com/TempeHS/The_Unsecure_PWA).

The activities within the teacher support resource (TSR) go beyond those required to achieve the outcomes related to the enhanced security of the PWA and could be used to inform a major project for this course. Teachers are encouraged to issue the Unsecure PWA assessment task to students to work on, alongside other lessons that deliver course content.

**Teacher note**: the progressive web app has been designed with a range of security vulnerabilities. The app has been specifically designed for students studying the Secure software architecture focus area from the [NESA HSC Software Engineering Course](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/content/n12/fa039e749d). The app is NOT secure and should NOT be used in any production environment.

During Weeks 1 to 2, students complete a pre-test on fundamental concepts within the Secure software architecture focus area. This helps teachers determine the background knowledge and understanding of their class and which of their students may have completed the Software development: Developing apps and web software focus area from the Computing Technology 7–10 Syllabus, as well as the understanding they have developed through the Programming for the web focus area from the Software Engineering syllabus. It also establishes a common language teachers may allocate to groups, according to expertise based on the results of this pre-test.

Students develop an overview of the latest trends impacting the internet and cybersecurity. They choose concepts referred to in the Software Engineering 11–12 Syllabus and create a single slide.

As students experience these concepts during coding activities, they provide a summary of each, including definitions, characteristics and examples, in table format. These include data protection, cyber attacks, static application security testing (SAST), dynamic application security testing (DAST), vulnerability assessment, penetration testing, application programming interface (API) security, cross-site scripting (XSS), cross-site request forgery (CSRF).

The slide deck provides a study resource and reference guide that is revisited during project production and theory classes.

For the enhanced security of their PWA project, students familiarise themselves with the existing [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA).

In Weeks 3 to 4, students describe the benefits of developing secure software to protect data and minimise cyber attacks and vulnerabilities by completing activities within the teacher support resource (TSR). They study and respond to lessons from [Software Security Case Studies](https://www.blueoptima.com/4-lessons-from-software-security-case-studies/).

Students establish and maintain a news board, journal or folder of hyperlinks to news stories from cybersecurity incidents to refer to in class as case studies. They complete matching exercises from the TSR on requirements definition, determining specifications, design and development phases, and apply these to ‘The Unsecure PWA Company’ scenario and their security report.

For their project, students interpret and apply fundamental software development steps to develop secure code. They research and implement two-factor authentication (2FA).

Students complete activities on integration, testing and maintenance in the TSR and apply these to the development of their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) project.

Students describe how the capabilities and experience of end users influence the secure design features of software. They apply this to the development of their secure [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) projects.

Students complete the activities in the TSR.

During Weeks 5 to 6, students explore fundamental software design security concepts when developing programming code, including confidentiality, integrity, availability, authentication, authorisation and accountability by completing activities in the TSR and applying their understanding to ‘The Unsecure PWA Company’ scenario.

Students research, define and create an audit of their application of security features incorporated into their software project, including data protection, security, privacy and regulatory compliance. They use and explain the contribution of cryptography and sandboxing to the ‘security by design’ approach in the development of software solutions Students complete the activities related to cryptography and sandboxing within the TSR.

Students visit the [Australian Signals Directorate](https://www.cyber.gov.au/resources-business-and-government/governance-and-user-education/secure-by-design) website and research: ‘What is security by design?’ They complete TSR activities to define ‘security by design’, including providing an analogy and explaining why it is important.

Students visit the [Privacy by design](https://www.oaic.gov.au/privacy/privacy-guidance-for-organisations-and-government-agencies/privacy-impact-assessments/privacy-by-design) by OAIC (Office of the Australian Information Commissioner) [Fact Sheet - Privacy by design](https://www.ipc.nsw.gov.au/fact-sheet-privacy-design) by IPC (Information and Privacy Commission)websites to summarise a definition of ‘privacy by design’, including key concepts of a proactive not reactive approach, embedding privacy into design and respect for user privacy.

Students design, develop and perform a role-play based on a security breach case study and simulate approaches to business continuity, including the incident response and disaster recovery.

Students test and evaluate the security and resilience of software by determining vulnerabilities, hardening systems, handling breaches, maintaining business continuity and conducting disaster recovery. They complete the activities in the TSR.

Students research case studies and use vulnerability scanning tools to identify security weaknesses in a sample application or system, including their own secure PWA.

Students visit the [Guidelines for System Hardening](https://www.cyber.gov.au/resources-business-and-government/essential-cyber-security/ism/cyber-security-guidelines/guidelines-system-hardening) by ASD (Australian Signals Directorate)website to research guidelines for systems hardening and to identify best practices for securing software systems.

Students complete the activities in the TSR.

In Weeks 7 to 9, students research strategies used by software developers to manage the security of programming code, including code review, static application security testing (SAST), dynamic application security testing (DAST), vulnerability assessment and penetration testing. They then apply and evaluate these to their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) project.

Students research code for defensive data input handling practices, including input validation, sanitisation and error handling They design, develop and implement this into their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) project.

Students design, develop and implement a safe application programming interface (API) to minimise software vulnerabilities.

Students design, develop and implement code, considering efficient execution for the user, including memory management, session management and exception management.

Students design, develop and implement secure code to minimise vulnerabilities in user action controls, including broken authentication and session management, cross-site scripting (XSS) and cross-site request forgery (CSRF), invalid forwarding and redirecting, and race conditions. They apply these to their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) project.

Students design, develop, and implement secure code to protect user file and hardware vulnerabilities from file attacks and side channel attacks. They apply and describe the benefits of collaboration to develop safe and secure software.

Students complete the activities in the TSR.

In Week 10, students investigate and explain the benefits to an enterprise of the implementation of safe and secure development practices, including improved products or services, influence on future software development, improved work practices, productivity and business interactivity.

Students evaluate the social, ethical, and legal issues and ramifications that affect people and enterprises resulting from the development and implementation of safe and secure software, including employment, data security, privacy, copyright, intellectual property and digital disruption.

Students complete the activities in the TSR.

Students present their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) solution using presentation software to the class (playing the role of ‘The Unsecure PWA Company’ client). They submit their project documentation and code.

This teacher support resource (TSR) provides activities that align with the content of the syllabus to ensure content is covered. After the initial linear delivery of this unit, teachers will identify opportunities to integrate theory and practice within single lessons. The secure software project is developed in the classroom under the supervision of the teacher using explicit teaching methods outlined in this TSR.

## Teaching advice on Secure software architecture

**Why is it important?**

The Secure software architecture focus area is crucial for students, as it helps them learn about creating secure software. Students explore industry-recognised methods for writing secure code and understanding how these methods contribute to developing safe software. This focus area also allows students to consider the social, ethical, and legal issues that can arise for individuals and businesses when using secure software.

**Additional advice**

Students can choose to study this focus area on its own or combine it with the Programming for the web topic. This combination can help reinforce important concepts for software development engineers.

For example, the PWA task made as a [Python Flask version](https://github.com/TempeHS/Flask_PWA_Programming_For_The_Web_Task_Source) for Programming the web could be made to be secure by students ‘from scratch’.

This could be delivered to integrate the focus areas: Programming for the web, Secure software architecture and the Software engineering project.

## Task description

Your client, ‘The Unsecure PWA Company’, has engaged you 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 progressive web app (PWA)](https://github.com/TempeHS/The_Unsecure_PWA).

You are to:

* run a range of security tests and scans along with a white/grey/ black box analysis of the application to identify as many security and privacy vulnerabilities as possible.
* provide examples of secure code solutions, for example, the application programming interface [API] may include a web-based login component that provides a solution using a [Python and Flask framework](https://github.com/TempeHS/Flask_PWA_Programming_For_The_Web_Task_Source) to allow for two-factor authentication (2FA) and cross-site request forgery (CSRF) protection and session management
* prepare a professionally written security report for your client that includes
* an overview of your approach to the technical analysis
* documentation of the scope of privacy and security issues, including
* security or privacy issues that cannot be mitigated by technical engineering solutions
* security issues that must be tested in the production environment
* security or privacy vulnerabilities you discover and an impact assessment of each
* recommendations relating to a security and privacy by design approach going forward for ‘The Unsecure PWA Company’
* designing and developing implementations using HTML/CSS/JS/SQL/JSON/Python code and/or web content changes as required to patch each vulnerability you discover.

**Outcomes being assessed**:

A student:

* justifies methods used to plan, develop and engineer software solutions **SE-12-01**
* applies structural elements to develop programming code **SE-12-02**
* evaluates practices to safety and securely collect, use and store data **SE-12-04**
* justifies the selection and use of tools and resources to design, develop, manage and evaluate software **SE-12-06**
* designs, develop and implements safe and secure programming solutions **SE-12-07**
* tests and evaluates language structures to refine code **SE-12-08**
* applies methods to manage and document the development of a software project   
  **SE-12-09**

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

**Suggested weighting**: 20%

# Submission details

Students are to submit 3 components – A, B and C.

## Component A

A complete security report (scaffolded example in the Appendix of the teacher support resource).

See the [course specifications](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) for standardised versions of project documentation. These need to be adhered to throughout the course and may form part of examinable content.

## Component B

A zip file of any implemented solution or sample secure code.

Component C

A presentation to the class who are playing the role of the client ‘The Unsecure PWA Company’ with questions and answers.

## Steps to success

Table 1 – assessment preparation schedule

|  |  |
| --- | --- |
| ****Steps**** | ****What I need to do**** |
| ****Test and evaluate the progressive web app (PWA)**** | 1. Experiment with the unsecure PWA. 2. Investigate how the unsecure PWA is built to see how data is handled. 3. Locate any security weaknesses in the code and how data is handled. 4. Conduct a security audit and vulnerability assessments of the unsecure PWA using special tools and tests to find any problems. 5. Document the processes you followed and the vulnerabilities or issues you discovered. This could include:  * browser developer tool reports (Google Lighthouse and Edge Application) * black box test the app for vulnerabilities and privacy issues * grey box test the app * code review/white box asses the application * Use of third-party tools like [Pentest-Tools.com](https://pentest-tools.com) or [Zed Attack Proxy (ZAP)](https://www.zaproxy.org/). |
| ****Designing software**** | * Write a requirements definition for the client. * Identify the user specifications for the solution. * Apply the fundamental software development steps to develop secure code. * Identify any problems that cannot be fixed by changing the code, such as how users behave or how data is managed.   Consider:   * What will be different when the application is in a production environment? * What might users do that you or the ‘The Unsecure PWA Company’ cannot control that could be a vulnerability or a privacy issue? * What can’t be tested? |
| ****Developing secure code**** | 1. Design, develop and implement code that changes the PWA's code and settings to fix the security issues found, like checking user inputs, encrypting data and controlling accessibility. 2. Use appropriate HTML/CSS/JS/SQL/JSON/Python code and web content changes to provide a close-to-industry standard solution that fully or near fully mitigates security and privacy vulnerabilities. 3. Apply strategies to manage the security of programming code. 4. Test and evaluate the security and resilience of the software. |
| ****Impact of safe and secure software development**** | 1. Describe the benefits of developing secure software to your client ‘The Unsecure PWA Company’. 2. Provide expert advice to the ‘The Unsecure PWA Company’ on the privacy and security of their progressive web app (PWA). 3. List all the security or privacy issues found and explain their impact if someone took advantage of them by providing an impact assessment of each. 4. Present your solution and security report to the client. 5. Explain the benefits of implementing safe and secure development practices to an enterprise. 6. Explain the social, ethical and legal issues that affect people and enterprises, resulting from the development and implementation of safe and secure software. |

## What is the teacher looking for?

Students have:

* performed a thorough analysis of the documentation, code, user interfaces and user experiences to discover the most vulnerabilities and privacy issues, and identified and explained the most out of scope vulnerabilities
* documented evidence of the effective application of a variety of automated tools and brute force attacks to discover the most vulnerabilities
* investigated how the PWA is built to find any security weaknesses in the code and how data is handled
* found any problems that cannot be fixed by changing the code, like how users behave or how data is managed
* suggested ways to build future PWAs with appropriate security and privacy from the start
* made changes to the PWA's code and settings to fix the security issues found, like checking user inputs, encrypting data and controlling who can access what
* used appropriate HTML/CSS/JS/SQL/JSON/Python code and web content changes to provide a close-to-industry standard solution that fully or near fully mitigates security and privacy vulnerabilities
* completed the security report and presented findings and a more secure PWA to the client during a class presentation.

## Teaching advice

This app has been designed as either a teaching tool, an assessment tool or an assessment as learning tool.

As a teaching tool, the teacher can use the app to demonstrate discrete vulnerabilities, then teach the preferred patch method.

As an assessment tool, students should be taught the knowledge and skills, then given the app to analyse and report on before designing and developing appropriate patches (patching all will be time prohibitive).

As an assessment as learning tool, teachers can teach vulnerabilities in the app, then support students to design and develop patches while assessing them formatively. As an assessment tool and for support to teachers guiding learning toward a solution, a [checklist of known vulnerabilities](https://schoolsnsw.sharepoint.com/:w:/r/sites/TASNSWStatewideStaffroom/Shared%20Documents/13.%20%F0%9F%A7%91%E2%80%8D%F0%9F%92%BB%20Software%20Engineering%2011-12/Secure%20Software%20Architecture/Teachers%20Only_The%20Unsecure%20Progressive%20Web%20App_Known%20Vunerabilities.docx?d=w68092224e05d46daa77e4cb0b08b1a04&csf=1&web=1&e=SvLYin) for the Unsecure PWA will be provided in the Software Engineering channel of the Statewide staffroom.

**Note:** technical support officers (TSOs) or computer coordinators may provide an old laptop to recycle into a web server (a desktop or Raspberry Pi could also be used). The web server would ensure delivery of the syllabus content in [Secure software architecture](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/content/year-12/fa039e749d) is done with an authentic, engaging and project-based approach and could be used as a foundation for the Year 12 project.

However, this unit, including the assessment task **does not** require this extra hardware. It does include dependencies: VSCode, Python 3+, Flask (pip install flask).

The learning experience will inform students who are considering extending on the Secure software architecture focus area for their major project. They will deepen their knowledge of the ‘plumbing’ and coding required at both front and back-ends of the web.

## Marking guidelines

Table 2 – assessment marking guidelines

|  |  |
| --- | --- |
| ****Grade**** | ****Marking guideline descriptors**** |
| ****A**** | The student:   * demonstrates an extensive understanding of the steps used by programmers when designing software * documents the software development steps thoroughly in a highly developed security report * develops highly effective algorithms that demonstrate the logic required for a software solution * uses highly effective analysis in the development of computing solutions using the <python> programming language and <Flask> framework.\* |
| ****B**** | The student:   * demonstrates a thorough understanding of the steps used by programmers when designing software * documents the software development steps in a well-developed security report * develops effective algorithms that demonstrate the logic required for a software solution * uses effective analysis in the development of computing solutions using the <python> programming language and <Flask> framework.\* |
| ****C**** | The student:   * demonstrates a sound understanding of the steps used by programmers when designing software * documents some of the software development steps in a sound security report * develops sound algorithms to demonstrate the logic required for a software solution * uses appropriate computing solutions using the <python> programming language and <Flask> framework.\* |
| ****D**** | The student:   * demonstrates a basic understanding of the steps used by programmers when designing software * documents some of the software development steps in the security report * develops basic algorithms to demonstrate some of the logic required for a software solution * implements basic computing solutions using the<python> programming language and <Flask> framework.\* |
| ****E**** | The student:   * identifies the requirements for the documentation and production of a software solution. |

\* Insert relevant <programming language> and <framework>.

## Student-facing rubric

Table 3 – rubric for Secure software architecture

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Submission criteria | Limited | Basic | Sound | High | Outstanding |
| ****Criteria 1****  ****Designing software**** | Labels the fundamental software development steps to develop secure code. | Identifies the fundamental software development steps to develop secure code. | Applies and outlines the fundamental software development steps to develop secure code. | Applies and explains fundamental software development steps to develop secure code. | Applies and justifies the use of all fundamental software development steps to develop secure code. |
| ****Criteria 2****  ****Developing solutions with code**** | Completes missing code to solve the task.  Documents use of data structures and errors. | Modifies code to solve the task.  Code includes some use of the control structures, 2FA, CSRF, some use of data structures and parameter passing.  Identifies their use of at least one software debugging tool.  Correctly describes at least one error experienced in the coding of their solution. | Creates code to solve the task.  Code includes some use of the control structures, 2FA, CSRF, some use of data structures and parameter passing.  Defines and documents their use of at least one software debugging tool.  Describes at least one error experienced in the coding of their solution. | Creates and documents secure code to solve the task.  Code includes correct use of most parameter passing, most of the control structures, 2FA, CSRF and use of data structures and files.  Correctly describes and documents a range of errors experienced in the coding of their solution and the debugging tools used to find and correct them. | Creates, documents, and justifies the use of efficient and secure code to solve the task.  Code includes correct use of parameter passing, all the control structures, 2FA, CSRF, effective use of data structures and files.  Correctly describes and documents, in detail, a range of errors experienced in the coding of their solution and how debugging tools were used to correct them. |
| ****Criteria 3****  ****Impact of safe and secure software development**** | Labels the session management technique used in their project.  Identifies a ‘Data dictionary’.  Identifies the data structures used in their project.  Identifies a security strategy used in programming code. | Defines the session management technique used in their project.  Fills in an incomplete ‘Data dictionary’.  Defines the data structures used in their project.  Defines a security strategy used in programming code. | Creates a ‘Data dictionary’ for their project.  Defines the data structures used in their project.  Discusses the session management technique used in their project.  Defines a security strategy for protecting their programming code. | Creates a comprehensive ‘Data dictionary’ for use with their project.  Explains the data structures and discusses their use in their project.  Explains their selection of session management technique used in their project.  Explains appropriate security strategies for their programming code. | Presents a secure PWA and a comprehensive security report to the client explaining the benefits to their enterprise of the implementation of safe and secure development practices.  Justifies the use of a comprehensive ‘Data dictionary’ for their project.  Justifies the data structures used in their project.  Justifies how their file is protected against vulnerabilities.  Justifies their selection of session management technique used in their project.  Justifies appropriate security strategies for their programming code. |

# Glossary

Teachers may choose to **cut up this table** and provide a **matching exercise** for teams of students as a classroom challenge. Alternatively, they can empty the definition column and have students research the meaning and complete the table.

The following words will gather more meaning as students work through this booklet.

Table 4 – glossary

|  |  |
| --- | --- |
| ****Word**** | ****Definition**** |
| **Two-factor authentication (2FA)** | A security process that requires users to provide 2 different authentication factors to gain access. |
| **Accountability** | The concept of individuals being responsible for their actions and decisions within a system. |
| **Application programming interface (API)** | An application programming interface is a way for 2 or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software. |
| **Authentication** | The process of verifying the identity of a user or system. |
| **Availability** | Ensuring that a system or service is accessible and usable when needed. |
| **Black box analysis** | A method of analysing software security where no knowledge of the internal workings is involved. |
| **Bootstrap** | A popular framework used for designing websites and web applications. It provides pre-made styles and components, like buttons, forms and navigation bars, which help you create a professional-looking interface quickly and easily. |
| **Broken authentication and session management** | Security vulnerabilities related to ineffective management of user authentication and session handling. |
| **Brute-force attacks** | A trial-and-error method used to obtain information, such as passwords or encryption keys. |
| **Code review** | A process of examining and evaluating code for quality, security and compliance with coding standards. |
| **Confidentiality** | Ensuring that sensitive information is only accessed by authorised individuals. |
| **Cross-site request forgery (CSRF) protection** | A security measure to prevent unauthorised commands from being executed on behalf of an authenticated user. |
| **Cross-site scripting (XSS)** | A security vulnerability where attackers inject malicious scripts into webpages viewed by other users. |
| **Cryptography** | The practice and study of techniques for secure communication and data protection. |
| **Data encryption** | The process of converting data into a code to prevent unauthorised access. |
| **Data privacy policy** | A statement outlining how an organisation collects, uses and protects personal data. |
| **Data protection** | Safeguarding data from unauthorised access or alterations. |
| **Data sanitisation** | The process of cleaning and validating input data to prevent security vulnerabilities. |
| **Data validation** | Verifying that data input meets specific criteria, ensuring its integrity and security. |
| **Dynamic application security testing (DAST)** | A method of testing software applications for security vulnerabilities while they are running. |
| **Encryption** | The process of encoding information to make it unreadable without the proper decryption key. |
| **Error handling** | Strategies and mechanisms for managing and resolving errors or unexpected situations in software. |
| **Exception management** | Handling and addressing exceptions or errors that occur during software execution. |
| **GitHub** | A web-based platform for hosting and sharing code repositories. |
| **Grey box analysis** | A method of analysing software security where partial knowledge of the internal workings is involved. |
| **HTML/CSS/JS/SQL/JSON/Python** | Common programming languages and technologies used in web development and software engineering. |
| **Impact assessment** | Evaluation of the potential consequences of security or privacy vulnerabilities. |
| **Installation** | The process of deploying software onto a system or network for use. |
| **Integrity** | Ensuring that data remains accurate and unchanged throughout its lifecycle. |
| **Maintenance** | Activities involved in keeping software operational, updated and secure over its lifespan. |
| **Middleware** | A term used in web development to refer to functions that sit between the request and response lifecycle in a web application. |
| **Parsing** | The process of breaking down data into smaller, understandable parts to convert data from one format to another so that a program can work with it effectively.  For example, turning a JSON string into a JavaScript object to access its properties. |
| **Penetration testing** | Simulating cyber attacks to identify and exploit vulnerabilities in software applications. |
| **Privacy** | Protecting personal information and data from unauthorised access and misuse. |
| **Progressive web app (PWA)** | A type of application software delivered through the web, built using common web technologies like HTML, CSS and JavaScript. |
| **Quality assurance** | Practices and processes used to ensure that software meets specified requirements and quality standards. |
| **Recommendations** | Suggestions for improving security and privacy practices in software development. |
| **Regulatory compliance** | Adhering to laws, regulations, and standards relevant to the industry or jurisdiction in which software operates. |
| **Requirements definition** | Establishing and documenting the needs and functional specifications of a software application. |
| **Resilience** | The ability of software to withstand and recover from system failures or cyber attacks |
| **Safe application programming interface (API)** | Developing and implementing APIs that minimise security vulnerabilities. |
| **Sandbox** | Isolating software processes to prevent them from affecting other parts of the system. |
| **Secure Sockets Layer (SSL) certificate** | A digital certificate that authenticates the identity of a website and enables secure connections. |
| **Secure Sockets Layer (SSL) and/or Hypertext Transfer Protocol Secure (HTTPS)** | Protocols used to secure communication over a computer network. |
| **Security scans** | Automated tools used to scan and detect security vulnerabilities in software applications. |
| **Security specialist** | An individual with expertise in analysing and addressing security vulnerabilities in software applications. |
| **Security tests** | Tests conducted to identify and address security flaws in software applications. |
| **Session management** | Process of securely managing user sessions on a website or application. |
| **Session timing** | Setting a limit on how long a user session remains active before requiring re-authentication. |
| **Software development lifecycle** | The process of planning, creating, testing and deploying software applications. |
| **Software engineering** | The application of engineering principles to the design, development, maintenance, testing, and evaluation of software and systems. |
| **Static application security testing (SAST)** | A method of testing software applications for security vulnerabilities without executing the code. |
| **Structured Query Language (SQL) injection** | A type of cyber attack where malicious SQL statements are inserted into an entry field for execution. |
| **Testing and debugging** | Processes of evaluating software for errors and issues and identifying and resolving them. |
| **Vulnerabilities** | Weaknesses in a system that can be exploited to compromise security. |
| **White box analysis** | A method of analysing software security where full knowledge of the internal code is involved. |

**Teacher note:** for students with an English as an additional language or dialect (EAL/D) background, the completed glossary can be provided so that they have additional time to understand the key terms with bilingual dictionaries. The glossary can be provided to students in their preferred communication mode. Teachers may consider using [semantic wave (PDF 422 KB)](https://static.raspberrypi.org/files/curriculum/quickreads/6-Pedagogy_Summary_Semantic_Waves_V3_2023.pdf) strategies to explore the meaning of these concepts.

## NESA glossary keywords

NESA keywords can be used in the syllabus and in the Higher School Certificate examination. Familiarisation with these keywords can assist in understanding how to write and respond to questions.

|  |  |
| --- | --- |
| Key term | Definition |
| Apply | Use, utilise, employ in a particular situation. |
| Describe | Provide characteristics and features. |
| Explain | Relate cause and effect; make the relationships between things evident; provide why and/or how. |
| Investigate | Plan, inquire into and draw conclusions about. |

[NESA: Glossary of key words](https://educationstandards.nsw.edu.au/wps/portal/nesa/11-12/hsc/hsc-student-guide/glossary-keywords)

**Teacher note:** develop, explore, select, and verify are used in this topic and are not listed.

# Secure software architecture pre-test

**Teacher note**: students will come to this focus area from diverse backgrounds, experience and understanding. There is no prerequisite mandated for this course and students may not have studied the Computing Technology 7–10 Syllabus.

A pre-test will enable teachers to determine a baseline measurement of understanding, track growth in understanding and help determine next steps. Teachers activate prior knowledge using [KWLH charts](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/562). These are explicit teaching strategies.

This pre-test could be delivered as a [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645) activity to promote classroom discussion and understanding. Students are advised that these concepts will be applied during the development of their assessment task for this focus area and inform their Software engineering project.

## Section 1 – Secure software architecture

1. Define ‘secure software architecture’.

|  |
| --- |
| ‘Secure software architecture’ refers to the structure and design of a software system that incorporates security measures to protect against cyber threats and vulnerabilities. |

1. List 3 benefits of developing secure software.

|  |
| --- |
| 1. Reduced risk of data breaches and unauthorised access. 2. Enhanced trust and credibility with users. 3. Lower maintenance costs associated with fixing security issues. |

1. Explain the importance of data protection in secure software development.

|  |
| --- |
| Data protection is essential in secure software development as it ensures that sensitive information is encrypted, access is restricted to authorised users, and data is securely stored and transmitted. |

1. How can software architecture help minimise cyber attacks and vulnerabilities?

|  |
| --- |
| Software architecture can help minimise cyber attacks and vulnerabilities by implementing security controls such as firewalls, encryption, access controls and secure coding practices. |

## Section 2 – designing software

1. What are the fundamental software development steps?

|  |
| --- |
| The fundamental software development steps include requirements definition, determining specifications, design, development, integration, testing and debugging, installation and maintenance. |

1. Describe the process of requirements definition in software development.

|  |
| --- |
| Requirements definition involves gathering and documenting the functional and non-functional requirements of the software project, ensuring a clear understanding of what needs to be achieved. |

1. Why is design a crucial step in software development?

|  |
| --- |
| Design is a crucial step in software development as it involves creating the blueprint for the software system, including the structure, components and interactions between different modules. |

1. How does integration contribute to the overall success of a software project?

|  |
| --- |
| Integration is essential as it involves combining individual software components into a cohesive system, ensuring that they work together seamlessly and meet the specified requirements. |

Section 3 – developing secure code

1. Explain the significance of testing and debugging in developing secure code.

|  |
| --- |
| Testing and debugging are essential in developing secure code to identify and fix vulnerabilities, errors and weaknesses in the software before deployment, ensuring a higher level of security. |

1. Describe the importance of maintenance in the context of secure software development.

|  |
| --- |
| Maintenance is important in secure software development to address any security issues that may arise post-deployment, apply patches and updates and continuously improve the security of the software. |

1. List the steps involved in the installation of secure software.

|  |
| --- |
| 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. |

Section 4 – end-user influence

1. How do the capabilities of end users impact the design features of software?

|  |
| --- |
| The capabilities of end users impact the design features of software by influencing the user interface, accessibility options, customisation preferences, and security settings based on the users' needs and technical skills. |

1. Explain how the experience of end users influences the development of secure software.

|  |
| --- |
| The experience of end users influences the development of secure software by providing feedback on usability, security concerns, performance issues and feature requests, helping developers enhance the software to meet user expectations and securityrequirements. |

**Teacher note:** computing syllabuses often use technical subject-specific language with lots of jargon to learn. This terminology has precise technical meaning. Familiarity with these terms can be developed by breaking down phrases into component words and concepts to help extract meaning for the whole phrase.

**Activity 1:** the title of this focus area is written on the whiteboard. What does this term mean? Group students into teams of 3. Each team researches their single-word definition in the context of software engineering. Students convene as a class to share a common definition.

#### Secure

|  |
| --- |
| In the context of software engineering, ‘secure’ refers to the state of being protected from unauthorised access, cyber threats and vulnerabilities. Secure software ensures that data, systems, and users are safeguarded against potential risks and attacks through the implementation of security measures and protocols. |

#### Software

|  |
| --- |
| ‘Software’ represents the programs, applications, and systems that are designed and developed to perform specific functions on computers or electronic devices. In the context of secure software architecture, this component highlights the focus on the design and structure of software systems to ensure they are secure and resilient against security threats. |

Architecture

|  |
| --- |
| ‘Architecture’ in software engineering refers to the high-level structure and design of software systems, including the organisation of components, modules and interactions within the system. ‘Secure software architecture’ involves the strategic planning and design of software systems with security considerations in mind, ensuring that security features are integrated into the overall structure of the software. |

|  |
| --- |
| Combining these 3 component words – ‘secure’, ‘software’ and ‘architecture – the term ‘secure software architecture’ conveys the concept of designing and developing software systems with a focus on security, protection, and resilience against cyber threats and vulnerabilities. It emphasises the importance of incorporating security measures into the architectural design of software to create robust, protected and trustworthy systems in today's digital landscape. |

# Secure software architecture

**Activity 2:** Why is studying secure software architecture important?

|  |
| --- |
| To:   * develop knowledge, understanding and skills associated with developing secure software * investigate and practise industry-recognised techniques for developing secure programming code and understand how these techniques contribute to the development of secure software * evaluate the social, ethical and legal issues that affect people and enterprises, resulting from the development and implementation of safe and secure software. |

**Activity 3:** teacher introduces students to [Lessons](https://www.hacksplaining.com/lessons) by Hacksplaining.

Students learn interactively about each vulnerability as it is addressed in the course or as homework activity.

Students should (as a minimum) be familiar with:

* [Structured query language (SQL) injection](https://www.hacksplaining.com/lessons/sql-injection/start)
* [cross-site scripting (XSS)](https://www.hacksplaining.com/lessons/xss-stored/start)
* [cross-site forgery request (CSRF)](https://www.hacksplaining.com/lessons/csrf/start)
* [invalid forwarding and redirecting](https://www.hacksplaining.com/lessons/open-redirects/start)
* [memory management](https://www.hacksplaining.com/lessons/information-leakage/start)
* [session mismanagement](https://www.hacksplaining.com/lessons/session-fixation/start)
* [broken authentication](https://www.hacksplaining.com/lessons/password-mismanagement/start)
* [race conditions](https://www.youtube.com/watch?v=mq59FxItJ2w).

## Designing software

**Activity 4:** to develop an overview of the latest trends impacting the internet and cybersecurity, students will access the [Imperva Learning Center](https://www.imperva.com/learn/).

Students list the 15 categories presented.

Students identify and list any of the hyperlinked subheadings they are not familiar with:

Table 5 – latest trends impacting the internet and cybersecurity

|  |  |
| --- | --- |
| Key cybersecurity concepts | Links |
| Cybersecurity 101 | * [OSI Model](https://www.imperva.com/learn/application-security/osi-model/) * [Ransom DDoS](https://www.imperva.com/learn/ddos/ransom-ddos-rddos/) * [Cybersecurity Threats](https://www.imperva.com/learn/application-security/cyber-security-threats/) * [DNS Protection](https://www.imperva.com/learn/application-security/dns-protection/) |
| Data security | * [Data Security](https://www.imperva.com/learn/data-security/data-security/) * [Data Protection](https://www.imperva.com/learn/data-security/data-protection/) * [Data Loss Prevention (DLP)](https://www.imperva.com/learn/data-security/data-loss-prevention-dlp/) * [Database Security](https://www.imperva.com/learn/data-security/database-security/) |
| Web and application security | * [Application Security: The Complete Guide](https://www.imperva.com/learn/application-security/application-security/) * [What Is WAF](https://www.imperva.com/learn/application-security/what-is-web-application-firewall-waf/) * [API Security](https://www.imperva.com/learn/application-security/api-security/) * [Web Application Security](https://www.imperva.com/learn/application-security/web-application-security/) |
| Attack types | * [Account Takeover](https://www.imperva.com/learn/application-security/account-takeover-ato/) * [Magecart](https://www.imperva.com/learn/application-security/magecart/) * [Cyber Warfare](https://www.imperva.com/learn/application-security/cyber-warfare/) * [Reflected cross site scripting (XSS) attacks](https://www.imperva.com/learn/application-security/reflected-xss-attacks/) |
| Distributed Denial-of-Service (DDoS) | * [Distributed Denial of Service (DDoS)](https://www.imperva.com/learn/ddos/denial-of-service/) * [DDoS Attacks](https://www.imperva.com/learn/ddos/ddos-attacks/) * [How To Stop DDoS Attacks](https://www.imperva.com/learn/ddos/how-to-stop-ddos-attacks/) * [Prevent DDoS Attacks](https://www.imperva.com/learn/ddos/how-to-prevent-ddos-attacks/) |
| Bots | * [Bots](https://www.imperva.com/learn/application-security/what-are-bots/) * [Bot Management](https://www.imperva.com/learn/application-security/bot-management/) * [Web Scraping](https://www.imperva.com/learn/application-security/web-scraping-attack/) * [Sneaker Bot](https://www.imperva.com/learn/application-security/sneaker-bot/) |
| Data | * [Data Warehouse](https://www.imperva.com/learn/data-security/data-warehouse/) * [Data Lakes](https://www.imperva.com/learn/data-security/data-lakes/) * [Data Breach](https://www.imperva.com/learn/data-security/data-breach/) * [Data Discovery](https://www.imperva.com/learn/data-security/data-discovery/) |
| Cloud security | * [Cloud Security](https://www.imperva.com/learn/application-security/cloud-security/) * [Container Security](https://www.imperva.com/learn/application-security/container-security/) * [Cloud Governance](https://www.imperva.com/learn/data-security/cloud-governance/) * [Cloud Migration](https://www.imperva.com/learn/application-security/cloud-migration/) |
| Testing and assessment | * [API Testing](https://www.imperva.com/learn/application-security/api-testing/) * [Penetration Testing](https://www.imperva.com/learn/application-security/penetration-testing/) * [Vulnerability Assessment](https://www.imperva.com/learn/application-security/vulnerability-assessment/) * [Application Security Testing](https://www.imperva.com/learn/application-security/application-security-testing/) |
| Regulation and compliance | * [General Data Protection Regulation (GDPR) Compliance](https://www.imperva.com/learn/data-security/general-data-protection-regulation-gdpr/) * [HIPAA Health Insurance Portability and Accountability Act](https://www.imperva.com/learn/data-security/hipaa-privacy-rule/) * [PCI DSS Certification](https://www.imperva.com/learn/data-security/pci-dss-certification/) * [SOC 2 Compliance](https://www.imperva.com/learn/data-security/soc-2-compliance/) |
| Network management | * [Network Security](https://www.imperva.com/learn/application-security/network-security/) * [Network Monitoring](https://www.imperva.com/learn/availability/network-monitoring/) * [CDN and SSL/TLS](https://www.imperva.com/learn/performance/cdn-and-ssl-tls/) * [DNS Protection](https://www.imperva.com/learn/application-security/dns-protection/) |
| Incident response and management | * [Incident Response](https://www.imperva.com/learn/application-security/define-security-incident-response/) * [Alert Fatigue](https://www.imperva.com/learn/data-security/alert-fatigue/) * [Extended Detection and Response (XDR)](https://www.imperva.com/learn/data-security/extended-detection-and-response-xdr/) |
| Authentication and access control | * [User and Entity Behavior Analytics (UEBA)](https://www.imperva.com/learn/data-security/ueba-user-and-entity-behavior-analytics/) * [Access Control List (ACL)](https://www.imperva.com/learn/data-security/access-control-list-acl/) * [Privileged User Monitoring](https://www.imperva.com/learn/data-security/privileged-user-monitoring/) * [Identity and Access Management (IAM)](https://www.imperva.com/learn/data-security/iam-identity-and-access-management/) |
| Cybersecurity solutions and tools | * [Cyber Security Solutions](https://www.imperva.com/learn/application-security/cyber-security-solutions/) * [Security information and event management (SIEM)](https://www.imperva.com/learn/application-security/siem/) * [Intrusion detection and intrusion prevention](https://www.imperva.com/learn/application-security/intrusion-detection-prevention/) * [SAST, DAST, IAST and RASP](https://www.imperva.com/learn/application-security/sast-iast-dast/) |
| Industries | * [Healthcare Cybersecurity](https://www.imperva.com/learn/data-security/healthcare-cybersecurity/) * [Financial Services Cybersecurity](https://www.imperva.com/learn/data-security/financial-services-cybersecurity/) |

**Activity 5:** students choose one of the concepts referred to in the Software Engineering 11–12 Syllabus and create a single-slide summary, including a definition, characteristics and an example. This summary contributes to a class slide deck to be developed into a reference for their client: ‘The Unsecure PWA Company’ and a revision resource this focus area.

**Topics**

* [Data protection](https://www.imperva.com/learn/data-security/data-protection/)
* [Cyber attacks](https://www.imperva.com/learn/application-security/cyber-security-threats/)
* [Static application security testing (SAST)](https://www.imperva.com/learn/application-security/sast-iast-dast/)
* [Dynamic application security testing (DAST)](https://www.imperva.com/learn/application-security/sast-iast-dast/)
* [Vulnerability assessment](https://www.imperva.com/learn/application-security/vulnerability-assessment/)
* [Penetration testing](https://www.imperva.com/learn/application-security/penetration-testing/)
* [API security](https://www.imperva.com/learn/application-security/api-security/)
* [Cross-site scripting (XSS)](https://brightsec.com/blog/xss/)

Students revisit [Cross-Site Scripting](https://www.hacksplaining.com/lessons/xss-stored/start) by Hacksplaining and complete the interactive lesson.

### Describe the benefits of developing secure software: data protection

#### Including data protection

**Activity 6**: students describe the benefits of developing secure software to protect data to their client: ‘The Unsecure PWA Company’ by completing the following table:

Table 6 – the benefits of developing secure software to protect data

|  |  |
| --- | --- |
| Element | Benefit |
| Confidentiality | Secure software ensures that sensitive data is protected from unauthorised access. This helps maintain the confidentiality of personal, financial and other sensitive information. |
| Integrity | By implementing security measures, software developers can prevent unauthorised modification of data. This ensures that data remains accurate and reliable. |
| Availability | Secure software helps maintain the availability of data by protecting against threats such as denial-of-service attacks. Users can access the data they need without disruptions. |
| Compliance with regulations | Developing secure software helps organisations comply with data protection regulations and standards such as General Data Protection Regulation (GDPR), Health Insurance Portability and Accountability Act (HIPAA) or Payment Card Industry Data Security Standard (PCI DSS). Compliance with these regulations is essential for avoiding legal consequences and maintaining trust with customers. |
| 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 prioritises their data protection, leading to increased customer loyalty and satisfaction. |
| Business continuity | Secure software contributes to the overall resilience of a business by reducing the likelihood of data loss or service disruptions due to security incidents. This ensures continuity of operations and minimises potential financial and operational impacts. |

Developing secure software is essential for protecting data, maintaining compliance, building trust with users and safeguarding against cyber threats, ultimately contributing to the overall success and longevity of an organisation. Students consider these factors when advising ‘The Unsecure PWA Company’ in their security report.

**Teacher note**: each of these activities should be delivered using the 5 Whys questioning technique to elicit better responses from students and improve their examination technique. An example is provided

**Activity 7**: students improve their examination technique by applying the 5 Whys technique.

**Applying the 5 Whys technique on the importance of data confidentiality**

1. Why is data confidentiality important?

|  |
| --- |
| Data confidentiality is important to protect sensitive information from unauthorised access. |

1. Why is it crucial to protect sensitive information from unauthorised access?

|  |
| --- |
| Unauthorised access to sensitive information can lead to data breaches, identity theft, financial losses, and privacy violations. |

1. Why are data breaches, identity theft, and privacy violations concerning?

|  |
| --- |
| Data breaches can result in the exposure of personal information, compromising individuals' privacy and potentially causing reputational damage to organisations. |

1. Why is maintaining individuals' privacy and protecting organisational reputation important?

|  |
| --- |
| Safeguarding privacy and protecting reputation are essential for building trust with customers, stakeholders, and the public. |

1. Why is building trust with customers and stakeholders crucial for organisations?

|  |
| --- |
| Building trust fosters strong relationships, enhances brand loyalty, and ultimately contributes to the long-term success and sustainability of organisations. |

### Describe the benefits of developing secure software

#### Including minimising cyber attacks and vulnerabilities

**Activity 8**: [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645). Students describe to their client: ‘The Unsecure PWA Company’ the benefits of developing secure software to minimise cyber attacks and vulnerabilities by completing the table below.

Table 7 – the benefits of developing secure software to minimise cyber attacks and vulnerabilities

|  |  |
| --- | --- |
| Element | Benefits |
| Prevent data breaches | Secure software helps prevent data breaches by implementing security measures that protect sensitive information from unauthorised access to reduce the risk of data exposure and financial loss. |
| Mitigate against cyber threats | Minimising vulnerabilities means that secure software 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. |
| 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. |
| Business continuity | By reducing the risk of downtime, financial loss, and reputational damage, secure software contributes to business continuity and operational stability. |
| Cost savings | Addressing security vulnerabilities during the software development lifecycle 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 remediation efforts and damage control. |
| Competitive advantage | Developing secure 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. |

Developing secure software to minimise cyber attacks and vulnerabilities not only protects data and systems but also enhances trust, ensures regulatory compliance, improves business continuity, saves costs, and provides a competitive edge in the market.

Students consider these factors when advising ‘The Unsecure PWA Company’ in their security report and fixing vulnerabilities in their [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA).

**Homework activity**: students conduct the ‘[How cyber secure are you?](https://www.cyber.gov.au/learn-basics)’ quiz.

They report their level of secureness in class and the skills that need developing.

**Activity 9**: students read [4 Lessons from Software Security Case Studies](https://www.blueoptima.com/4-lessons-from-software-security-case-studies/).

Students create 4 teams. Each team researches and discusses one of the problems and the lessons to learned. Students report back to the class to complete the table below.

Table 8 – vulnerability case studies

|  |  |  |
| --- | --- | --- |
| Vulnerability | Description | Lessons learned |
| Equifax (2017) | A vulnerability in Apache Struts allowed hackers to execute malicious code, leading to a breach affecting 143 million Americans. | Ensure timely application of patches and regular monitoring of software estate. |
| Veeam (2018) | A server containing millions of customer records was publicly accessible due to lack of password protection. | Regularly check and update database security to avoid external alerts and maintain customer confidence. |
| Facebook (2021) | A misconfiguration in the contact importer allowed hackers to access the data of 533 million users. | Employ knowledgeable staff to avoid misconfigurations, especially in cloud-based systems. |
| Log4j | A vulnerability in the logging tool Log4j allowed attackers to access and control systems. | Treat third-party resources with caution and frequently update open-source software. |

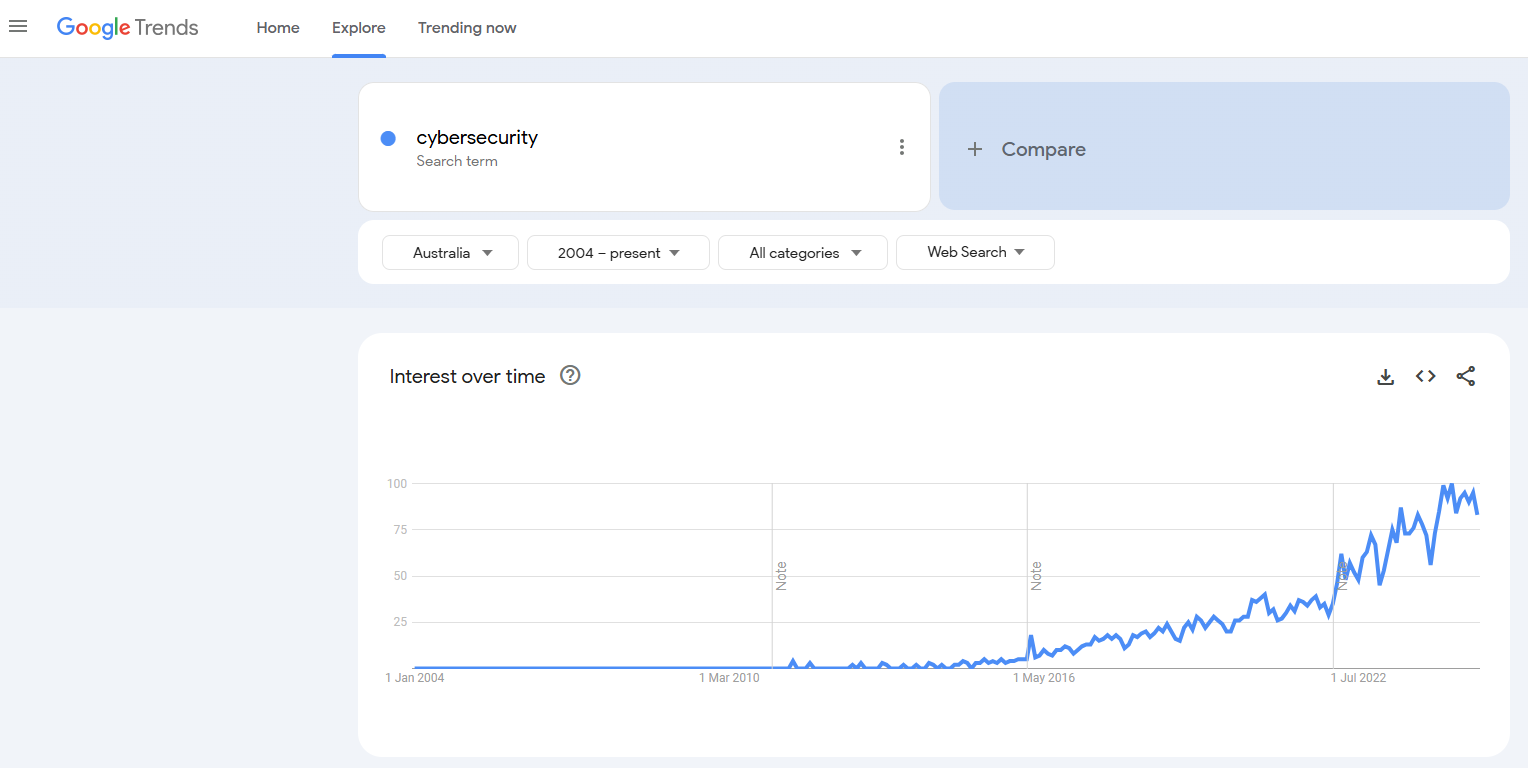
**Activity 10**: individual students, teams or class maintain a news board, journal, or folder of hyperlinks to news stories from cybersecurity incidents to refer to in class as case studies.

Students visit:

* [Australian Signals Directorate](https://www.cyber.gov.au/) homepage
* [13 Biggest Data Breaches in Australia [Updated 2024]](https://www.upguard.com/blog/biggest-data-breaches-australia) blog
* the [Information Age](https://ia.acs.org.au/) website daily to contribute to the news board.

**Activity 11:** students use [Google Trends](https://trends.google.com/home?hl=en-US) to monitor the peaks and troughs of interest in cybersecurity and speculate upon events that may have given rise to these spikes.

Figure 2 – Google Trends: cybersecurity

****

Screenshot from [Google Trends](https://trends.google.com/trends/explore?date=all&geo=AU&q=cybersecurity&hl=en-GB), accessed 21 November 2024.

**Activity 12**: students describe to their client, ‘The Unsecure PWA Company’ the issues, description and mitigation strategies involved in ensuring the security and protection of sensitive data from unauthorised access, tampering, or loss. They complete the following table for their security report.

Table 9 – data protection and mitigation strategies

|  |  |  |
| --- | --- | --- |
| Issue | Description | Mitigation strategy |
| Confidentiality | Secure software ensures that sensitive data remains confidential and only accessible to authorised users. | Implement encryption techniques and access controls to restrict access to sensitive data to authorised users. |
| Integrity | Secure software protects data integrity by preventing unauthorised modifications or tampering of data. | Conduct data validation and integrity checks to ensure data remains accurate and reliable. |
| Availability | Secure software ensures the availability of data and services to legitimate users while mitigating downtime. | Implement robust backup and recovery mechanisms to maintain data availability in case of disruptions. |

**Activity 13**: students complete the table below on minimising cyber attacks and vulnerabilities.

Table 10 – minimising cyber attacks and vulnerabilities

|  |  |  |
| --- | --- | --- |
| Issue | Description | Mitigation strategy |
| Eventing data breaches | Secure software development practises help prevent data breaches that expose sensitive information, lead to financial losses, and damage reputation. | Implement security controls like encryption, authentication, and access controls to reduce the risk of data breaches. |
| Mitigating malware and ransomware attacks | Secure software incorporates measures to detect and prevent malware and ransomware attacks that compromise system integrity and disrupt operations. | Conduct regular security updates, malware scans, and use intrusion detection systems to mitigate malware and ransomware threats. |
| Addressing software vulnerabilities | Secure software development involves identifying and addressing vulnerabilities through secure coding practises, security assessments, and vulnerability management. | Proactively identify and address software vulnerabilities to reduce the likelihood of exploitation by cyber attackers. |

Students advise their client, ‘The Unsecure PWA Company’ in the security report and by addressing vulnerabilities in the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA).

### Interpret and apply fundamental software development steps to develop secure code

#### Including:

* requirements definition
* determining specifications
* design
* development
* integration
* testing and debugging
* installation
* maintenance.

The task for this Secure software architecture focus area is write a security report for ‘The Unsecure PWA Company’ and provide software solutions. A scaffolded security report template can be found in the [Appendix](#_Appendix_1_:) of this document. Students will interpret and apply fundamental software steps while developing this secure code and addressing vulnerabilities in the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA).

**Activity 14:** requirements definition

Students complete the requirements definition of their security report for ‘The Unsecure PWA Company’ client scaffolded in the [Appendix](#_Appendix_1_:) of this document.

For revision and examination preparations, teachers **cut up** the following table to separate and distribute the requirements and the description randomly to student teams.

Student teams reassemble the table by matching the requirement with the correct description.

Table 11 – requirement definitions

|  |  |
| --- | --- |
| Requirement | Description |
| Objective | * Enhance the security of an existing unsecured progressive web app (PWA) by implementing a range of security measures and algorithms. Secure the login page, incorporating two-factor authentication (2FA) using Google Authenticator, [hashing passwords (4:05)](https://www.youtube.com/watch?v=deNIgVyDyJY) in the database and sanitising input fields to prevent common vulnerabilities. |
| Tools and environment | * Visual Studio Code (VS Code) with, SQLite, and Python Flask, Bcrypt * Access to Google Authenticator for implementing 2FA |
| Functional requirements | * Develop a login page for the PWA that allows users to input usernames and passwords * Implement a secure password hashing algorithm (for example, [Bcrypt](https://marketplace.visualstudio.com/items?itemName=aabegg.vscode-bcrypt)) to store passwords in the database securely * Sanitise input fields, such as the username field, to prevent SQL injection and cross-site scripting attacks * Integrate Google Authenticator for two-factor authentication to enhance login security * Conduct penetration testing before and after implementing security measures to identify and address vulnerabilities |
| Non-functional requirements | * **Security:** enhance the overall security of the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) by implementing industry-standard security measures and best practices * **Usability:** ensure that the security enhancements do not compromise the usability and user experience of the PWA * **Performance:** implement security measures in a way that does not significantly impact the performance of the PWA * **Scalability:** ensure that the security enhancements are scalable as the PWA grows in terms of users and features |
| Testing and validation | * Conduct [penetration testing](https://www.codingdojo.com/blog/what-is-penetration-testing) using tools like [OWASP](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbVZGX0dhRDBBZHJQYUtfRGVNU0FxRWU3TWFWQXxBQ3Jtc0tsMzFXTTMwNnB5R0o3bHdhSEdRWktheFNhQ0JfeVdjclRMZUp0NWZ0dmpnR1pOYWhGWldBSGFFOW82QkZ1Vm9xVDRBamU3c0ItckRHR2pqb282d3d3LUdZWTR0LW84M0h5NEhpUExfZnJ4Q3lmeV9aSQ&q=https%3A%2F%2Fowasp.org%2F&v=cUTeazTFKq0), ZAP or Burp Suite to identify security vulnerabilities * Validate the effectiveness of the implemented security measures through post-implementation pen testing * Address and resolve any vulnerabilities discovered during testing |
| Documentation and presentation | * Document the security enhancements made to the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA), including details of the implemented security measures and algorithms * Prepare a presentation to highlight the security improvements, including before-and-after comparisons and pen testing results |
| Timeline | * Establish a timeline for completing each phase of the project, including development, testing, validation, and documentation * Ensure that milestones are set for implementing security measures, conducting pen testing, and finalising documentation and presentation |

**Activity 15**: students watch [What is Penetration Testing? (2:52)](https://www.youtube.com/watch?v=q2t91jLmh3k) and draft an email to advise their ‘The Unsecure PWA Company’ client on the need to conduct penetration testing on the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA).

**Email to client**

|  |
| --- |
| **Subject: Protecting your PWA: The critical need for penetration testing**  Dear < >,  I hope this message finds you well! As the digital landscape continues to evolve, ensuring the security of your progressive web application (PWA) is more crucial than ever.  Given that your application is currently classified as ‘unsecure’, I want to highlight the importance of conducting penetration testing.  **Why penetration testing is essential**   1. Identify vulnerabilities: penetration testing simulates real-world attacks to uncover vulnerabilities in your PWA. This proactive approach helps you understand where your application may be exposed to threats and weaknesses. 2. Protect sensitive data: if your PWA handles user data, such as personal information or payment details, a security breach could lead to severe consequences, including data theft and loss of customer trust. Penetration testing helps safeguard this information by identifying potential entry points for attackers. 3. Mitigate risks: by conducting penetration testing, you can prioritise and address vulnerabilities before they are exploited. This risk mitigation is essential for protecting your reputation and ensuring compliance with industry regulations. 4. Enhance customer trust: demonstrating that your company takes security seriously through regular testing can enhance your brand's reputation. Customers are more likely to engage with a platform that prioritises their security and privacy. 5. Cost-effective security: addressing vulnerabilities after a breach can be significantly more expensive than identifying and fixing them beforehand. Investing in penetration testing is a cost-effective strategy to prevent potential financial losses associated with security incidents. 6. Stay ahead of threats: cyber threats are continuously evolving. Regular penetration testing helps you stay one step ahead of attackers by adapting your security measures in response to new vulnerabilities and attack vectors.   **Next steps**  I recommend scheduling a consultation to discuss how we can tailor a penetration testing strategy specifically for your PWA. Our team of experts can provide insights and solutions that align with your security goals and business objectives.  Let’s work together to fortify your application and ensure the safety of your users. Please feel free to reach out at your convenience to discuss this critical investment in your company’s future.  Best regards,  [Your Name]  [Your Position]  [Your Company]  [Contact Information] |

**Activity 16**: students visit [SQL Injection](https://www.w3schools.com/sql/sql_injection.asp) by W3Schools and answer the following question: What is an SQL injection?

|  |
| --- |
| **SQL injection**:  This is where a threat actor enters data into an input box that includes SQL statements. It is one of the most common web hacking techniques. Common attacks are:   * Incorporating a statement that forces the login to return true, usually by adding an OR statement so the literal comparison returns true. * Incorporating a query that adds a user the database usually using an INSERT to an obvious table name. * Incorporating a query that inserts malicious code to a webpage usually causing a visitor to the page to automatically download a malicious file. |

**Activity 17**: students watch [SQL Injection Attacks – Explained in 5 Minutes (5:15)](https://www.youtube.com/watch?v=FHCTfA9cCXs). They visit [SQL Injection](https://www.w3schools.com/sql/sql_injection.asp) by W3Schools and answer the following question: How does an SQL injection work?

|  |
| --- |
| SQL injection occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database. SQL injects are exposed because of 2 algorithm design vulnerabilities:   1. A failure to sanitise data on entry to the program. Developers should check that inputs do not contain common scripting characters <>[]{}= etc before processing the data. 2. Using string concatenation/interpolation before literal comparisons like == because they are easy to write SQL that will force them to return true. Rather, software engineers should prefer query parameters where no literal comparison occurs.   SQL injects are more likely to be successful where obvious table names have been used. A table used to store users named 12\_tl-u5ers will be much harder to inject. |

**Activity 18**: students complete the online interactive lessons on [SQL Injection.](https://www.hacksplaining.com/lessons/sql-injection/start)

**Activity 19**: Students watch [Hashing passwords with Python and Bcrypt (15:43)](https://www.youtube.com/watch?v=hNa05wr0DSA). They visit [Hashing Passwords in Python with BCrypt](https://www.geeksforgeeks.org/hashing-passwords-in-python-with-bcrypt/)and answer the following question: What is hashing?

|  |
| --- |
| Hashing is a process of converting one string to another using a hash function. The process is one way so you cannot return the hash to its original string, so unlike some encryption it can’t be decrypted. This means if the SQL database is exposed, no passwords are exposed. The s alt also prevents password database matching. Bcrypt uses strong cryptography to hash and salt passwords to make encryption stronger making brute-force attacks slower and harder. |

**Activity 20**: determining specifications. Students continue working on with their security report for ‘The Unsecure PWA Company’ scaffolded in the [Appendix](#_Appendix_1_–) of this document.

For revision and examination preparations, teachers **cut up** the following table to separate and distribute the specifications and the description randomly to student teams.

Student teams reassemble the table by matching the requirement with the correct description

Table 12 – security specifications

|  |  |
| --- | --- |
| Specification | Description |
| Login page development | * Develop a login page for the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) that includes input fields for username and password * Implement a secure authentication process that validates user credentials before granting access to the PWA |
| Password hashing | * Utilise a strong hashing algorithm (for example, Bcrypt) to securely hash and store passwords in the database. * Ensure that hashed passwords are not reversible and are protected against brute-force attacks |
| Input field sanitisation | * Implement input field sanitisation to prevent common vulnerabilities such as SQL injection and cross-site scripting (XSS) * Validate and sanitise user inputs, especially in the username field, to mitigate security risks |
| Two-factor authentication (2FA) | * Integrate Google Authenticator for two-factor authentication to enhance login security * Enable users to set up 2FA for their accounts, providing an additional layer of security beyond passwords |
| Penetration testing | * Conduct initial pen testing to identify security vulnerabilities in the unsecured PWA * Address and resolve any vulnerabilities discovered during pen testing before implementing security measures * Perform post-implementation pen testing to validate the effectiveness of the security enhancements |
| Documentation and presentation | * Document the security enhancements made to the PWA, including the implemented security measures and algorithms * Prepare a presentation highlighting the before-and-after security improvements, pen testing results, and the rationale behind the security measures |
| Security tools and technologies | * Use Visual Studio Code with, SQLite3, and Flask extensions for web development and database management * Implement industry-standard security tools and technologies to enhance the security of the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA) |
| Timeline and milestones | * Establish a timeline with clear milestones for each phase of the project, including development, testing, validation and documentation * Ensure that deadlines are set for implementing security measures, conducting pen testing, and finalising documentation and presentation |
| Scalability and performance | * Ensure that the security enhancements do not significantly impact the performance of the PWA * Design security measures to be scalable as the PWA grows in terms of users and features |
| Compliance and best practices | * Ensure that the security enhancements align with industry best practices and compliance requirements related to data protection and privacy * Follow coding standards and security guidelines to maintain the integrity and security of the PWA |

**Activity 21**: design and development phases.

Students document the key design and development phases within their security report for ‘The Unsecure PWA Company’ scaffolded in the [Appendix](#_Appendix_1_:) of this document.

For revision and examination preparations, teachers **cut up** the following table to separate and distribute the key design and development phases and the description randomly to student teams. Student teams reassemble the table by matching the requirement with the correct description

Table 13 – design and development phases

|  |  |
| --- | --- |
| Phases | Description |
| Requirement analysis | * Understand the current state of the unsecured PWA and identify security vulnerabilities that need to be addressed * Define the specific security requirements, such as implementing password hashing, input sanitisation, two-factor authentication, and pen testing |
| Design phase | * Design the architecture and security features of the PWA, including the login page, password hashing mechanism, input sanitisation process, and 2FA integration * Create wireframes and sketches to visualise the layout and user interface elements of the secure PWA |
| Development phase | * Develop the login page with secure authentication mechanisms using Python Flask and SQLite for database management * Implement secure password hashing using algorithms like Bcrypt to store passwords securely in the database * Integrate input field sanitisation to prevent SQL injection and XSS attacks * Configure Google Authenticator for two-factor authentication and ensure seamless integration with the login process |
| Testing phase | * Conduct initial pen testing to identify vulnerabilities in the unsecured PWA and address critical security issues * Test the implementation of password hashing, input sanitisation, and 2FA for functionality and security compliance * Perform user acceptance testing to ensure a seamless user experience while maintaining robust security measures |
| Security assessment and validation | * Validate the effectiveness of the security enhancements through post-implementation pen testing and security assessments * Address any vulnerabilities discovered during testing and ensure that all security measures are functioning as intended |
| Documentation and presentation | * Document the security design and development process, including detailed specifications, implementation steps, and security measures * Prepare a presentation highlighting the security enhancements, pen testing results, and the rationale behind the chosen security features |
| Deployment and monitoring | * Deploy the secure PWA to a web server or hosting environment, ensuring that all security measures are operational * Implement monitoring tools to track security metrics, detect potential security incidents, and ensure continuous security of the PWA |

**Activity 22:** students investigate [ZAP to perform security testing](https://www.zaproxy.org/getting-started/).

Students watch [OWASP Zap Tutorial (7:34)](https://www.youtube.com/watch?v=_VpFaqF0EcI).

**Teacher note:** ZAP is only to be used on sites owned by the students.

Also note the // Disclaimer // Hacking without permission is illegal. This channel is strictly educational for learning about cybersecurity in the areas of ethical hacking and penetration testing so that we can protect ourselves against the real hackers.

**Activity 23**: integration: [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645).

Student teams contribute to the security report for the client ‘The Unsecure PWA Company’ on how this project could be integrated into an existing software system. They are provided with an incomplete table listing the steps and provide detailed descriptions.

Table 14 – steps to integration (sample answers)

|  |  |
| --- | --- |
| Steps | Description |
| Assessment of existing system | * Assess the existing software system to understand its architecture, components, security vulnerabilities and the impact of integrating new security measures * Identify areas of the system that require enhanced security, such as user authentication, data storage and input validation |
| Requirements alignment | * Align the security requirements and design specifications of the PWA enhancement project with the existing software system's architecture and security framework * Ensure that the new security measures seamlessly integrate with the existing system's functionalities and do not conflict with any existing security protocols |
| Modular development approach | * Adopt a modular development approach to implement the security enhancements in components or modules that can be integrated into the existing system without causing disruptions * Develop the new security features as separate modules that can be easily integrated and configured within the existing system |
| API integration | * Utilise APIs and web services to integrate the new security features, such as password hashing, input sanitisation and 2FA, with the existing software system * Ensure that the APIs are well-documented, secure and compatible with the existing system's communication protocols |
| Database integration | * Integrate the secure password hashing mechanism with the existing database system to store passwords securely * Implement input field sanitisation processes within the database queries to prevent SQL injection attacks |
| User interface integration | * Update the user interface of the existing software system to incorporate the new security features, such as the login page with 2FA functionality * Ensure that the user interface changes are intuitive and user-friendly to maintain a seamless user experience |
| Testing and validation | * Conduct thorough testing of the integrated security features within the existing software system to verify functionality, security compliance, and interoperability * Perform regression testing to ensure that the integration of the new security measures does not impact the overall performance and stability of the system |
| Deployment and monitoring | * Deploy the integrated security enhancements into the existing software system in a staged manner to minimise disruption and ensure a smooth transition * Implement monitoring tools to track the performance, security metrics, and user feedback after the integration to address any potential issues promptly |

**Jigsaw Activity 24**: students form teams of 3. The teacher numbers each member of the team from 1 or 2 or 3. All 1’s form a group responsible for researching testing and debugging. All 2’s form a group responsible for researching installation and deployment. All 3’s form a group responsible for researching maintenance and monitoring. Each group brainstorms and researches their software development step to develop secure code.

Groups reform into teams (1, 2 and 3) to combine their findings and present to the client, ‘The Unsecure PWA Company’ on ‘How will this secure PWA be tested, debugged, installed and maintained?’ Students update their security report in the [Appendix](#_Appendix_1_:) of this document.

|  |
| --- |
| **Sample answers:**   1. Testing and debugging   **Testing strategies**   * Conduct unit testing to test individual components and functions, ensuring they work as intended. * Perform integration testing to verify that the security features integrate seamlessly with the existing software system. * Implement end-to-end testing to validate the entire flow of the secure login process, including password hashing, input sanitisation and 2FA. * Utilise automated testing tools to run regression tests and ensure that new security measures do not introduce any regressions.   **Debugging process**   * Use debugging tools in Visual Studio Code and Python to identify and resolve any coding errors or issues in the implementation of security features. * Employ logging and error handling mechanisms to track and troubleshoot any potential bugs or security vulnerabilities. * Collaborate with team members to review code, perform code reviews and address any identified issues efficiently. |
| 1. Installation and deployment   **Deployment plan**   * Develop a deployment plan that outlines the steps for deploying the secure PWA with enhanced security features into the production environment. * Ensure that all necessary configurations, dependencies and security measures are in place before deployment.   **Deployment process**   * Deploy the secure PWA to the hosting environment or web server using Git Bash or other deployment tools. * Conduct a pilot deployment to a test environment to validate the deployment process and ensure that the security features function correctly. |
| 1. Maintenance and monitoring:   **Security monitoring**   * Implement monitoring tools to continuously monitor the security metrics, performance and user activity of the secure PWA. * Set up alerts and notifications to detect and respond to any security incidents or anomalies in real-time.   **Regular updates and patches**   * Schedule regular updates and patches to address security vulnerabilities, enhance features and ensure the ongoing security of the PWA. * Maintain documentation of updates and changes made to the system for future reference and auditing purposes.   **User training and support**   * Provide user training and support to educate users on the new security features, such as 2FA, and address any questions or concerns related to security practices. * Maintain communication channels for users to report security incidents or seek assistance with security-related issues. |

### Describe how the capabilities and experience of end users influence the secure design features of software

‘The Unsecure PWA Company’ is the fictious client for this project. This provides a role-playing opportunity where the teacher or critical friends and peers take on the role of members of the company scenario as well as purchasers of the software. Alternatively commercially available software could be analysed to determine the capabilities and experience of the end users and how this has influenced the secure design features.

**Activity 25**: [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645). Suggest ways in which the capabilities and experience of end users can influence the secure design features of the software project.

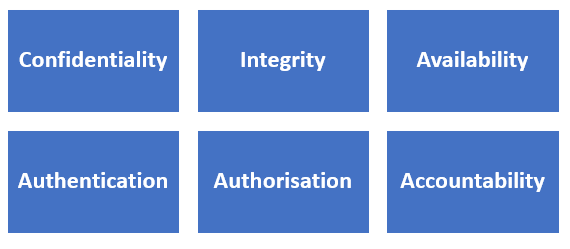
|  |
| --- |
| **User-friendly authentication:** considering the capabilities and experience of end users can help in designing authentication mechanisms that are secure yet easy to use. For instance, offering options for two-factor authentication for users who may have higher security needs, while providing simpler authentication methods for less tech-savvy users.  **Role-based access control:** understanding the roles and responsibilities of different users can help in implementing role-based access control within the PWA. This ensures that users only have access to the features and data that are relevant to their roles, enhancing security by limiting potential attack surfaces.  **Training and awareness:** tailoring security features based on the experience of end users can also involve providing training and educational resources on secure practices within the PWA. This can help less experienced users understand the importance of security measures and how to use them effectively.  **User feedback and testing:** involving end users in the testing phase can provide valuable insights into how secure design features are perceived and used in practice. This feedback can help in refining security features to better align with the capabilities and preferences of the end users.  **Simplified security controls:** for less experienced users, it may be beneficial to simplify security controls and provide clear guidance on how to use them effectively. This can prevent security features from becoming barriers to usability and ensure that users can easily adopt secure practices. |

## Developing secure code

### Explore fundamental software design security concepts when developing programming code

**Including:** confidentiality**,** integrity**,** availability**,** authentication**,** authorisation**,** accountability

Figure 3 – fundamental software design security concepts



**Activity 26**: student teams develop a brief presentation for ‘The Unsecure PWA Company’ (client) on the fundamental software design security concepts when developing programming code. This presentation should include ‘What it is, why is it important? And how to ensure it?’ This provides a role-playing opportunity where the teacher or critical friends and peers take on the roles of members of the company scenario, as well as purchasers of the software.

Students are provided a blank table to complete. A sample response is provided below.

Table 15 – security concepts

|  |  |  |  |
| --- | --- | --- | --- |
| Security concept | What it is | Why is it important | How to ensure it |
| Confidentiality | Ensures that sensitive information is encrypted and securely stored to prevent unauthorised access. | Protects user credentials and personal data from being accessed by unauthorised parties, maintaining privacy and trust in the system. | * Encrypt sensitive data at rest and in transit * Implement secure communication protocols (for example, HTTPS) |
| Integrity | Verifies data integrity to prevent unauthorised tampering or alteration of data. | Ensures that data remains accurate, consistent, and trustworthy throughout its lifecycle. | * Implement checksums, digital signatures, and hash functions to detect and prevent data tampering * Use encryption to protect data integrity during transmission |
| Availability | Ensures that system services and resources are accessible to authorised users when needed. | Prevents downtime, ensures continuity of services, and maintains user satisfaction. | * Design with redundancy, load balancing, and disaster recovery mechanisms * Regularly monitor system performance and address potential issues proactively |
| Authentication | Verifies the identity of users before granting access to the system. | Prevents unauthorised access and ensures that users are who they claim to be. | * Implement strong authentication mechanisms (passwords, 2FA, biometrics) and use secure protocols for user authentication |
| Authorisation | Enforces access control mechanisms to determine user permissions and restrict access based on roles. | Prevents unauthorised activities and protects sensitive data from being accessed by unauthorised users. | * Implement role-based access control (RBAC) or attribute-based access control (ABAC) * Regularly review and update user permissions based on roles and responsibilities |
| Accountability | Implements logging and auditing mechanisms to track user activities and security events. | Establishes traceability and responsibility in case of security incidents or breaches. | * Maintain logs of user actions, access attempts, and security events * Regularly review audit logs and investigate any suspicious activities |

### Apply security features incorporated into software including data protection, security, privacy and regulatory compliance

**Activity 27**: students ‘tick off’ each of the features that are appropriate to apply in their project.

Table 16 – security features incorporated into software and their applications

|  |  |
| --- | --- |
| Security feature | Application |
| Data protection | Implement secure password hashing using strong algorithms like Bcrypt to protect user passwords stored in the database.  Encrypt sensitive data at rest and in transit using encryption protocols (for example, TLS) to prevent unauthorised access to data.  Ensure data minimisation by collecting only necessary user information and implementing data retention policies to limit data storage. |
| Security measures | Implement input validation and sanitisation techniques to prevent common vulnerabilities like SQL injection and cross-site scripting (XSS).  Use parameterised queries and prepared statements to protect against SQL injection attacks on the database.  Employ secure coding practices to mitigate security risks, such as input validation, output encoding, and error handling. |
| Privacy protection | Develop a privacy policy that clearly outlines how user data is collected, used and protected within the PWA.  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. |

**Activity 28:** students follow the ‘How-to guide’ to create a [Login page using Python, Flask and sqlite3 DB](https://gist.github.com/jironghuang/24e0577e58844882604c0013407bf606).

### Use and explain the contribution of cryptography and sandboxing to the ‘security by design’ approach in the development of software solutions

**Teacher note**: students may benefit from defining these terms prior to their explanations.

**Activity 29**: students research [What is Cryptography?](https://www.fortinet.com/resources/cyberglossary/what-is-cryptography) and provide a summary in the space below.

|  |
| --- |
| Cryptography is the process of encoding information so that only the intended recipient can read it. It has been used for thousands of years and is still used in modern technology like bank cards and passwords. Modern cryptography uses algorithms and ciphers to encrypt and decrypt information securely, with techniques like encryption keys. This practice, which combines computer science, engineering, and mathematics, ensures that messages are hidden from unauthorised access. Cryptography is crucial for securing data during communication and online transactions, using complex codes to protect privacy and prevent unauthorised access. |

**Activity 30**: students research [What is Sandboxing?](https://www.fortinet.com/resources/cyberglossary/what-is-sandboxing) and provide a summary in the space below.

|  |
| --- |
| Sandboxing is a security method that involves using an isolated environment to test and analyse code safely without affecting the main system. It is effective against new, unidentified threats called ‘zero-day threats’ that might not be caught by regular security filters. Sandboxing provides a higher level of protection by containing potentially harmful code within a secure environment for testing, ensuring that the rest of the system remains safe. |

**Activity 31**: students investigate and use ‘sandboxing’ tools.

|  |
| --- |
| Sandboxing enables practice running programs in a secure environment and testing their behaviour without affecting the actual system.  [VirtualBox](https://www.virtualbox.org/): a free and open-source virtualization software that allows users to create virtual machines. Students can set up virtual environments to test software and configurations safely.  [VMware:](https://www.vmware.com/) a popular virtualization software that students can use to create and run virtual machines for sandboxing purposes.  [Windows Sandbox:](https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-overview) a feature available in Windows 10 Pro and Enterprise editions that allows users to run applications in an isolated environment. It is a quick and easy way to practice sandboxing on a Windows machine.  [Cuckoo Sandbox](https://cuckoosandbox.org/) is an open-source automated malware analysis system that can be used by students to analyse and sandbox potentially malicious software.  Watch: [Google Chrome & Security: Sandboxing (2:27)](https://www.youtube.com/watch?v=29e0CtgXZSI) and explore how Chrome has implemented sandboxing to its security design features.  Browser-based sandboxes where users can practice running and testing code in a safe virtual environment include: [JSFiddle](https://jsfiddle.net/), [CodePen](https://codepen.io/), and [Replit](https://replit.com/).  These tools can help learn about sandboxing practices and experiment with different software and configurations without putting systems at risk. |

**Activity 32**: students visit the [Australian Signals Directorate: Secure-by-Design](https://www.cyber.gov.au/resources-business-and-government/governance-and-user-education/secure-by-design) and [What is Security by Design?](https://www.wiz.io/academy/security-by-design) to complete the table that defines security by design (SbD), provides an analogy and explains why it is important.

|  |  |  |
| --- | --- | --- |
| Definition | Analogy | Importance |
| A software development approach that integrates security controls into software products from the design phase, making security a core element rather than an afterthought. | Like building a warehouse with unbreachable materials for valuable goods, SbD focuses on creating systems that are inherently secure, reducing the need for additional security measures. | * **Establishes secure products**: SbD results in secure software that can be deployed without additional costs for security features. * **Moves away from vulnerable by design (VbD)**: in the past, software was shipped without built-in security, leading to vulnerabilities and the need for costly add-ons. SbD shifts the responsibility to both vendors and users, enhancing software security and reducing the attack surface. * **Reduces patching efforts**: proactively secure products require fewer and less critical patches, making the process of applying security fixes more efficient, cost-effective, and less error prone. |

**Activity 33**: teachers **cut up** the table on the following page and distribute randomly to students as a matching exercise

Table 17 – cryptography and sandboxing

|  |  |  |  |
| --- | --- | --- | --- |
| Contribution | Description | Explanation | Example |
| Cryptography | The practice of secure communication using codes and ciphers to protect information. It contributes to security by providing techniques for ensuring data confidentiality, integrity and authenticity. | Enables developers to implement strong encryption algorithms to protect sensitive data within the software. By encrypting data at rest and in transit, cryptographic techniques ensure that information is secure even if unauthorised access occurs. | Implementing end-to-end encryption in a messaging application ensures that messages exchanged between users are securely encrypted and decrypted only by the intended recipients, safeguarding communication from eavesdropping and tampering. |
| Sandboxing | Isolates applications or processes from the rest of the system, limiting the potential impact of security breaches or vulnerabilities. It provides a controlled space for running untrusted code or applications securely. | Helps prevent malicious actions from spreading to other parts of the software or system by isolating and containing potentially risky operations. | An environment to test third-party plugins or extensions in a web application helps ensure that these components do not have unintended access to sensitive data or system resources, enhancing the overall security posture of the application. |

### Use and explain the ‘privacy by design’ approach in the development of software solutions

#### Including: proactive not reactive approach, embed privacy into design, respect for user privacy

**Activity 34**: students visit the Office of the Australian Information Commissioner and summarise a definition of ‘[privacy by design](https://www.oaic.gov.au/privacy/privacy-guidance-for-organisations-and-government-agencies/privacy-impact-assessments/privacy-by-design)’.

|  |
| --- |
| ‘Privacy by design’ is a method of incorporating strong privacy practices into the initial design of technologies, business practices and physical infrastructures. This involves integrating privacy considerations into the design specifications and structure of new systems and processes from the beginning. It is recommended to address privacy risks proactively during the development phase rather than having to make changes later to address privacy concerns that may arise. |

**Activity 35**:students visit the [Information and Privacy Commission: Fact Sheet – Privacy by design](https://www.ipc.nsw.gov.au/fact-sheet-privacy-design), read and summariseproactive not reactive approaches,embedding privacy into designandrespect for user privacy:

|  |
| --- |
| Take a proactive approach, anticipating risks and preventing privacy-invasive events before they occur.  Embed privacy into the design of any systems, services, products, and business practices. You should ensure that privacy becomes one of the core functions of any system or service.  Keep the interest of individuals paramount in the design and implementation of any system or service. You can do this by offering strong privacy defaults and user-friendly options, as well as ensuring appropriate notice is given. |

### Test and evaluate the security and resilience of software by determining vulnerabilities, hardening systems, handling breaches, maintaining business continuity, and conducting disaster recovery

**Activity 36**: teacher-led discussionon the importance of software security, resilience, and the potential consequences of security breaches including an overview of common vulnerabilities in software and the impact of cyber attacks on businesses.

**Teacher note**: this is another opportunity to consolidate student understanding of vulnerabilities explicitly mentioned in the syllabus:

* **SQL injection**: attackers exploit SQL injection vulnerabilities to manipulate a database by inserting malicious SQL code into input fields. This can lead to unauthorised access, data leakage, and data manipulation.
* **Cross-site scripting (XSS)**: XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. This can result in the theft of sensitive information, session hijacking, and website defacement.
* **Cross-site request forgery (CSRF)**: CSRF attacks trick users into executing unwanted actions on a web application where they are authenticated. Attackers forge requests to perform actions on behalf of the victim, such as changing settings or making transactions.

**Activity 37**: students watch [How Equifax was Hacked (8:25)](https://www.youtube.com/watch?v=MKaNxE7pGWA). Theyresearch and analyse the case study in software vulnerabilities and discuss how they could have been prevented.

|  |
| --- |
| **Case study: Equifax data breach**  **Description of the incident**:   * In May 2017, the Apache Software Foundation released a security patch to address a critical vulnerability (CVE-2017-5638) in the Apache Struts framework, commonly used by Equifax in its web applications. * Equifax failed to apply the patch to its systems, leaving them vulnerable to exploitation. * In July 2017, hackers exploited the unpatched Apache Struts vulnerability to gain access to Equifax's systems and exfiltrate sensitive personal data, including names, Social Security numbers, birth dates, and credit card information.   **How the breach could have been prevented:**  **Timely patch management**: Equifax could have prevented the breach by promptly applying the available security patch to address the Apache Struts vulnerability. Establishing a robust patch management process that prioritises critical security updates is essential to mitigating known vulnerabilities.  **Vulnerability scanning and monitoring**: regular vulnerability scanning and monitoring of systems could have identified the unpatched Apache Struts vulnerability in Equifax's environment. Proactive detection of vulnerabilities allows organisations to address security weaknesses before they are exploited by threat actors.  **Security awareness and training**: Equifax could have enhanced security awareness among its employees and IT staff to ensure they understood the importance of applying security patches promptly. Training programs on cybersecurity best practices, including patch management, can help prevent lapses that lead to security incidents.  **Risk assessment and mitigation**: conducting thorough risk assessments to identify critical assets, potential vulnerabilities and associated risks can help organisations prioritise security measures. Equifax could have implemented a risk-based approach to security that focuses on protecting sensitive data and critical systems from known threats.  **Incident response planning**: Equifax could have developed and tested an effective incident response plan to quickly detect, contain, and mitigate security incidents. A well-prepared incident response team with defined roles and responsibilities could have minimised the impact of the breach and facilitated a swift response to the security incident. |

**Activity 38**: hands-on exercise.

Students use vulnerability scanning tools to identify security weaknesses in a sample application or system. These could include:

* [OpenVAS](https://openvas.org/) (Open Vulnerability Assessment Scanner): open-source vulnerability scanner for comprehensive vulnerability assessment in networks and web applications.
* [Nexpose](https://www.rapid7.com/products/nexpose/) Community Edition: free vulnerability scanner by Rapid7 for detecting vulnerabilities, misconfigurations, and exposures in systems and networks.
* [Nmap](https://nmap.org/) (Network Mapper): versatile network scanning tool for vulnerability scanning, network discovery, and security auditing to identify open ports, services, and potential vulnerabilities.
* [OWASP ZAP](https://www.zaproxy.org/) (Zed Attack Proxy): open-source web application security scanner by OWASP for detecting security vulnerabilities in web applications, including injection flaws, broken authentication, and insecure direct object references.

**Activity 39:** class visits the Australian Signals Directorate and the [Guidelines for System Hardening](https://www.cyber.gov.au/resources-business-and-government/essential-cyber-security/ism/cyber-security-guidelines/guidelines-system-hardening) identifies some of the best practices for securing software systems.

**Activity 40:** student teams visit the [Information Age](https://ia.acs.org.au/) website and find a recent data breach. Each team selects a different breach. They research this incident to develop a deep understanding of the event and using the outline below reenact it by creating a role-play. Students select or are appointed a role and research the nature of their work.

|  |
| --- |
| **Role-play: Security breach and business continuity simulation**  **Scenario**: You are part of the IT security team at TechSecure Inc, a fictional company that advises on security breaches that compromise sensitive data. As your team investigates and responds to the breach, you must also ensure that business operations continue seamlessly and that you have a plan in place for disaster recovery.  **Roles to research and play**:   * **IT Security Team Lead**: oversees the incident response, business continuity and disaster recovery efforts. * **Security Analysts (team members)**: investigate the breach and work on containment and eradication strategies. * **IT Support**: assists in technical solutions to maintain business operations and recover from the breach. * **Business Continuity Manager**: focuses on maintaining critical business functions during the incident. * **Disaster Recovery Specialist**: plans and implements strategies for recovering data and systems after the breach. |

|  |
| --- |
| **Role-play: Security breach and business continuity simulation**  Steps of incident response, business continuity, and disaster recovery:   1. **Detection and containment**:  * **Task**: detect suspicious activity, contain the breach and prevent further damage. * **Action**: investigate alerts, isolate affected systems and implement controls to stop the breach's spread.  1. **Eradication and recovery**:  * **Task**: remove the threat, recover data/systems and restore normal operations. * **Action**: eliminate malware, restore data from backups and ensure systems are secure before resuming operations.  1. **Business continuity and disaster recovery**:  * **Task**: maintain critical business functions during the incident and plan for post-incident recovery. * **Action**: activate business continuity plans, ensure essential services are operational and prepare for disaster recovery efforts. |

**Note**: teachers use explicit teaching throughout the procedure:

* **Briefing**: provide an overview of the security breach researched scenario, roles and incident response objectives, business continuity, and disaster recovery.
* **Simulation**: conduct the role-playing activity where students respond to the breach, maintain business functions, and plan for recovery.
* **Guided steps**: facilitate the incident response, business continuity and disaster recovery processes, ensuring coordination between roles.
* **Debriefing**: conclude the activity with a debriefing session to discuss the effectiveness of the response, challenges faced, and lessons learned in maintaining business operations during a security incident.

### Apply and evaluate strategies used by software developers to manage the security of programming code

#### Including: code review, static application security testing (SAST), dynamic application security testing (DAST), vulnerability assessment, penetration testing

**Jigsaw Activity 41**: each team of students is given a strategy to research and report back to their peers.

Table 18 – strategies to manage code security

|  |  |  |
| --- | --- | --- |
| Strategy | Description | Evaluation |
| Code review | Systematic examination of software code by peers to find and fix mistakes and improve code quality and security. | Effective in improving code quality and security, promotes collaboration and knowledge sharing within the team, helps identify and fix issues early in the development process. |
| Static application security testing (SAST) | Analyses source or compiled code to identify security vulnerabilities and weaknesses, such as SQL injection and cross-site scripting. | Proactive approach to identifying security vulnerabilities early in the development lifecycle, helps catch security issues before they are apparent in the running application, reduces the likelihood of security breaches and cost of fixing vulnerabilities later. |
| Dynamic application security testing (DAST) | Assesses application security while running by simulating attacks to identify vulnerabilities like authentication issues and input validation errors. | Provides a realistic assessment of an application's security strength, helps identify vulnerabilities that may not be apparent in static code analysis, offers insights into potential security weaknesses and how attackers could exploit them. |
| Vulnerability assessment | Identifies, quantifies, and prioritises security vulnerabilities in applications or systems to assess security risks and recommend mitigation strategies. | Helps understand security weaknesses in applications, prioritises remediation efforts based on vulnerability severity, essential for a comprehensive security strategy to protect against potential security threats. |
| Penetration testing | Simulated cyberattack to identify security weaknesses that could be exploited by malicious actors and assess the system's resilience against real-world threats. | Provides insights into the effectiveness of security controls, identifies critical security vulnerabilities that could lead to a breach, simulates real-world attack scenarios to test system defences under controlled conditions. |

### Design, develop and implement code using defensive data input handling practices, including input validation, sanitisation, and error handling

**Teacher note**: these defensive data input handling practices should be referred to during students' project development. Students should attempt to make their website or PWA more secure by adding these to their code.

**Activity 42**: students read the following descriptions of HTML, CSS and JavaScript.

**Design**

When designing HTML, CSS, and JavaScript code with defensive data input handling practices, create a structured layout using semantic HTML elements to enhance accessibility and maintainability. Use external CSS files to style the content consistently and improve the separation of concerns between HTML and CSS. In JavaScript, design modular functions to handle input validation, sanitisation, and error handling, promoting code reusability and maintainability.

**Development**

During development, implement input validation by leveraging HTML form validation attributes like required, pattern and maxlength to enforce data integrity on the client side. Use CSS to provide visual feedback on input validation errors, such as highlighting invalid fields. In JavaScript, validate form inputs programmatically using event listeners and functions that check input values against predefined criteria. Implement sanitisation functions to cleanse input data of potentially harmful characters or scripts to prevent XSS attacks.

**Implementation**

In the implementation phase, integrate input validation mechanisms into form submission events to validate user inputs before processing them. Use JavaScript to handle form submission events, validate input fields, and display error messages to users if validation fails. Implement error handling strategies to gracefully handle exceptions, log errors for debugging purposes, and provide informative feedback to users when errors occur during data processing.

**Activity 43**: students explore the code in [The Unsecure\_PWA](https://github.com/TempeHS/The_Unsecure_PWA/tree/main/.student_resources/secure_form_attributes#readme):

Students research [defensive data handling](https://github.com/TempeHS/The_Unsecure_PWA/tree/main/.student_resources/defensive_data_handling) and describe input validation, data sanitisation and exception handling.

|  |
| --- |
| Defensive data handling and input validation involves verifying that input data is valid before it is processed or stored in an application. It is best practice to perform this validation at the point of entry. Invalid data should be discarded, and users should receive feedback through the user interface. Frontend validation can be implemented using form field attributes, while backend validation can utilise regular expressions (regex) and binary selection methods.  Data sanitisation is where data is 'sanitised' or cleaned for processing or storing. This is the process of replacing any potentially malicious characters with non-processing codes so the text will render as expected, but no processing will occur.  Exception handling is essential in defensive data handling as a malicious user may attempt to exploit the application by providing it with invalid input to attempt to trigger a vulnerability. While simple Boolean analysis is the minimum. It is important be familiar with [Python exception handling](https://docs.python.org/3/tutorial/errors.html), specifically [the try statement](https://docs.python.org/3/reference/compound_stmts.html#try). |

**Activity 44**: students read the following descriptions of Python and implement the following code.

|  |
| --- |
| **Design**  When designing Python code with defensive data input handling practices, structure the code using functions and classes to encapsulate logic and promote code reusability.  Use type hints in function signatures (the private or public visibility, name, arguments, return types and modifiers) to enforce data types and improve code readability.  Design input validation functions to check user inputs against predefined criteria and sanitisation functions to cleanse input data of potentially harmful characters or scripts.  **Development**  During development, implement input validation by checking user inputs for correctness and ensuring they meet specified requirements.  Use Python's built-in functions and libraries like regex for pattern matching and validation. Implement sanitisation functions to remove or escape special characters from input data to prevent injection attacks.  Use try-except blocks for error handling to capture and handle exceptions gracefully. |

Students implement the [defensive data handling](https://github.com/TempeHS/The_Unsecure_PWA/tree/main/.student_resources/defensive_data_handling) code.

|  |
| --- |
| import data\_handler as sanitiser  import logging  logger = logging.getLogger(\_\_name\_\_)  logging.basicConfig(filename='security\_log.log', encoding='utf-8', level=logging.DEBUG, format='%(asctime)s %(message)s')  if \_\_name\_\_ == '\_\_main\_\_':  print (f"Does 'password' meet security requirements: {sanitiser.simple\_check\_password("password")}" )  print (f"Make <HTML> web safe: {sanitiser.make\_web\_safe('<html>')}")  print (f"Is 'name@example.com' an email address: {sanitiser.check\_email('name@example.com')}")  print (f"Is '123!' an name: {sanitiser.validate\_name('123!')}")  print (f"Is '1234567890' a number: {sanitiser.validate\_number('1234567890')}")  print ("--PYTHONIC EXCEPTION HANDLING--")  #password = "!1234abcD&"  password = 123  try:  print (f"The password as the byte string : {sanitiser.check\_password(password).hex()} is ready to be encryted" )  except TypeError:  logger.error(f"Type errors for password:{password}")  print("TypeError has been logged")  except ValueError as inst:  print(f"Not a valid password because it has {inst.args}.")  except Exception as inst:  print(f"Log as a {type(inst)}") |

Improved password checking with Pythonic exception handling and conversion to a byte string for privacy

Documentation: <https://docs.python.org/3/tutorial/errors.html>

**Note**: the password has been stored as a variable to make it easy for you to test the code. It is best practice to pass the date on entry to be validated first before storing.

**Activity 46**: students apply the PRIMM technique (Predict Rum Investigate Modify and Make) the above code to explain how it works.

|  |
| --- |
| Imports  import data\_handler as sanitiser  import logging  **data\_handler as sanitiser:**  This line imports a module named data\_handler and gives it the nickname sanitiser.  This means you can use sanitiser to call functions from data\_handler.  **import logging:**  This imports Python's built-in logging module, which allows you to record messages that can help you debug your programs. |

|  |
| --- |
| Setting Up Logging  logger = logging.getLogger(\_\_name\_\_)  logging.basicConfig(filename='security\_log.log', encoding='utf-8', level=logging.DEBUG, format='%(asctime)s %(message)s')  Creating a Logger:  logger = logging.getLogger(\_\_name\_\_) creates a logger that will record messages.  Basic Configuration:  logging.basicConfig(...) sets up how logging works:  filename='security\_log.log': Logs will be saved to a file named security\_log.log.  encoding='utf-8': Ensures that the log file can handle special characters.  level=logging.DEBUG: This means all messages of level DEBUG or higher will be logged.  format='%(asctime)s %(message)s': This sets the format of the log messages to include the date and time. |

|  |
| --- |
| Main Program Execution  if \_\_name\_\_ == '\_\_main\_\_':  This line checks if the script is being run directly (not imported as a module).  If it is, the code inside this block will execute. |

|  |
| --- |
| Function Calls and Outputs  The following lines print the results of various functions from the sanitiser module:  print (f"Does 'password' meet security requirements: {sanitiser.simple\_check\_password('password')}" )  print (f"Make <HTML> web safe: {sanitiser.make\_web\_safe('<html>')}")  print (f"Is 'name@example.com' an email address: {sanitiser.check\_email('name@example.com')}")  print (f"Is '123!' a name: {sanitiser.validate\_name('123!')}")  print (f"Is '1234567890' a number: {sanitiser.validate\_number('1234567890')}")  Each line calls a different function from the sanitiser module and prints out the result:  simple\_check\_password('password'): Checks if the password meets security requirements (returns True or False).  make\_web\_safe('<html>'): Makes the string <html> safe for web use.  check\_email('name@example.com'): Checks if the string is a valid email format.  validate\_name('123!'): Checks if the string is a valid name (only letters).  validate\_number('1234567890'): Checks if the string is a valid number. |

|  |
| --- |
| Exception Handling  password = 123  try:  print(f"The password as the byte string : {sanitiser.check\_password(password).hex()} is ready to be encrypted")  except TypeError:  logger.error(f"Type errors for password: {password}")  print("TypeError has been logged")  except ValueError as inst:  print(f"Not a valid password because it has {inst.args}.")  except Exception as inst:  print(f"Log as a {type(inst)}")  Password Variable: The variable password is set to 123, which is not a valid password.  try Block: This block attempts to check the password using sanitiser.check\_password(password).  If successful: It converts the password to a byte string and prints a message.  Exception Handling:  TypeError: If the input is not a string, it logs this error and prints a message.  ValueError: If the password fails any validation checks, it prints a message explaining why.  Exception: A catch-all for any other exceptions that might occur, printing the type of error. |

The code checks the validity of various inputs (like passwords and emails) and logs errors if anything goes wrong.

It uses functions from an imported module (data\_handler) to perform specific checks, and it handles errors in a way that provides useful feedback and logging for debugging.

The if \_\_name\_\_ == '\_\_main\_\_':part ensures that the code only runs when the script is executed directly, not when imported elsewhere.

**Activity 45**: students apply the PRIMM technique (Predict Rum Investigate Modify and Make) to the above code to explain how it works. They are provided with the **keywords** and explain what each does in this Python code.

|  |
| --- |
| simple\_check\_password(password)  **Checks if a given password meets certain criteria**  How it works: type check: it first checks if the input is a string. If not, it returns False.  Length check: the password must be between 8 and 20 characters. If it isn't, it returns False.  Space check: the password cannot contain spaces. If it does, it returns False.  Character checks:   * Must contain at least one uppercase letter (A–Z) * Must contain at least one lowercase letter (a–z) * Must contain at least one digit (0–9) * Must contain at least one special character from the set @$!%\*?&   Return: if all checks pass, it returns True, indicating the password is valid.  check\_password(password)  Similar to simple\_check\_password, but raises errors instead of returning False.  How it works: it performs the same checks assimple\_check\_password.  Instead of returning False for invalid inputs, it raises specific exceptions (like TypeError or ValueError) with descriptive messages.  Return: if the password is valid, it encodes the password (to prevent logging it in plain text) and returns it.  make\_web\_safe(string)  Converts a string into a ‘safe’ format for web use.  How it works: uses the html.escape() function to replace special HTML characters (like< and >) with safe representations so they don't break the HTML structure.  Return: returns the escaped string.  check\_email(email)  Validates an email address format.  How it works: uses a regular expression (regex) to check if the email matches a standard email format.  Return: returns True if the email is valid; otherwise, returns False.  validate\_name(name)  Checks if a name consists only of alphabetic characters.  How it works: uses the isalpha() method, which returns True if all characters in the string are letters.  Return: returns False if the name contains non-alphabetic characters; otherwise, returns True.  validate\_number(number)  Checks if a number is valid by ensuring it doesn't contain alphabetic characters.  How it works: similar to validate\_name but checks if the string is entirely numeric.  Return: returns False if the string contains any letters; otherwise, returns True.  salt\_and\_hash(password)  This function is meant to hash a password securely, but it is currently not implemented.  How it works: the idea is that when you store passwords, they should not be stored in plain text for security reasons. Instead, they should be ‘hashed’ (a one-way encryption).  Return: nothing is returned because the implementation is pending. |

This code is mainly focused on validating user input for security purposes (like passwords and emails) and ensuring that it adheres to specific rules.

It helps prevent common security issues, such as weak passwords or invalid email formats.

The salt\_and\_hash function is a placeholder for future implementation to securely store passwords.

### Design, develop and implement a safe application programming interface (API) to minimise software vulnerabilities

**Activity 47**: teachers issue the following examples of a simple API using **Flask and Python**. As the teacher reads through the code aloud students annotate their hand out with a description of the purpose of the code.

|  |
| --- |
| API: https://github.com/TempeHS/The\_Unsecure\_PWA/tree/main/.studentResources/safe\_API  from flask import Flask  from flask import request  from flask import jsonify  import database\_management as dbHandler  from flask\_cors import CORS, cross\_origin  app = Flask(\_\_name\_\_)  cors = CORS(app)  app.config['CORS\_HEADERS'] = 'Content-Type'  @app.route('/', methods=['GET'])  def get\_film():  film = dbHandler.get\_random\_film()  # For security data is validated on entry  if request.args.get("like") and request.args.get("like").isdigit():  film\_id = request.args.get("like")  app.logger.info(f"You have liked the film id={film\_id}") #debugging statement only  dbHandler.record\_like(film\_id)  # For security data is validated on entry  if request.args.get("dislike") and request.args.get("dislike").isdigit():  film\_id = request.args.get("dislike")  app.logger.critical(f"You have disliked the film id={film\_id}") #debugging statement only  dbHandler.record\_dislike(film\_id)  return jsonify(film), 200  @app.route('/add\_film', methods=['POST', 'HEAD'])  def add\_film():  data = request.get\_json()  info = dict(request.headers)  app.logger.critical(f"User {info}")  app.logger.critical(f"Has added the movie {data}")  dbHandler.add\_film(data)  return data, 201  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True, host='0.0.0.0', port=1000) |

Annotations to Python

|  |
| --- |
| **Importing Required Modules**  from flask import Flask  from flask import request  from flask import jsonify  import database\_management as dbHandler  from flask\_cors import CORS, cross\_origin  **Flask**: a web framework in Python used to build web applications.  **request**: a Flask module for handling HTTP requests.  **jsonify**: a Flask module for converting Python dictionaries to JSON format.  **database\_management as dbHandler**: a custom module for handling database operations, abbreviated as dbHandler.  **CORS, cross\_origin**: modules to handle Cross-Origin Resource Sharing (CORS), which allows your web application to access resources from different origins.  **Setting up Flask application**  app = Flask(\_\_name\_\_)  cors = CORS(app)  app.config['CORS\_HEADERS'] = 'Content-Type'  **app = Flask(name)**: ceates a Flask application instance.  **cors = CORS(app)**: enables CORS for the Flask app to allow resource sharing across different origins.  **app.config['CORS\_HEADERS'] = 'Content-Type'**: configures the CORS settings to include 'Content-Type' headers in responses.  **Defining routes**  The Root Route ('/')  @app.route('/', methods=['GET'])  def get\_film():  film = dbHandler.get\_random\_film()  # For security data is validated on entry  if request.args.get("like") and request.args.get("like").isdigit():  film\_id = request.args.get("like")  app.logger.info(f"You have liked the film id={film\_id}") #debugging statement only  dbHandler.record\_like(film\_id)  # For security data is validated on entry  if request.args.get("dislike") and request.args.get("dislike").isdigit():  film\_id = request.args.get("dislike")  app.logger.critical(f"You have disliked the film id={film\_id}") #debugging statement only  dbHandler.record\_dislike(film\_id)  return jsonify(film), 200  **@app.route('/', methods=['GET'])**: Defines a route for the root URL ('/') that only accepts GET requests.  **def get\_film():** function to handle requests to the root URL.  **film = dbHandler.get\_random\_film()**: calls a function from dbHandler to get a random film.  **if request.args.get("like") and request.args.get("like").isdigit():** checks if a 'like' parameter is present in the query string and if it's a digit.  **film\_id = request.args.get("like")**: retrieves the 'like' parameter value.  **app.logger.info(f"You have liked the film id={film\_id}")**: logs the 'like' action for debugging.  **dbHandler.record\_like(film\_id)**: records the 'like' action in the database.  **if request.args.get("dislike") and request.args.get("dislike").isdigit():** checks if a 'dislike' parameter is present in the query string and if it's a digit.  **film\_id = request.args.get("dislike")**: retrieves the 'dislike' parameter value.  **app.logger.critical(f"You have disliked the film id={film\_id}")**: logs the 'dislike' action for debugging.  **dbHandler.record\_dislike(film\_id)**: records the 'dislike' action in the database.  **return jsonify(film), 200**: returns the film data in JSON format with an HTTP status code of 200 (OK).  **The Add Film Route ('/add\_film')**  @app.route('/add\_film', methods=['POST', 'HEAD'])  def add\_film():  data = request.get\_json()  info = dict(request.headers)  app.logger.critical(f"User {info}")  app.logger.critical(f"Has added the movie {data}")  dbHandler.add\_film(data)  return data, 201  **@app.route('/add\_film', methods=['POST', 'HEAD'])**: defines a route for '/add\_film' that accepts POST and HEAD requests.  **def add\_film():** function to handle requests to '/add\_film'.  **data = request.get\_json()**: extracts the JSON payload from the request.  **info = dict(request.headers)**: converts the request headers to a dictionary.  **app.logger.critical(f"User {info}")**: logs the user information for debugging.  **app.logger.critical(f"Has added the movie {data}")**: logs the movie data being added for debugging.  **dbHandler.add\_film(data)**: calls a function from dbHandler to add the film data to the database.  **return data, 201**: returns the added film data with an HTTP status code of 201 (Created).  **Running the Flask application**  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True, host='0.0.0.0', port=1000)  **If name == 'main':**: ensures the application runs only if this script is executed directly, not when imported as a module.  **app.run(debug=True, host='0.0.0.0', port=1000)**: runs the Flask application with debugging enabled, accessible from any network interface (host='0.0.0.0') on port 1000. |

Table 19 – Flask summary

|  |  |
| --- | --- |
| Key steps | Description |
| **Flask application setup** | Initialises a Flask app with CORS enabled. |
| **Routes** | Defines 2 routes: one for getting a random film and optionally liking or disliking it, and another for adding a film. |
| **Logging** | Uses Flask's logging mechanism to log important actions for debugging. |
| **Database interaction** | Interacts with a database using functions from a custom database\_management module (dbHandler). |
| **Running the app** | Runs the app with debugging enabled on port 1000 when the script is executed directly. |

## Design, develop and implement code considering efficient execution for the user

#### Including: memory management, session management, exception management

**Activity 48**: students watch [JavaScript Memory Management (3:23)](https://www.youtube.com/watch?v=gOeW5_UnN9g) and visit: [Optimizing JavaScript Performance: Memory Management and Code Profiling](https://www.tutorialspoint.com/optimizing-javascript-performance-memory-management-and-code-profiling) by Tutorials Point to explain the importance of memory management.

|  |
| --- |
| Memory management is important because it helps ensure that the computer's resources are used wisely including:  **Saving resources**: managing memory well means using the computer's memory in the best way possible. This prevents wasting memory and helps the program run smoothly.  **Avoiding memory leaks**: if a program uses memory but forgets to give it back when it's done, it causes a memory leak. This can make the program slow and may even make it crash. Good memory management helps find and fix these issues.  **Making things faster**: when memory is managed efficiently, the program runs faster. This means things happen quickly, and users do not have to wait too long for actions to be completed.  **Handling more users**: as more people start using the program, good memory management ensures that the program can keep up with the increased demand without slowing down or crashing.  **Keeping the program stable**: memory problems can cause the program to crash or behave unpredictably. By managing memory well, the program becomes more stable and reliable.  **Protecting against attacks**: poor memory management can create security weaknesses that hackers can exploit. By managing memory securely, these vulnerabilities can be reduced, making the program safer for users. |

**Activity 49**: students visit [Session Management 101: A Beginner's Guide for Web Developers](https://mojoauth.com/blog/session-management-a-beginners-guide-for-web-developers/) by MojoAuth and [Session Management Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html) by the Cheat Sheets Series Team to explain the importance of session management.

|  |
| --- |
| Session management is important because it helps keep track of user interactions with a website or application. Reasons why it is important include:  **Personalised experience**: session management allows the website or app to remember user preferences and settings. This makes the user's experience more personalised and convenient.  **Security**: sessions help authenticate users and ensure that only authorised individuals can access certain parts of the website or application. It helps protect user data and prevent unauthorised access.  **Remembering user actions**: sessions store information about what the user is doing on the website or app. This helps remember where the user left off, like items in a shopping cart, even if they navigate away and come back later.  **Preventing data loss**: by storing data temporarily during a session, session management prevents the loss of essential information when users move between different pages or perform actions on the site.  **Managing user interactions**: sessions help track how users interact with the website or app, such as logging in, submitting forms, or browsing content. This information can be used to improve the user experience and analyse user behaviour.  **Enhancing performance**: efficient session management ensures that user sessions are handled smoothly, without unnecessary delays or errors. This contributes to a faster and more responsive user experience. |

**Activity 50**: students visit Software [Error vs Exception – In Real World Examples](https://stackify.com/application-exception-vs-error-difference/).

Explain the importance of exception management in the space below.

|  |
| --- |
| **Dealing with surprises**: exception management allows the program to handle unexpected problems, like errors or issues that were not planned for. It helps the program respond appropriately when things do not go as expected.  **Preventing crashes**: when a program encounters an error or a problem it can't handle, it may crash. Exception management helps catch these issues and prevents the program from crashing, keeping it running smoothly.  **Providing guidance**: when an error occurs, exception management can show a helpful message to the user, explaining what went wrong and how to proceed. This guidance helps users understand the problem and how to address it.  **Maintaining stability**: by managing exceptions well, the program remains stable and reliable even when unexpected issues arise. This ensures that users can continue using the program without disruptions.  **Debugging help**: exception management provides valuable information about what went wrong in the program. This information can be used for debugging and fixing problems, making the program more robust over time.  **Security**: handling exceptions properly can also help protect the program from potential security threats. By managing errors effectively, the program can prevent vulnerabilities that could be exploited by attackers.  Exception management is like having a safety net for a tightrope walker. If they lose their balance, the safety net catches them and helps them get back on track. Similarly, in a program, exception management catches unexpected problems and helps the program recover smoothly, keeping things running safely and securely. |

## Design, develop and implement secure code to minimise vulnerabilities in user action controls

#### Including: broken authentication and session management, cross-site scripting (XSS) and cross-site request forgery (CSRF), invalid forwarding and redirecting, race conditions

The video activity below looks at the first 2 examples of vulnerabilities in JavaScript. This activity could be extended to complete summaries of all 10 vulnerabilities and solutions are provided in the Appendix. This may be useful where students wish to create a website for their major project

**Activity 51**: As a class, watch [JavaScript Security Vulnerabilities Tutorial – With Code Examples (25:04)](https://www.youtube.com/watch?v=ypNKKYUJE5o).

Teacher pauses the video as Brandon asks the viewer to think about the vulnerability, then explains it and provides a solution. Students take notes for each example and provide a summary in the spaces below:

1. Cross-site scripting

|  |
| --- |
| In this example, a simple web app is created using JavaScript and React.   * The app demonstrates how query parameters work, which are used to pass information in a URL. The vulnerability in this code is due to a security issue called **cross-site scripting.** * This type of attack allows malicious code to be executed on a user's browser. * The vulnerability arises from the use of the JavaScript protocol, which can execute code embedded in a URL. In the context of the web app, if a user clicks on a button that redirects based on a query parameter value, an attacker could manipulate the URL to execute harmful code on the user's machine. |

Cross-site scripting FIX:

|  |
| --- |
| To fix the vulnerability in the web app, ensure that the URL used is safe.  To do this create a validate URL function.  This function will check if the URL uses a safe protocol like HTTPS.  How it works:   1. The function takes a URL as input. 2. It creates a new URL object to access details about the URL. 3. It checks the URL's protocol. If it is HTTPS, the URL is safe, and we return it as is. 4. If the protocol is not safe (for example, JavaScript), we return a default value ("/") to avoid potential attacks.   Passing the query parameter through this validate URL function, ensures that the app doesn't execute harmful scripts, protecting against cross-site scripting attacks. |

1. Server-side request forgery (SSRF)

|  |
| --- |
| **Description:** a JavaScript program creates a simple web app.  The app listens for GET requests at the endpoint "/API/data" and uses query parameters to fetch data.  When you visit a URL like "example.com/API/data?URL=foo", the app assigns "foo" to a constant named URL.  It then fetches data from the specified URL, converts it to JSON, and returns it to the user.  The app also fetches data from internal files, like "countries.json" and "states.json", which are publicly accessible.  If you visit the URL with these query parameters, the app will fetch and return the data from these internal files.  **Challenge**: the challenge is to figure out why this setup might be vulnerable. Think about the potential risks of fetching data based on user-provided URLs and how it might be exploited.  **Definition**: this code example is vulnerable to a type of attack called **server-side request forgery (SSRF**). In SSRF attacks, an attacker manipulates a server into performing actions on their behalf that the attacker does not have permission to do. The vulnerability in this case arises from the web app fetching data from URLs specified in query parameters, potentially accessing confidential information that should not be public. |

Server-side request forgery (SSRF) FIX:

|  |
| --- |
| By creating a list of allowed URLs containing only public data sources, like "states.json", the web app can verify if the fetched URL is within the allowed list.  If the URL is not in the allowed list, the app will return an error message and prevent the server from accessing unauthorised or confidential data.  By implementing this approach, the web app ensures that it only accesses public data sources and mitigates the risk of an SSRF attack, safeguarding against unauthorised access to sensitive information. |

**Activity 52**: students complete the online interactive lessons on [Cross-site scripting](https://www.hacksplaining.com/lessons/xss-stored/start).

**Activity 53**: students investigate the Python code from the [Unsecure PWA](https://github.com/TempeHS/The_Unsecure_PWA/tree/main/.student_resources/defensive_data_handling) to describe the importance of feedback, exceptions and logging.

|  |
| --- |
| **Input validation**  Ensures that only valid data is processed or stored. By validating data at the entry point, you can prevent invalid or malicious input from causing harm. Feedback should be provided to users if their input is not valid, enhancing user experience and security.  **Data sanitisation**  This process cleans data before processing or storage, replacing potentially harmful characters with safe alternatives. It prevents malicious code from being executed, ensuring that only web-safe strings are handled.  **Exception handling**  Essential for protecting applications from malicious users who might exploit vulnerabilities through invalid inputs. Familiarity with exception handling (for example, using try statements in Python) allows developers to manage errors effectively and maintain application stability.  **Logging**  Implementing logging helps in detecting and analysing malicious behaviour. Each error or unexpected event should be recorded with sufficient details, aiding in the improvement of data handling practices. Regular reviews of logs are crucial for maintaining security and aligning with best practices as recommended by cybersecurity authorities. |

**Session management**

|  |
| --- |
| from flask import Flask, request, session, redirect, url\_for  from flask\_bcrypt import Bcrypt  import secrets  app = Flask(\_\_name\_\_)  bcrypt = Bcrypt(app)  app.secret\_key = secrets.token\_hex(16)  # In-memory user database (replace with a secure database in production)  users = {      'user1': {'username': 'user1', 'password': bcrypt.generate\_password\_hash('password1').decode('utf-8')},      'user2': {'username': 'user2', 'password': bcrypt.generate\_password\_hash('password2').decode('utf-8')}  }  def is\_authenticated():      return 'username' in session  def authenticate(username, password):      user = users.get(username)      if user and bcrypt.check\_password\_hash(user['password'], password):          session['username'] = username          return True      return False  def logout():      session.pop('username', None)  @app.route('/')  def home():      if is\_authenticated():          return f"Welcome, {session['username']}! <a href='/logout'>Logout</a>"      else:          return "Welcome! Please <a href='/login'>login</a>."  @app.route('/login', methods=['GET', 'POST'])  def login():      if request.method == 'POST':          username = request.form.get('username')          password = request.form.get('password')          if authenticate(username, password):              return redirect(url\_for('home'))          else:              return "Invalid credentials. <a href='/login'>Try again</a>."      else:          return '''          <form method="post" action="/login">              <label for="username">Username:</label>              <input type="text" name="username" required><br>              <label for="password">Password:</label>              <input type="password" name="password" required><br>              <input type="submit" value="Login">          </form>          '''  @app.route('/logout')  def logout\_route():      logout()      return redirect(url\_for('home'))  if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True)  # Write your code here :-) |

Including a session timeout and reauthentication for administrative processes would show best practice session handling.

**Cross-site scripting (XSS)**

|  |
| --- |
| #import html  def render\_html(content):      # Example: Escape user input to prevent XSS      sanitized\_content = html.escape(content)      return f"<p>{sanitized\_content}</p>"   Write your code here :-) |

Students visit[Cross-Site Request Forgery (CSRF)](https://github.com/TempeHS/The_Unsecure_PWA/blob/main/.student_resources/CSRF/README.md)and compare this with the solution in the [Secure Flask PWA Template](https://github.com/TempeHS/Secure_Flask_PWA_Template/tree/main).

|  |
| --- |
| The implementation of Flask WTForms generates and requires a unique key by default:  app = Flask(\_\_name\_\_)  app.secret\_key = b'\_53oi3uriq9pifpff;apl' ##create your own key  app.secret\_key = b"\_53oi3uriq9pifpff;apl" ##create your own key  csrf = CSRFProtect(app) |

**Invalid forwarding and redirecting, race conditions**

|  |
| --- |
| from flask import Flask, render\_template, redirect, request, abort, url\_for  app = Flask(\_\_name\_\_)  def is\_valid\_redirect\_url(target):      # Check if the target URL is a safe and valid URL within the application      # You may customize this function based on your application's requirements      safe\_urls = ['/home', '/dashboard', '/profile']      return target.startswith('/') and target in safe\_urls  @app.route('/')  def home():      return render\_template('index.html')  @app.route('/redirect', methods=['GET'])  def redirect\_example():      target = request.args.get('target')      if target and is\_valid\_redirect\_url(target):          # Perform the redirect only if the target is valid          return redirect(target)      else:          # Invalid or unsafe redirect, handle appropriately (e.g., log, redirect to a safe location)  abort(400)  if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True)  # Write your code here :-) |

**Race conditions**

|  |
| --- |
| from flask import Flask, render\_template, request, session  from threading import Lock  import time  app = Flask(\_\_name\_\_)  app.secret\_key = 'your\_secret\_key'  lock = Lock()  def simulate\_database\_operation():      # Simulate a time-consuming database operation      time.sleep(2)  @app.route('/')  def home():      return render\_template('index.html')  @app.route('/update\_profile', methods=['POST'])  def update\_profile():      if 'username' in session:          username = session['username']          # Use a lock to prevent race conditions during the update operation          with lock:              # Simulate a database operation that may lead to a race condition              simulate\_database\_operation()              # Perform the update operation safely              # In a real-world scenario, this could involve updating user profile information              return f"Profile updated successfully for {username}."      return "User not authenticated."  if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True)  # Write your code here :-) |

**Extension**: students watch the [JavaScript Security Vulnerabilities Tutorial – With Code Examples (25:04)](https://www.youtube.com/watch?v=ypNKKYUJE5o) research how these are treated in other platforms and frameworks.

### Design, develop and implement secure code to protect user file and hardware vulnerabilities from file attacks and side channel attacks

**Activity 54**: students investigate the Python code samples below to assist in their design, development, and implementation of secure code to protect user file and hardware vulnerabilities from file attacks and side channel attacks.

**Secure file handling**

|  |
| --- |
| from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC  from cryptography.hazmat.backends import default\_backend  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes  import os  def encrypt\_file(file\_path, key):      # Generate a random IV (Initialization Vector)      iv = os.urandom(16)      # Derive a key from the user's key using PBKDF2      kdf = PBKDF2HMAC(          algorithm=algorithms.SHA256(),          length=32,          salt=os.urandom(16),          iterations=100000,          backend=default\_backend()      )      derived\_key = kdf.derive(key)      # Encrypt the file content      with open(file\_path, 'rb') as file:          plaintext = file.read()      cipher = Cipher(algorithms.AES(derived\_key), modes.CFB(iv), backend=default\_backend())      encryptor = cipher.encryptor()      ciphertext = encryptor.update(plaintext) + encryptor.finalize()      # Save the encrypted content back to the file      with open(file\_path + '.encrypted', 'wb') as encrypted\_file:          encrypted\_file.write(iv + ciphertext)  def decrypt\_file(file\_path, key):      # Derive the key using PBKDF2      kdf = PBKDF2HMAC(          algorithm=algorithms.SHA256(),          length=32,          salt=os.urandom(16),          iterations=100000,          backend=default\_backend()      )      derived\_key = kdf.derive(key)      # Read the encrypted content from the file      with open(file\_path, 'rb') as encrypted\_file:          data = encrypted\_file.read()      iv, ciphertext = data[:16], data[16:]      # Decrypt the content      cipher = Cipher(algorithms.AES(derived\_key), modes.CFB(iv), backend=default\_backend())      decryptor = cipher.decryptor()      plaintext = decryptor.update(ciphertext) + decryptor.finalize()      # Save the decrypted content back to the file      with open(file\_path + '.decrypted', 'wb') as decrypted\_file:          decrypted\_file.write(plaintext)  # Example usage  file\_path = 'example.txt'  user\_key = b'my\_secret\_key'  # Encrypt the file  encrypt\_file(file\_path, user\_key)  # Decrypt the file  decrypt\_file(file\_path + '.encrypted', user\_key)  # Write your code here :-) |

**Side-channel attacks**

|  |
| --- |
| import time  def secure\_comparison(a, b):      # Perform a constant-time comparison to mitigate timing attacks      if len(a) != len(b):          return False      result = 0      for x, y in zip(a, b):          result |= x ^ y      return result == 0  def authenticate\_user(user\_input, stored\_password):      # Simulate user authentication (check against stored password)      time.sleep(0.1)  # Simulate a time delay to exploit in a timing attack      return secure\_comparison(user\_input.encode(), stored\_password)  # Example usage  stored\_password = b'secret\_password'  # Simulate timing attack  for i in range(10):      user\_input = b'wrong\_password'  # Try different passwords      start\_time = time.time()      authenticate\_user(user\_input, stored\_password)      end\_time = time.time()      elapsed\_time = end\_time - start\_time      print(f'Trial {i + 1}: Elapsed time: {elapsed\_time:.6f} seconds')  # Write your code here :-) |

## Impact of safe and secure software development

### Apply and describe the benefits of collaboration to develop safe and secure software Including: considering various points of view, delegating tasks based on expertise, quality of the solution.

**Teacher** **note**: students should be able to identify where they have applied collaboration in their secure software architecture assessment task and describe the benefits.

**Activity 55**: students complete the table using the key words.

|  |
| --- |
| **Collaboration** among team members when developing software can lead to numerous benefits, especially in terms of ensuring safety and security. Some key advantages of collaboration include:   * **Diverse points of view**: when team members with divergent backgrounds, experiences, and expertise come together to collaborate on a project, they bring diverse perspectives to the table. This diversity of thought can help identify potential vulnerabilities or security risks that may have been overlooked by individuals working in isolation. * **Delegating tasks based on expertise**: by leveraging the unique skills and expertise of team members, tasks can be delegated to those best equipped to handle them. This ensures that each aspect of the software development process is managed by individuals with the appropriate knowledge and experience, leading to a higher quality end-product. * **Continuous feedback and review:** collaboration facilitates ongoing communication and feedback among team members throughout the development process. This constant exchange of ideas and information allows for timely identification and resolution of security issues, resulting in a safer and more secure software solution. * **Shared responsibility:** when team members collaborate effectively, there is a shared sense of responsibility for the security and quality of the software being developed. This collective accountability fosters a culture of vigilance and attention to detail, leading to a more thorough and secure end-product. |

### Investigate and explain the benefits to an enterprise of the implementation of safe and secure development practices

#### Including: improved products or services, influence on future software development, improved work practices, productivity, business interactivity

**Activity 56**: investigate and explain the benefits of the implementing safe and secure development practices to your client, ‘The Unsecure PWA Company’.

|  |
| --- |
| **Improved products or services**: by prioritising safe and secure development practices, enterprises can enhance the quality and reliability of their products or services. This can lead to increased customer trust and satisfaction, as users are more likely to engage with solutions that are known for their robust security measures.  **Influence on future software development**: adopting safe and secure development practices sets a precedent within the organisation for future projects. Teams are more likely to incorporate security considerations from the outset, leading to a proactive approach to safeguarding data and systems in all software development initiatives.  **Improved work practices**: safe and secure development practices often involve implementing structured processes and guidelines for coding, testing, and deployment. This can result in more efficient work practices, as developers follow best practices and standards to mitigate security risks and vulnerabilities throughout the development lifecycle.  **Enhanced productivity**: investing in safe and secure development practices can boost productivity within the enterprise. By addressing security concerns early in the development process, teams can avoid costly rework and time-consuming fixes later. This streamlined approach can lead to faster time-to-market for products and services.  **Business interactivity**: enterprises that prioritise safe and secure development practices are better positioned to engage with partners, clients, and stakeholders in a secure manner. This can open new opportunities for collaboration and business growth, as the organisation demonstrates a commitment to protecting sensitive information and maintaining data integrity. |

### Evaluate the social, ethical, and legal issues and ramifications that affect people and enterprises resulting from the development and implementation of safe and secure software

#### Including: employment, data security, privacy, copyright, intellectual property, digital disruption

Figure 4 – social, ethical and legal ramifications of secure code

A diagram displaying social, ethical and legal considerations in software development, including:
Social: employment and digital disruption.
Ethical: data security and copyright.
Legal: privacy and intellectual property.

**Jigsaw Activity 57**: students form teams of 3. Each student is number 1, 2 or 3.

All the 1’s join to research and evaluate social issues and ramifications that result from safe and secure code.

All the 2’s join to research and evaluate ethical issues and ramifications that result from safe and secure code.

All the 3’s join to research and evaluate legal issues and ramifications that result from safe and secure code.

Each group produces an information slide deck for their topic. Teams reform and make a complete revision slide deck of this topic.

These teams present to the class and students complete the table below including providing examples.

Table 20 – social, ethical and legal issues

|  |  |  |
| --- | --- | --- |
| Social issues | Ethical issues | Legal issues |
| Employment | **Data security** | **Privacy** |
| Impact on job roles and skills due to automation and AI technologies. | Ensuring the protection of sensitive data from unauthorised access or breaches. | Compliance with data protection regulations and laws to safeguard personal information. |
| Example: technology companies implementing AI and automation systems leading to job displacement and evolving skill requirements. | Example: implementation of encryption and access controls to prevent data breaches and cyber attacks | Example: adhering to GDPR requirements to protect customer data privacy and ensure lawful processing. |
| Digital disruption | **Copyright** | **Intellectual property** |
| Disruption of traditional business models and industries by digital technologies. | Protecting original works and preventing unauthorised copying or distribution. | Safeguarding intangible assets such as software code, patents, and trade secrets. |
| Example: rise of e-commerce platforms disrupting traditional retail industries and brick-and-mortar stores. | Example: enforcing copyright laws to prevent the unauthorised reproduction of software or digital content. | Example: securing software patents to protect unique algorithms and innovations developed by the organisation. |

This table provides a structured overview of the social, ethical, and legal issues related to secure software development, highlighting key considerations such as employment impacts, data security concerns, privacy regulations, copyright protection, intellectual property rights, and the effects of digital disruption on traditional business practices.

# Appendix 1 – security report

This following security report is based upon syllabus content.

Students should view the [Cloud Security Assessment Report Template](https://www.cyber.gov.au/resources-business-and-government/assessment-and-evaluation-programs/infosec-registered-assessors-program/irap-resources) on the Australian Signals Directorate website to see a professional example of a security report for cybersecurity.

**SECURITY FUNDAMENTALS AND ASSESSMENT REPORT**

**THE UNSECURE PWA COMPANY**

**UNSECURE PWA Security Report**

<Assessor Details>

Document details

### **Assessment**

|  |  |
| --- | --- |
| **Date** | <Month YYYY> |
| **Control classification** | Choose an item. |
| **Definition** | Choose an item. |
| **Deployment model** | Choose an item. |
| **Version** | V1.0 |

### **Prepared by <Assessor Organisation Name>**

|  |  |  |
| --- | --- | --- |
| **Address** |  |  |
| **Assessor name** |  |  |
| **Assessor qualifications** |  |  |
| **Contact email** |  |  |

### **Prepared for <Organisation Name>**

|  |  |  |
| --- | --- | --- |
| **Address** |  |  |
| **Contact name** |  |  |
| **Contact email** |  |  |

### **Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description** | **Author** |
| vX.X | DD/MM/YYYY |  |  |
|  |  |  |  |
|  |  |  |  |

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

Contents

Ensure that your security report contains a Table of contents.

To insert a Table of contents in Microsoft Word:

1. Navigate to **References** > **Table of Contents** > **Custom Table of Contents**.
2. Choose the number of heading levels you would like to display in the Table of contents.
3. Select **OK**.

To update the Table of contents:

1. Right click on the table and select **Update Field**.
2. Select **Update entire table**. Your table numbers should then update to reflect your changes.

## 1. Executive summary

* 1. Briefly outline the purpose of the report.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

## 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](https://github.com/TempeHS/The_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](https://www.imperva.com/learn/data-security/data-protection/) |  |  |
| [Cyber attacks](https://www.imperva.com/learn/application-security/cyber-security-threats/) |  |  |
| [Static application security testing (SAST)](https://www.imperva.com/learn/application-security/sast-iast-dast/) |  |  |
| [Dynamic application security testing (DAST)](https://www.imperva.com/learn/application-security/sast-iast-dast/) |  |  |
| [Vulnerability assessment](https://www.imperva.com/learn/application-security/vulnerability-assessment/) |  |  |
| [Penetration testing](https://www.imperva.com/learn/application-security/penetration-testing/) |  |  |
| [API security](https://www.imperva.com/learn/application-security/api-security/) |  |  |
| [Cross-site scripting (XSS)](https://brightsec.com/blog/xss/) |  |  |
| [Cross-site request forgery (CSRF)](https://brightsec.com/blog/cross-site-request-forgery-csrf/) |  |  |

## 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 |  |
| Integrity |  |
| Availability |  |
| Compliance with regulations |  |
| Protection against cyber threats |  |
| Customer trust |  |
| Business continuity |  |

3.1.2 Strategies employed for data encryption and storage.

|  |
| --- |
|  |

3.2 Minimising cyber attacks and vulnerabilities

3.2.1 Benefits of minimising common cyber threats and vulnerabilities.

Table 23 – minimising common cyber threats and vulnerabilities

|  |  |
| --- | --- |
| Issue | Explanation |
| Prevent data breaches |  |
| Mitigate against cyber threats |  |
| Protect customer trust |  |
| Comply with regulations |  |
| Business continuity |  |
| Cost savings |  |
| Competitive advantage |  |

Discussion on preventative measures implemented.

|  |
| --- |
|  |

## 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 |  |
| Tools and environment |  |
| Functional requirements |  |
| Non-functional requirements |  |
| Testing and validation |  |
| Documentation and presentation |  |
| Timeline |  |

Outline the process for gathering security requirements.

|  |
| --- |
|  |

4.2 Determining specifications

Table 25 – determining specifications

|  |  |
| --- | --- |
| Specification | Description |
| Login page development |  |
| Password hashing |  |
| Input field sanitisation |  |
| Two-factor authentication (2FA) |  |
| Penetration testing |  |
| Documentation and presentation |  |
| Security tools and technologies |  |
| Timeline and milestones |  |
| Scalability and performance |  |
| Compliance and best practices |  |

Discuss the specifications that enhance security.

|  |
| --- |
|  |

* 1. Design

Key design principles for secure architecture.

4.4 Development

Best practices for writing secure code.

Table 26 – design and development phases

|  |  |
| --- | --- |
| Phases | Description |
| Requirement analysis |  |
| Design phase |  |
| Development phase |  |
| Testing phase |  |
| Security assessment and validation |  |
| Documentation and presentation |  |
| Deployment and monitoring |  |

4.5 Integration

Security considerations during integration.

Table 27 – integration

|  |  |
| --- | --- |
| Steps | Description |
| Assessment of existing system |  |
| Requirements alignment |  |
| Modular development approach |  |
| API integration |  |
| Database integration |  |
| User interface integration |  |
| Testing and validation |  |
| Deployment and monitoring |  |

4.6 Testing and debugging

Methods for ensuring security through testing.

|  |
| --- |
|  |

* 1. Installation

Security measures taken during installation.

|  |
| --- |
|  |

* 1. Maintenance

Strategies for ongoing security monitoring and updates.

|  |
| --- |
|  |

## 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 |  |
| Role-based access control |  |
| Training and awareness |  |
| User feedback and testing |  |
| Simplified security controls |  |

5.2 Importance of user feedback in the design process.

|  |
| --- |
|  |

## 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 |  |
| Integrity |  |
| Availability |  |
| Authentication |  |
| Authorisation |  |
| Accountability |  |

## 7. Security features incorporated into software

7.1 Overview of security features implemented.

Table 30 – security features

|  |  |
| --- | --- |
| Security feature | Application |
| Data protection |  |
| Security measures |  |
| Privacy protection |  |
| Regulatory compliance |  |
| User authentication and authorisation |  |

## 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’.

|  |
| --- |
|  |

Explain how sandboxing limits exposure to vulnerabilities.

|  |
| --- |
|  |

Summarise security by design for the client.

|  |
| --- |
|  |

8.2 Privacy by design approach

Provide the client with the principles of proactive privacy measures.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

## 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 |  |
| Handling breaches |  |
| Disaster recovery plan |  |

## 10. Strategies for managing security of programming code

* 1. Code review

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

* 1. 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 |  |
| Static application security testing (SAST) |  |
| Dynamic application security testing (DAST) |  |
| Vulnerability assessment |  |
| Penetration testing |  |

## 11. Defensive data input handling

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

|  |
| --- |
|  |

## 12. Safe API development

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

|  |
| --- |
|  |

## 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 |  |
| Session management |  |
| Exception management |  |

## 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 |  |
| Cross-site scripting (XSS) |  |
| CSRF |  |
| Invalid forwarding and redirecting |  |
| Race conditions |  |

## 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.

|  |
| --- |
|  |

## 16. Conclusion

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

|  |
| --- |
|  |

## 17. References

List of resources and references used in the report.

## 18. Security report appendix

Any additional information or supporting documents.

# Appendix 2 – enhancing security of a progressive web app (PWA) with security measures

**Objective**

In this task, students will enhance the security of an unsecured PWA by implementing a range of security measures and algorithms. Students will focus on improving security for a login page, incorporating two-factor authentication (2FA) using Google Authenticator, hashing passwords in the database, and sanitising input fields. Additionally, students will conduct penetration testing (pen testing) to identify and address security vulnerabilities before implementing the security measures.

**Tools required**

* Visual Studio Code (VS Code)
* Node.js
* SQLite
* Git Bash
* Google Authenticator for 2FA implementation

**Task steps**

1. Set up environment

Install and configure Node.js, SQLite, and Git Bash in VS Code for web development.

1. Create Unsecured PWA

Develop a basic unsecured website with a login page that allows users to input usernames and passwords.

1. Penetration testing

Conduct pen testing using tools like OWASP ZAP or Burp Suite to identify security vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure direct object references.

1. Implement security measures

* Hash passwords using secure algorithms like Bcrypt before storing them in the database.
* Sanitise input fields, such as the username field, to prevent SQL injection attacks and cross-site scripting.
* Integrate Google Authenticator for two-factor authentication (2FA) to enhance login security.

1. Penetration testing (post-security measures)

* Conduct pen testing again to assess the effectiveness of the implemented security measures.
* Address any vulnerabilities discovered during pen testing and validate the security enhancements.

1. Documentation and presentation

* Document the security enhancements made to the PWA, including details of the implemented security measures and algorithms.
* Prepare a presentation to highlight the before-and-after security improvements and the pen testing results.

**Activity**: students watch [Beginner's Guide on How to Set up & Use Google Authenticator (2024) (8:59)](https://www.youtube.com/watch?v=seyOjyJGsxY).

Students summarise the video in in the space below.

|  |
| --- |
| The video tutorial demonstrates how to set up and use Google Authenticator including explaining its purpose and functionality.  Google Authenticator is a free security app that generates unique codes on smartphones for two-factor authentication, adding an extra layer of security to accounts.  The video covers setting up Google Authenticator with Gmail and discusses:   * backing up codes * security considerations * adding and removing accounts * alternative authentication methods.   It also provides guidance on downloading and setting up Google Authenticator, including enabling a privacy screen for added security |

**Activity**: students watch [Introducing 2FA with Google Authenticator and Node.js (8:12)](https://www.youtube.com/watch?v=6mxA9Zp8600).

Students describe how Node.js and Speakeasy work regarding 2FA Google Authenticator.

|  |
| --- |
| Node.js is a popular runtime environment that allows you to run JavaScript code outside of a web browser, making it suitable for server-side applications. Node.js can be used to build backend server applications that handle user authentication and generate the required codes for 2FA.  Developers can create APIs that communicate with client applications (such as web or mobile apps) to implement 2FA functionalities like generating and verifying authentication codes. Speakeasy is a Node.js library that provides functions for generating and verifying Time-based One-Time Passwords (TOTP) and HMAC-based One-Time Passwords (HOTP) for 2FA.  TOTP and HOTP are algorithms used to generate temporary codes that change after a specific time interval (TOTP) or based on a counter value (HOTP).  Speakeasy simplifies the process of integrating 2FA into Node.js applications by handling the generation and validation of these one-time passwords. |

**Extension activity**: students watch [Two-Factor Authentication (2FA) in Python (12:19)](https://www.youtube.com/watch?v=o0XZZkI69E8) and describe how OTP and HOTP work:

|  |
| --- |
| 1. **OTP (One-time password)** is a type of password that is valid for only one login session or transaction. Once used, it cannot be used again. It is typically used as the second factor, where the user enters a temporary code along with their regular password to access an account. Python libraries like pyotp can be used to generate OTP codes. These codes are often time-based, meaning they change every few seconds to enhance security. 2. **HOTP (HMAC-based one-time password)** is a specific type of OTP that is generated based on a counter value and a secret key using a cryptographic hash function. When a user logs in using HOTP for 2FA, the server and the user's device both maintain a counter value that increments with each login attempt. Python libraries like pyotp provide functions to create and validate HOTP codes based on the counter value and a shared secret key. |

**Extension activity**: students watch [Two Factor Authentication | Node.js & Speakeasy (24:58)](https://www.youtube.com/watch?v=KQya9i6czhM) and experiment with [Implementing two-factor authentication using Speakeasy](https://blog.logrocket.com/implementing-two-factor-authentication-using-speakeasy/) on their PWA by following the tutorial.

# Appendix 3 – how to guide – 2FA using Google Authenticator and Flask

Adding an additional layer of security to the Flask app using Google Authenticator requires the addition of a page in-between the login page and end page the user is aiming to access.

**Install the required libraries**

Ensure qrcode and pyotp are installed.

Use a package manager like pip to install qrcode and pyotp.

The qrcode package issued to generate the image (PNG file typically) and to render the QR codes directly to the console. In this case, we will send the file name to our html page to be rendered.

The pyotop package is used to generate the one-time password and is used to implement two-factor authentication in web applications. Further information is provided in the article [pyotp 2.9.0](https://pypi.org/project/pyotp/).

from flask import Flask, render\_template, request, redirect, url\_for, session

import userManagement as dbHandler

import pyotp

import pyqrcode

import os

import base64

from io import BytesIO

**Adding 2FA**

Start by adding in a secret key variable at beginning of your python file. This will be used for securely signing the session cookies and can be used for any other security related needs.

app = Flask(\_\_name\_\_)

app.secret\_key = 'my\_secret\_key'

We also need to add in code to generate the one-time passcode (user\_secret = pyotp.random\_base32())

def home():

user\_secret = pyotp.random\_base32() #generate the one-time passcode

return redirect(url\_for('enable\_2fa')) #redirect to 2FA page

**Create the HTML page**

Create the index HTML page within the template folder and name it index.html.

We then need to insert the code below which simply displays the QRCode generated in the python file, renders to image using the img tag and request entering of the one-time passcode using a form.

This tag in HTML adds the QRcode generated in the python file. Make sure the name used within the { } brackets, in this case, qr\_code, matches the one created in the Python file.

<img src="data:image/png;base64,{{ qr\_code }}">

<h1>Welcome Enable 2FA {{ value }}!</h1>

<h1>Scan this QR Code with Google Authenticator</h1>

<img src="data:image/png;base64,{{ qr\_code }}">

We then need to add a form to the page to get the code entered by the user after they scan the QRcode.

Again, make sure the name you assign to the input matches the one used in the Python file, in this case, otp.

<form action="/index.html" method="post">

<label for="otp">Enter the OTP from your app:</label><br>

<input type="text" id="otp" name="otp"><br>

<input type="submit" value="Enable 2FA">

</form>

**Create routes in Flask**

Create routes to handle HTTP requests (like GET and POST). When a user submits the one time passcode.

@app.route('/index.html', methods=['POST', 'GET'])

@app.route('/', methods=['POST', 'GET'])

Create the function home to deal with the generation of the QRcode image, one-time passcode and to verify the code entered matches:

This section of code we need to generate a secret key for the user.

def home():

user\_secret = pyotp.random\_base32()

We are now going to generate the QRCode and one-time passcode.

The line of code below is used generate the one time passcode based on the secret key generated in the previous step. This will then be used to generate the QRCode using the inbuilt function totp.provisioning\_uri(name=username,issuer\_name="YourAppName").

totp = pyotp.TOTP(user\_secret)

**Note**: the name of the image created is qr\_code.png. Remember this name needs to match the one used in the HTML file.

The line, qr\_code.png(stream, scale=5), allows you to adjust the size of the QR Code

The final line of code is used to encode binary data into printable ASCII characters and decoding such encodings back to binary data.

totp = pyotp.TOTP(user\_secret)

otp\_uri = totp.provisioning\_uri(name=username,issuer\_name="YourAppName")

qr\_code = pyqrcode.create(otp\_uri)

stream = BytesIO()

qr\_code.png(stream, scale=5)

qr\_code\_b64 = base64.b64encode(stream.getvalue()).decode('utf-8')

All that is left is to validate the entry of the one-time passcode and redirect the user to the desired page.

In this section we are retrieving the input from the form using the line of code:

otp\_input = request.form[‘otp’]

We then use the inbuilt function totp.verify to validate the code entered matches the one generated by Google Authenticator app. If the valid code matches, we direct the user to the desired page, as they have successfully logged on using 2FA.

Note, you will need to either created a new some\_page.html page or change the name to the page you wish to display.

You may also wish to create a page which indicates the code entered is invalid as currently it simply displays a plain message to a blank page.

if request.method == 'POST':

otp\_input = request.form['otp']

if totp.verify(otp\_input):

return render\_template('some\_page.html')

#return redirect(url\_for('home')) # Redirect to home if OTP is valid

else:

return "Invalid OTP. Please try again.", 401

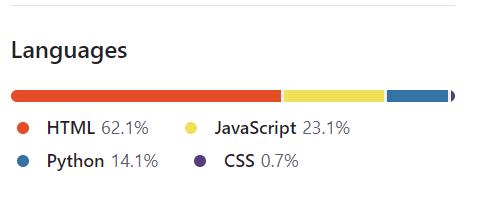
return render\_template(index.html')

**Run your Flask app**

Finally, run the Flask app, and test the newly created page using the Google Authenticator app. Note, you may need to rescan the QRcode every time modifications are made to the code.

# Appendix 4 – secure Flask Bootstrap PWA template

[**Safe Flask Bootstrap PWA Template**](https://github.com/TempeHS/Secure_Flask_PWA_Template/tree/main)



Screenshot from [GitHub](https://github.com/TempeHS/Practical_Approach_to_Secure_Software_Architecture_Resources/tree/main), accessed 25 November 2024.

This is a basic template for creating a safe progressive web app (PWA) using Flask.

It comes with a secure form and example parts from Bootstrap (front-end), which you can easily link to a SQLite3 database (back-end) through the Flask app. The template meets the basic requirements for PWAs, and it includes placeholder images for all the icons and pictures you might need.

Bootstrap is a popular framework used for designing websites and web applications. It provides pre-made styles and components, like buttons, forms, and navigation bars, which help you create a professional-looking interface quickly and easily.

By using Bootstrap in the Flask template, you can focus more on the functionality of your app rather than spending a lot of time on design. It ensures that your app looks good on different devices, like smartphones and tablets, making it responsive.

**Dependencies**

* VSCode or GitHub Codespaces (preconfigured for docker)
* Python 3+
* pip install Flask
* pip install SQLite3
* pip install Bcrypt
* pip install flask\_wtf

**Secure features**

* Meta Content Security Policy declared
* HTML Language declared
* Meta character set declared
* Private folders use.folderName syntax
* No inline <script>
* Bootstrap components are self-contained
* CSRFProtect applied to form
* Form Pattern expression declared

**To be implemented by developers**

* Web content, [Bootstrap](https://getbootstrap.com/) ready

* [SQLite](https://docs.python.org/3/library/sqlite3.html)3 database design and integration
* Input sanitisation
* Login, authentication, or session management
* Password encryption using [Bcrypt](https://pypi.org/project/bcrypt/)
* SSL Encryption

## Advice for developers

**Privacy advice**

* The app should have a privacy handling policy
* Only data essential for the purposes of the app should be collected
* Users should be given the option to download or delete their data
* Passwords should be encrypted, including a salt, before hashing

**Security advice**

* All inputs should be sanitised before processing or storing
* If including login, authentication and session management should be implemented
* SSL Encryption and HTTPS should be implemented
* Use Jinga2 components when passing variables to the frontend
* Use query parameters for all SQL queries

**Content advice**

* All templates are [Bootstrap](https://getbootstrap.com/) ready
* [SQLite](https://docs.python.org/3/library/sqlite3.html)3 has been provided for database design

**Tip**

**Content Security Policy (UNSECURE PWA) note**

Content-Security-Policy is the name of an HTTP response header that modern browsers use to enhance the security of the app. The Content-Security-Policy header allows you to restrict which resources (such as JavaScript, CSS, Images, etc.) can be loaded, and the URLs that they can be loaded from.

Although it is best applied as an HTTP response header, this template has applied it as a meta tag, which is the minimum standard.

**Warning**

* Bootstrap 5.3.3 has been packaged in the template, developers should [monitor for discovered vulnerabilities](https://security.snyk.io/package/npm/bootstrap), and patch or update as needed.

## How to guide

Linking Bootstrap to a SQLite3 database isn't about connecting them directly, as Bootstrap is a front-end framework for styling and layout, while SQLite3 is a back-end database for storing data. However, this can easily be set up in a Flask app.

**Set up Flask and SQLite3**

Ensure Flask and SQLite3 installed.

Use a package manager like pip to install Flask.

**Create the Flask app**

Start by creating a basic Flask app. This is where requests will be handled and interact with the database.

**Connect to SQLite3**

Use the sqlite3 library in Python to connect to the SQLite3 database.

Create or open a database file like this:

import sqlite3

conn = sqlite3.connect('your\_database.db')

cursor = conn.cursor()

**Create database tables**

Define the tables (like users or products) in the database using SQL commands.

cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')

conn.commit()

**Create routes in Flask**

Create routes to handle HTTP requests (like GET and POST). When a user submits a form created with Bootstrap, the data will save to the SQLite database.

@app.route('/submit', methods=['POST'])

def submit():

name = request.form['name'] # Assuming you have a form field named 'name'

cursor.execute('INSERT INTO users (name) VALUES (?)', (name,))

conn.commit()

return redirect('/success')

**Use Bootstrap for front-end**

In the HTML templates, include Bootstrap’s CSS and JavaScript files, a CDN (Content Delivery Network) can be used to easily add these:

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

**Create forms with Bootstrap**

Design forms using Bootstrap classes to make them user friendly and responsive:

<form action="/submit" method="post">

<div class="form-group">

<label for="name">Name:</label>

<input type="text" class="form-control" id="name" name="name" required>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

**Run your Flask app**

Finally, run the Flask app, and when users fill out the Bootstrap form, their data will be saved into the SQLite3 database.

References

This resource contains NSW Curriculum and syllabus content. The NSW Curriculum is developed by the NSW Education Standards Authority. This content is prepared by NESA for and on behalf of the Crown in right of the State of New South Wales. The material is protected by Crown copyright.

Please refer to the NESA Copyright Disclaimer for more information <https://educationstandards.nsw.edu.au/wps/portal/nesa/mini-footer/copyright>.

NESA holds the only official and up-to-date versions of the NSW Curriculum and syllabus documents. Please visit the NSW Education Standards Authority (NESA) website <https://educationstandards.nsw.edu.au> and the NSW Curriculum website <https://curriculum.nsw.edu.au>.

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

[Software Engineering course specifications](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

[Computing Technology 7–10 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/computing-technology-7-10-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in the State of New South Wales, 2022.

Abegg A (2019) [*BCrypt*](https://marketplace.visualstudio.com/items?itemName=aabegg.vscode-bcrypt), Microsoft Visual Studio Marketplace website, accessed 24 October 2024.

ACS (Australian Computer Society) (2024) [*Information Age*](https://ia.acs.org.au/) [website], accessed 24 October 2024.

ASD’s ACSC (Australian Signals Directorate’s Australian Cyber Security Centre) (n.d.) [*ASD’s ACSC*](https://www.cyber.gov.au/) [website], accessed 24 October 2024.

——(n.d.) ‘[Secure-by-Design](https://www.cyber.gov.au/resources-business-and-government/governance-and-user-education/secure-by-design)’, Resources for business and government, ASD’s ACSC website, accessed 24 October 2024.

ASD’s ACSC (2022) ‘[Cloud Security Assessment Report Template (July 2022)](https://www.cyber.gov.au/resources-business-and-government/assessment-and-evaluation-programs/infosec-registered-assessors-program/irap-resources)’, ACSC, ASD, accessed 18 November 2024.

——(2024) ‘[Guidelines for System Hardening](https://www.cyber.gov.au/resources-business-and-government/essential-cyber-security/ism/cyber-security-guidelines/guidelines-system-hardening)’, Cyber Security Guidelines, ASD’s ACSC website, accessed 24 October 2024.

BlueOptima (7 June 2022) ‘[4 Lessons from Software Security Case Studies](https://www.blueoptima.com/4-lessons-from-software-security-case-studies/)’, BlueOptima Articles, accessed 24 October 2024.

Broadcom Inc (2005–2024) ‘[Why Private Cloud](https://www.vmware.com/solutions/cloud-infrastructure/why-private-cloud)’, VMware Cloud Infrastructure, VMware website, accessed 24 October 2024.

Cheat Sheets Series Team (2024) [*Session Management – OWASP Cheat Sheet Series*](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html), Cheat Sheets Series website, accessed 24 October 2024.

CodePen (2024) [*CodePen*](https://codepen.io/) [website], accessed 24 October 2024.

Embrace The Red (13 November 2020) [‘Web Application Security Fundamentals (must know basics for developers, testers and hackers)’ [video]](https://www.youtube.com/watch?v=-7OX58nHPb8), Embrace The Red, YouTube, accessed 5 August 2024.

JSFiddle (n.d.) [*JSFiddle – Code Playground*](https://jsfiddle.net/) [website], accessed 24 October 2024.

Fortinet, Inc (2024) ‘[What is Cryptography?](https://www.fortinet.com/resources/cyberglossary/what-is-cryptography)’, Cyber Glossary, Fortinet website, accessed 24 October 2024.

Fortinet, Inc (2024) [What is Sandboxing?](https://www.fortinet.com/resources/cyberglossary/what-is-sandboxing)’, Cyber Glossary, Fortinet website, accessed 24 October 2024.

Foundation (1998–2024) [*Foundation Framework*](https://get.foundation/) [website], accessed 5 August 2024.

freeCodeCamp.org (9 March 2023) [‘Harvard CS50’s Web Programming with Python and JavaScript – Full University Course’ [video]](https://www.youtube.com/watch?v=vzGllw18DkA), freeCodeCamp.org, YouTube, accessed 5 August 2024.

freeCodeCamp.org (12 April 2023) [‘Bash Scripting Tutorial for Beginners’ [video]](https://www.youtube.com/watch?v=tK9Oc6AEnR4), freeCodeCamp.org, YouTube, accessed 5 August 2024.

freeCodeCamp.org (17 May 2023) [‘JavaScript Security Vulnerabilities Tutorial – With Code Example [video]](https://www.youtube.com/watch?v=ypNKKYUJE5o), freeCodeCamp.org, YouTube, accessed 24 October 2024

Geeksforgeeks (3 June 2022) ‘[Hashing Passwords in Python with BCrypt](https://www.geeksforgeeks.org/hashing-passwords-in-python-with-bcrypt/)’, GeeksforGeeks articles, accessed 24 October 2024.

Google Chrome (16 December 2010) [‘Google Chrome & Security: Sandboxing’ [video]](https://www.youtube.com/watch?v=29e0CtgXZSI), Google Chrome, YouTube, accessed 24 October 2024.

Google LLC (2016) ‘[Introduction to Lighthouse](https://developer.chrome.com/docs/lighthouse/overview/)’, Lighthouse, Chrome for Developers website, accessed 24 October 2024.

Google LLC (2024) [*Google Trends*](https://trends.google.com/home?hl=en-US) [website], accessed 24 October 2024.

Greenbone AG (2020–2024) [*Open Source Vulnerability Management*](https://www.greenbone.net/en/open-source-vulnerability-management/), Greenbone website, accessed 24 October 2024.

Hacksplaining Inc (2024) ‘[Cross-Site Scripting](https://www.hacksplaining.com/lessons/xss-stored/start)’, Lessons, Hacksplaining website, accessed 24 October 2024.

Hitachi Systems Security (14 July 2017) [‘What is Penetration Testing?’ [video]](https://www.youtube.com/watch?v=q2t91jLmh3k), Hitachi Systems Security, YouTube, accessed 24 October 2024.

howtonetwork (18 August 2020) [‘SQL Injection Attacks – Explained in 5 Minutes’ [video]](https://www.youtube.com/watch?v=FHCTfA9cCXs), howtonetwork, YouTube, accessed 24 October 2024.

Huang J (2017) ‘[Login page using Python, Flask and sqlite3 DB](https://gist.github.com/jironghuang/24e0577e58844882604c0013407bf606)’, jironghuang, GitHub website, accessed 24 October 2024.

Imperva (2024) [*Imperva Learning Center*](https://www.imperva.com/learn/), Imperva website, accessed 24 October 2024.

ISO (International Organization for Standardization) (2022) [ISO/IEC 27001:2022 – Information security management](https://www.iso.org/standard/27001), ISO website, accessed 10 April 2024.

Jones B (2024) ‘[The\_Unsecure\_PWA](https://github.com/TempeHS/The_Unsecure_PWA)’, *TempeHS (Tempe High School*), GitHub website, accessed 18 November 2024.

Jones B (2024) ‘[Flask\_PWA\_Programming\_For\_The\_Web\_Task\_Source](https://github.com/TempeHS/Flask_PWA_Programming_For_The_Web_Task_Source)’, TempeHS (Tempe High School), GitHub website, accessed 24 October 2024.

Kost E (16 September 2024) ‘[13 Biggest Data Breaches in Australia [Updated 2024]](https://www.upguard.com/blog/biggest-data-breaches-australia)’, UpGuard blog, accessed 24 October 2024.

Latiyan M (2023) [*Optimizing JavaScript Performance: Memory Management and Code Profiling*](https://www.tutorialspoint.com/optimizing-javascript-performance-memory-management-and-code-profiling), Tutorials Point website, accessed 24 October 2024.

LaunchCode (4 April 2017) [‘Introduction to Object-Relational Mapping’ [video]](https://www.youtube.com/watch?v=dHQ-I7kr_SY), LaunchCode, YouTube, accessed 7 August 2024.

LearnWebCode (11 February 2010) [‘Learn HTML – Your First Web Page (For Absolute Beginners)’ [video]](https://www.youtube.com/watch?v=BBcq9abR9Es), LearnWebCode*,* YouTube, accessed 5 August 2024.

LearnWebCode (12 February 2010) [‘Learn CSS (Tutorial) – Basics + Selectors (For Absolute Beginners)’ [video]](https://www.youtube.com/watch?v=Awmx8XlgcaE), LearnWebCode, YouTube, accessed 5 August 2024.

Lucid Software Inc (2024) [*Decision tree diagram maker*](https://www.lucidchart.com/pages/examples/decision-tree-maker?a=0), Lucidchart website, accessed 19 August 2024.

——(2024)[*How to Make a Decision Tree Diagram*](https://www.lucidchart.com/pages/how-to-make-a-decision-tree-diagram), Lucidchart website, accessed 3 April 2024.

——(2024) [*Lucidchart*](https://www.lucidchart.com/pages/?) [website], accessed 3 April 2024.

MDN contributors (2023) ‘[Progressive web apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps?ref=arctype.com)’, References: Web technology for developers, MDN website, accessed 7 August 2024.

Microsoft (2024) [*Extensions for Visual Studio Code*](https://marketplace.visualstudio.com/vscode), Microsoft Visual Studio Marketplace website, accessed 7 August 2024.

Microsoft (2024) ‘[Windows Sandbox](https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-overview)’, Windows application security, Microsoft Learn website, accessed 24 October 2024.

Mitchell B (16 January 2023) ‘[What Is Penetration Testing? A Beginner’s Guide](https://www.codingdojo.com/blog/what-is-penetration-testing)’, Coding Dojo blog, accessed 24 October 2024.

Mustache (n.d.) [*Mustache*](https://mustache.github.io/) [website], accessed 5 August 2024.

Nash J (7 March 2019) [‘Hashing passwords with Python and Bcrypt’ [video]](https://www.youtube.com/watch?v=hNa05wr0DSA), Julian Nash, YouTube, accessed 24 October 2024.

NESA (NSW Education Standards Authority (2024) ‘[Glossary of key words](https://www.nsw.gov.au/education-and-training/nesa/hsc/student-guide/glossary)’, Student guide, NSW Government website, accessed 18 November 2024.

Nmap (n.d.) [N*map: the Network Mapper – Free Security Scanner*](https://nmap.org/) [website], accessed 24 October 2024.

OAIC (Office of the Australian Information Commissioner) (2024) ‘[Privacy by design](https://www.oaic.gov.au/privacy/privacy-guidance-for-organisations-and-government-agencies/privacy-impact-assessments/privacy-by-design)’, Privacy impact assessments, OAIC website, accessed 24 October 2024.

Oracle (2024) [*Oracle VirtualBox*](https://www.virtualbox.org/) [website], accessed 24 October 2024.

OWASP Foundation, Inc (2024) [*OWASP Foundation, the Open Source Foundation for Application Security*](https://owasp.org/) [website], accessed 24 October 2024.

Password Bits (6 December 2022) [‘What is password hashing? [Simplified]’ [video]](https://www.youtube.com/watch?v=deNIgVyDyJY), Password Bits, YouTube, accessed 24 October 2024.

Python Software Foundation (2001–2024) ‘[8. Errors and Exceptions](https://docs.python.org/3/tutorial/errors.html)’, The Python Tutorial, Python website, accessed 24 October 2024.

Python Software Foundation (2001–2024) ‘[The try statement](https://docs.python.org/3/reference/compound_stmts.html#try)’, The Python Language Reference, Python website, accessed 24 October 2024.

Rapid7 (n.d.) ‘[Nexpose Vulnerability Scanner](https://www.rapid7.com/products/nexpose/)’, Cybersecurity Products, Rapid7 website, accessed 24 October 2024.

Replit, Inc (2024) [*Replit – Build software faster*](https://replit.com/) [website], accessed 24 October 2024.

Sharma N (9 April 2024) ‘[Session Management 101: A Beginner's Guide for Web Developers](https://mojoauth.com/blog/session-management-a-beginners-guide-for-web-developers/)’, MojoAuth blog, accessed 24 October 2024.

Sluiter S (8 August 2019) [‘How Equifax was Hacked’ [video]](https://www.youtube.com/watch?v=MKaNxE7pGWA), Programming w/ Professor Sluiter, YouTube, accessed 24 October 2024.

State of New South Wales (Department of Education) (n.d.) [*Digital Learning Selector*](https://app.education.nsw.gov.au/digital-learning-selector/), NSW Department of Education website, accessed 5 August 2024.

State of New South Wales (Department of Education) (2024) ‘[Connecting learning](https://education.nsw.gov.au/teaching-and-learning/curriculum/explicit-teaching/explicit-teaching-strategies/connecting-learning)’, Explicit teaching strategies, NSW Department of Education website, accessed 5 August 2024.

Stichting Cuckoo Foundation (2024) [*Cuckoo Sandbox Automated Malware Analysis*](https://cuckoosandbox.org/) [website], accessed 24 October 2024.

TechKnowSurge (13 July 2024) [‘Foundations of Cybersecurity 2–6: Race Conditions’ [video]](https://www.youtube.com/watch?v=mq59FxItJ2w), TechKnowSurge,YouTube, accessed 24 October 2024.

Watson M (7 April 2017) ‘[Software Error vs Exception – In Real World Examples](https://stackify.com/application-exception-vs-error-difference/)’, Stackify blog, accessed 24 October 2024.

WittCode (27 March 2023) [‘JavaScript Memory Management’ [video]](https://www.youtube.com/watch?v=gOeW5_UnN9g), WittCode, YouTube, accessed 24 October 2024

Wiz Experts Team (7 June 2024) ‘[What is Security by Design?](https://www.wiz.io/academy/security-by-design)’, Wiz articles, accessed 24 October 2024.

Yang LL (8 May 2021) [‘OWASP Zap Tutorial’ [video]](https://www.youtube.com/watch?v=_VpFaqF0EcI), Loi Liang Yang, YouTube accessed 24 October 2024

ZAP by Checkmarx (2024) [*Getting Started*](https://www.zaproxy.org/getting-started/), ZAP website, accessed 24 October 2024.

## Legislation

Privacy and Personal Information Protection Act 1998 No 133

**© State of New South Wales (Department of Education), 2024**

The copyright material published in this resource is subject to the Copyright Act 1968 (Cth) and is owned by the NSW Department of Education or, where indicated, by a party other than the NSW Department of Education (third-party material).

Copyright material available in this resource and owned by the NSW Department of Education is licensed under a [Creative Commons Attribution 4.0 International (CC BY 4.0) license](https://creativecommons.org/licenses/by/4.0/).

[](https://creativecommons.org/licenses/by/4.0/)

This license allows you to share and adapt the material for any purpose, even commercially.

Attribution should be given to © State of New South Wales (Department of Education), 2024.

Material in this resource not available under a Creative Commons license:

* the NSW Department of Education logo, other logos and trademark-protected material
* material owned by a third party that has been reproduced with permission. You will need to obtain permission from the third party to reuse its material.

**Links to third-party material and websites**

Please note that the provided (reading/viewing material/list/links/texts) are a suggestion only and implies no endorsement, by the New South Wales Department of Education, of any author, publisher, or book title. School principals and teachers are best placed to assess the suitability of resources that would complement the curriculum and reflect the needs and interests of their students.

If you use the links provided in this document to access a third-party's website, you acknowledge that the terms of use, including licence terms set out on the third-party's website apply to the use which may be made of the materials on that third-party website or where permitted by the Copyright Act 1968 (Cth). The department accepts no responsibility for content on third-party websites.