**Checklist for TASK 2 Deliverables**

**Functional Code Implementation**

* **Develop program code as a prototype that**

Is written in at least two different programming languages.

Is mostly efficient, with only minor errors.

Implements consistently efficient functional code throughout.

Uses precise logic and programming structures to produce mostly or consistently correct outcomes.

**Program Documentation**

* **Produce a README file that includes**

An overview of the program's purpose and functionality.

Clear instructions on how to run the program.

A summary of dependencies and system requirements.

Notes on any known limitations or areas for improvement.

**Code Maintainability**

* **Ensure the program code is easily maintainable by providing**

Appropriate naming conventions.

Logical organisation of code.

Informative comments explaining key sections of code.

**User Experience**

* **Create a program with a good user experience, including**

Effective input handling.

Clear user guidance and error messages.

Meaningful outputs.

*Ensure the program is*

Largely robust, handling most common and unexpected errors.

Fully robust, handling all common and unexpected errors effectively.

**Standards and Guidelines**

* **Apply standards and guidelines consistently and effectively for**

**Accessibility**.

* + *Text and Contrast*

Ensure readable text with high contrast using basic colour schemes. Example, use black text on a white background for clarity.

* + *Keyboard Navigation*

Ensure all interactive elements (e.g., buttons, links) are accessible via the "Tab" key. Example, add a focus outline to clearly show active elements.

* + *Alt Text for Images*

Add descriptive alt text for any images in the prototype.

**Compatibility**.

* + *Responsive Design*

Use simple CSS rules to ensure the layout adjusts to different screen sizes

* + *Browser Testing*

Test the prototype in at least two common browsers (e.g., Chrome and Edge).

Document any issues or ensure that the prototype runs as expected.

* + *Device Testing*

Ensure the interface works on both desktop and mobile devices.

Test by resizing the browser window.

**Legal and ethical considerations.**

* + *Data Privacy*

If data collection is required, display a simple message explaining why.

Only collect data that is required.

* + *Copyright Compliance*

Use royalty-free assets or clearly document the source of all external resources in an asset log.

Example of an asset log entry

Image: "example.jpg" – Source: Unsplash (License: Free to Use).

* **Implement procedures and security controls to ensure**

**Confidentiality.**

* *Data Encryption*

Encrypt sensitive data stored within the application (e.g., passwords, user information).

Use HTTPS for secure communication between the application and server.

* *Access Control*

Implement user authentication (e.g., username and password or multi-factor authentication).

Restrict access to certain program features or data based on user roles (e.g., admin vs. user).

* *Data Masking*

Mask sensitive information in user interfaces (e.g., show only the last 4 digits of a credit card).

**Integrity.**

* + *Input Validation*

Validate all user inputs to prevent incorrect or malicious data from being entered.

Use data type checks, length validation, and pattern matching for inputs.

* *Error Handling*

Implement error-handling routines that log issues without exposing sensitive information.

Avoid application crashes caused by unexpected inputs or events.

* *Hashing*

Hash sensitive data (e.g., passwords) before storing them in the database. Use strong algorithms like SHA-256 or bcrypt.

* *Database Constraints*

Use primary keys, foreign keys, and other database constraints to ensure data relationships remain intact.

**Availability.**

* *Error Recovery basic Implementation*

Include try-catch blocks to handle unexpected errors gracefully without crashing the program.

Provide meaningful error messages to guide users if something goes wrong. Example, if a file fails to load, notify the user and suggest a retry.

* *Backups*

Create a simple backup mechanism for critical data (e.g., saving a copy of user input or outputs).

Use a local file system to store a backup file.

* *Simple Health Checks*

Create a simple status check to verify critical components (e.g., database connection, file existence). Example, display a "System Ready" message after verifying all required files are present.

* *Prevent Crashes*

Use basic validation to ensure the prototype can handle invalid user inputs or unexpected conditions. Example, check for null or empty inputs before processing.

**Testing**

* Create a testing log that demonstrates

Thorough and detailed testing with

Normal data.

Erroneous data.

Extreme data.

Evidence of testing inputs, calculations, validation, and processes.

**Error Identification and Resolution**

* Provide comments that explain

How errors/problems were identified.

How errors were rectified for inputs, calculations, validation, and processes.

Include evidence of an iterative development process.

**Iterative Development Process**

* Provide thorough documentation of development, including

Notable changes made throughout the process.

Rationales for notable changes.

Evidence of fully effective version control.

**Additional Documentation**

* Maintain an asset log that includes

A list of all assets used in the program (e.g., images, libraries, datasets).

Sources and licensing information for each asset.

Any modifications made to the assets.

**Final Deliverables**

* Ensure the following are submitted

Program code (prototype).

README file with clear documentation.

Testing log with detailed test cases and results.

Evidence of version control, including version history and notes on changes.

Asset log documenting all external resources.

**Notes**

* All deliverables must meet the outlined standards and demonstrate clear, effective implementation.