Template Overview

In Task 2, you are developing a working web-based system using HTML, CSS, JavaScript, and SQL through Supabase, built and managed in Visual Studio Code and hosted using GitHub Pages.

Your system must allow users to enter, view, update, and delete data stored in a Supabase database. This data must be handled safely, validated properly, and displayed clearly on the website. Your website should behave like a real application, not a static page.

The examiner will not just look at whether your code exists. They will look at **how your system behaves**, **how your code is structured**, **how well it works when hosted**, and **how clearly you show testing and fixes**.

The expected flow of your project is:

HTML form → JavaScript logic → Supabase database (SQL) → JavaScript processing → Website output → GitHub Pages hosted version, with evidence.

To reach higher marks, your work must show that:

- JavaScript correctly connects to Supabase.
- CRUD functionality (Create, Read, Update, Delete) is implemented and works.
- The hosted version behaves the same as your local version.
- Errors are handled properly, not ignored.
- Your system looks and feels usable to a real user.

Grade outcomes:

- A basic, partly working system may reach a **Pass**.
- A mostly working system with clear structure and testing can reach a **Merit**.
- A fully working, well-tested, professional-looking system is needed for a **Distinction**.

Important Notes for Students

This document is your **working guide during Task 2**. It is not just something to read once. You should return to it **every day** to check what evidence you are producing.

You are expected to **document as you build**, not after everything is finished. Screenshots, testing notes, and explanations should be created alongside your code.

A common reason students lose marks is not because their code is wrong, but because:

- The hosted version does not work properly.
- CRUD is only partly implemented.
- Errors appear but are not handled.
- Testing only uses “perfect” data.
- Fixes are made but not recorded or re-tested.

To reach a **Pass (Band 2)**, your system must show that JavaScript connects to Supabase and at least some CRUD actions work, even if there are issues.

To reach a **Merit (Band 3)**, most features must work correctly, CRUD should be nearly complete, the site should function when hosted, and testing should include invalid, blank, and extreme data.

To reach a **Distinction (Band 4)**, your system must:

- Fully implement CRUD with no major bugs.
- Work cleanly on GitHub Pages.
- Use correct Supabase table structures and column names.
- Show strong testing, fixes, and re-tests.
- Be organised, readable, and professional, with no secret keys exposed.

You are not expected to rush. You are expected to work **methodically**, test often, and record what you do. Marks are awarded for **evidence of correct process**, not just finished code.

Time Breakdown Overview

- You have **30 hours** to complete Task 2. Use the table below to track your progress.
- Tick items **only when evidence exists** (code, screenshots, testing entries, or commits).
- Do not move on until the day's core tasks are complete.

Day 1–2: Project Setup and Foundations

Task	Tick
Visual Studio Code project folder created	
Folder structure set up (docs/, static/, index.html, README.md)	
GitHub repository created and linked to project	
GitHub Pages enabled and live	
HTML structure created using semantic tags (header, main, section, footer)	
CSS file created and linked correctly	
JavaScript file created and linked correctly	
Initial README.md created (project purpose + live link)	
testing-log.md created (even if empty)	
GitHub commit showing setup progress	

Band signal: Completing this properly supports **Band 2** and prevents early Band 1 issues.

Day 3: Supabase and Data Handling

Task	Tick
Supabase project created	
Correct table(s) created using SQL (DDL)	
Column names checked carefully against JS	
Sample data inserted (INSERT statements)	
Supabase URL and public anon key added to app.js	
JavaScript successfully reads data from Supabase	
At least two HTML forms created	
Validation added (JS and/or SQL constraints)	
Testing logged for ADD actions	
GitHub commit showing Supabase integration	

Band signal: Working Supabase + JS integration moves you towards **Band 2 → Band 3**.

Day 4–5: Core Functionality and Quality

Task	Tick

CRUD partially implemented (more than just ADD)	
EDIT and DELETE functionality started	
JavaScript functions organised and readable	
Errors handled with messages (not silent failures)	
CSS refined for layout and clarity	
Accessibility considered (labels, contrast, readable text)	
W3C-compliant HTML, CSS, and JS maintained	
Testing expanded beyond perfect data	
Bugs recorded and fixed in testing-log.md	
GitHub commits show steady progress	

Band signal: Consistent functionality plus testing supports **Band 3 (Merit)**.

Day 6: Full CRUD, Testing, and Debugging

Task	Tick
Full CRUD working (Create, Read, Update, Delete)	
Hosted version behaves the same as local version	
EDIT and DELETE tested on at least two tables	
Normal, invalid, blank, extreme data tested	
At least 30 test cases recorded	
Bugs fixed and re-tested clearly	
Console errors resolved	
No secret keys exposed anywhere	
GitHub commit shows final functionality	
README.md fully completed and clear	
Screenshots collected for all sections	
Testing-log.md complete and tidy	
Code commented and readable	
Accessibility checked again	
GitHub Pages link tested one final time	
Final commit made	

Band signal: This is the minimum expectation for **Band 4 (Distinction)** functionality and testing. Strong documentation and polish secure **Distinction-level evidence**.

Key Reminder for Students

You are marked on:

- How well your system works
- How clearly you show testing and fixes
- How professional your process looks

A rushed system with weak evidence will not score highly, even if some code works.

A calm, structured approach with steady evidence collection is the safest way to succeed.

2. Functionality (Core System Behaviour)

This section is about whether your system **actually works as a real application**.

The examiner is looking for a **clear flow of data**:

HTML form → JavaScript → Supabase (SQL) → JavaScript → Website output → Hosted version

Your system must use **JavaScript to communicate with Supabase** and perform **CRUD operations** on real data. Static pages or hard-coded data will not score highly.

What You Are Being Marked On (In Simple Terms)

- Does your website load and function correctly?
- Does JavaScript successfully connect to Supabase?
- Can users **Create, Read, Update, and Delete** data?
- Does the hosted version work properly?
- Are errors handled and tested?

Band Expectations (Read This Carefully)

Band 2 (Pass level): Some features work. JavaScript connects to Supabase at least partially. One or two CRUD actions may work, but others may be missing or unreliable. Errors may still appear.

Band 3 (Merit level): Most features work. CRUD is nearly complete. Supabase tables are set up correctly. The site works locally and mostly when hosted. Only small issues remain.

Band 4 (Distinction level): Everything works smoothly. Full CRUD is implemented and reliable. The hosted version works perfectly. The system behaves like a real application, not a demo .

Evidence You Must Collect

You must include:

- Screenshots of working functionality
- Code snippets showing JavaScript and Supabase interaction
- Testing evidence showing success and failure cases
- Proof the hosted version works (GitHub Pages)

2.1 JavaScript Setup and Supabase Connection

Task: Set up your JavaScript file so it connects correctly to your Supabase project using the **public anon key only**.

Your JavaScript must:

- Store the Supabase URL and anon key correctly
- Initialise the Supabase client
- Successfully connect without console errors

Evidence to Show

- Screenshot of `app.js` showing Supabase connection setup
- Screenshot of browser console with no connection errors
- Screenshot of hosted version loading correctly

Band Signal

- Band 2: Connection exists but errors may appear
- Band 3: Connection works reliably
- Band 4: Clean connection, no errors, hosted version works

2.2 HTML Structure and User Input

Task: Create HTML forms that allow users to input and interact with data.

Forms must:

- Be linked correctly to JavaScript

- Use appropriate input types
- Include labels for accessibility

Evidence to Show

- Screenshot of forms in the browser
- Screenshot of linked HTML and JS code

Band Signal

- Band 2: Forms exist but may be basic
- Band 3: Forms work reliably and send data correctly
- Band 4: Forms are clear, accessible, and user-friendly

2.3 Data Validation and Error Handling

Task: Validate user input before it reaches the database.

Validation may be done using:

- JavaScript checks
- SQL constraints in Supabase
- Or a combination of both

Your system must:

- Reject invalid or empty data
- Display helpful error messages
- Avoid crashes

Evidence to Show

- Screenshots of validation errors
- Code showing validation logic
- Testing-log entries for invalid and blank data

Band Signal

- Band 2: Some validation present

- Band 3: Most invalid data handled correctly
- Band 4: Validation is thorough and reliable

2.4 Supabase Database Interaction (CRUD)

Task: Use JavaScript to interact with Supabase tables using SQL-backed operations.

You must demonstrate:

- **Create** new records
- **Read** and display existing records
- **Update** existing records
- **Delete** records

Column names must exactly match your JavaScript code.

Evidence to Show

- Screenshot of Supabase table structure
- Code snippets for each CRUD operation
- Screenshots showing data changing correctly

Band Signal

- Band 2: One or two CRUD actions work
- Band 3: Most CRUD actions work
- Band 4: Full CRUD works smoothly and reliably

2.5 Hosted Functionality (GitHub Pages)

Task: Ensure your system works when hosted, not just locally.

The examiner will check:

- The live GitHub Pages URL
- Whether JavaScript, CSS, and data loading work online
- Whether Supabase interactions still function

Evidence to Show

- Screenshot of live hosted site
- Screenshot of CRUD working on hosted version
- Testing evidence specifically for hosted behaviour
- Links to the project and project files on GitHub available for the examiner.

Band Signal

- Band 2: Hosted version partly works
- Band 3: Hosted version mostly works
- Band 4: Hosted version works perfectly

2.6 Functional Behaviour and Reliability

Task: Your system should behave consistently and predictably.

This includes:

- Clear success messages
- Clear error messages
- No unexplained failures
- Logical flow between actions

Evidence to Show

- Screenshots of successful actions
- Screenshots of handled errors
- Testing-log entries showing fixes and re-tests

Band Signal

- Band 3 and 4 depend heavily on reliability and consistency

Final Reminder for Functionality

If your system:

- Only works locally
- Only adds data
- Crashes with bad input
- Has missing CRUD actions

Then it cannot reach the highest bands, even if the idea is good.

A simple system that works perfectly will score higher than a complex system that only partly works.

3. Code Organisation (How Professional Your Work Looks)

This section is about whether your project looks like it was built by a **real developer**, not just whether it works.

The examiner will look at:

- How your files are structured in Visual Studio Code
- How readable and organised your code is
- Whether your progress is clearly shown using GitHub
- Whether someone else could understand your project by reading it

Good organisation makes your functionality easier to follow and helps protect higher marks in other sections.

What the Examiner Expects to See First

Your project **must follow a clear structure** in Visual Studio Code.

A typical structure looks like this:

- `docs/`
  - `schema.sql` – SQL used to create and populate Supabase tables
  - `testing-log.md` – record of all testing, fixes, and re-tests
- `static/`
  - `app.js` – JavaScript logic and Supabase interaction
  - `style.css` – CSS styling
- `index.html` – main HTML page
- `README.md` – project explanation and live link

This structure is shown in the marking guidance and is what examiners expect to navigate when marking .

Band Expectations (Read Carefully)

Band 2 (Pass level)

Files are separated into HTML, CSS, and JavaScript. Some comments exist. GitHub shows some commits. A basic README.md is present.

Band 3 (Merit level)

3.1 File Structure and Separation

Task: Organise your project so each file has a **clear purpose**.

You must:

- Keep HTML, CSS, and JavaScript in separate files
- Use folders to group related files
- Avoid putting all code into one file

Evidence to Show

- Screenshot of your VSC folder structure
- Screenshot of files open in the correct locations

Band Signal

- Band 2: Files separated but structure may be basic
- Band 3: Clear folder structure used consistently
- Band 4: Structure is logical, tidy, and professional

3.2 JavaScript Organisation and Readability

Task: Your JavaScript must be easy to read and follow.

This includes:

- Using meaningful variable and function names
- Grouping related logic together

- Avoiding long, messy code blocks

JavaScript should clearly show:

- Supabase connection
- CRUD operations
- Validation logic
- User feedback handling

Evidence to Show

- Screenshot of organised JavaScript code
- Example functions with clear names

Band Signal

- Band 2: JavaScript works but may be untidy
- Band 3: JavaScript is clear and mostly well structured
- Band 4: JavaScript is modular, readable, and professional

3.3 Code Comments

Task: Add comments to explain **why** code exists, not just what it does.

Comments should be used to:

- Explain complex logic
- Clarify Supabase queries
- Describe validation rules
- Support someone reading your code for the first time

Do not comment every line. Comment where understanding matters.

Evidence to Show

- Screenshot highlighting commented sections of code

Band Signal

- Band 2: Some comments present
- Band 3: Comments are helpful and consistent
- Band 4: Comments clearly explain intent and logic

3.4 Naming Conventions

Task: Use clear and consistent names for:

- Files
- Variables
- Functions
- Supabase columns

Names should:

- Describe purpose clearly
- Match exactly between JavaScript and Supabase
- Avoid spaces and unclear abbreviations

Evidence to Show

- Screenshot of well-named variables and functions
- Screenshot of Supabase column names matching JavaScript

Band Signal

- Band 3 and 4 rely heavily on clear naming

3.5 Version Control and Commit Quality

Task: Use GitHub to show how your project developed over time.

You must:

- Commit regularly (at least twice a day during the exam period for Task 2).
- Use meaningful commit messages
- Show progress, not one final upload

Avoid commit messages like “update” or “final”.

Evidence to Show

- Screenshot of GitHub commit history
- Example of a clear commit message

Band Signal

- Band 2: Some commits exist
- Band 3: Regular commits show progress
- Band 4: Many clear, meaningful commits show professional workflow .

3.6 README.md Quality

Task: Your README.md must explain your project clearly.

It should include:

- What the project does
- How to use it
- How to set it up
- The live GitHub Pages link

The examiner may read this first.

Evidence to Show

- Screenshot of completed README.md

Band Signal

- Band 2: Basic README exists

- Band 3: README explains the project clearly
- Band 4: README is professional and complete

3.7 Coding Standards and Best Practice

This section focuses on whether your code follows **recognised standards and best practices** expected of a professional developer.

Standards are not about making the site look better. They are about ensuring your code is **consistent, readable, maintainable, and safe**.

What Is Expected

You are expected to follow recognised standards when writing:

- **HTML** (structure and semantics)
- **CSS** (layout, clarity, and maintainability)
- **JavaScript** (clarity, predictability, and safety)
- **SQL** (clear structure and data integrity)

These standards should be applied consistently across your project.

HTML, CSS, and JavaScript Standards

Your front-end code should:

- Use semantic HTML elements appropriately
- Avoid deprecated or non-standard syntax
- Keep CSS and JavaScript in external files
- Follow W3C-compliant patterns for structure and behaviour
- Avoid unnecessary inline styles or scripts

These practices make your code easier to read, debug, and extend.

SQL and Database Best Practice

Although SQL is not governed by W3C, it has recognised best practices which you are expected to follow.

Your database design should:

- Use clear, meaningful table and column names
- Use appropriate data types
- Apply NOT NULL constraints where required
- Use a primary key
- Match column names exactly between Supabase and JavaScript
- Avoid storing sensitive information unnecessarily

This helps prevent errors, protects data integrity, and supports security requirements later in the task.

Evidence to Show

You should include:

- Screenshot of clean, standards-compliant HTML, CSS, and JavaScript
- Screenshot of SQL schema (for example, `schema.sql`)
- Brief explanation of how standards were followed
- Confirmation that no deprecated or unsafe practices were used
- Ensure you update your commit on GitHub specific to W3C standard update (used for later).

Band Expectations

- **Band 2 (Pass level):** Some standards followed, but inconsistently. Code works but may be untidy.
- **Band 3 (Merit level):** Standards followed consistently across most of the project. Code is readable and predictable.
- **Band 4 (Distinction level):** Standards followed throughout. Code looks professional, deliberate, and well structured. Database design supports reliability and safety.

Final Reminder for Code Organisation

- Messy code, missing files, weak commits, or no README can **limit your marks**, even if your system works.
- Clear structure, readable code, and visible progress make it easier for the examiner to award higher bands.

4. User Experience (How Easy and Accessible Your System Is to Use)

This section focuses on how your system feels to a **real user**.

The examiner is looking at whether your website is:

- Clear and easy to understand
- Accessible to different users
- Consistent and predictable
- Helpful when something goes wrong

A system can work correctly but still score poorly here if it is confusing, inaccessible, or frustrating to use.

What You Are Being Marked On

- Layout and navigation
- Visual clarity and consistency
- User feedback (success and error messages)
- Accessibility and inclusive design
- Behaviour of the hosted version

User Experience is not about complexity. It is about **clarity and care**.

Band Expectations (Very Important)

- **Band 2 (Pass level):** The site is usable but basic. Layout works, some styling is present, and simple messages appear. Some parts may feel unclear.
- **Band 3 (Merit level):** The site is clear and easy to use. Layout is consistent. Users receive helpful feedback. The site works well on desktop and at least one other screen size.
- **Band 4 (Distinction level):** The site is fully accessible and polished. All inputs are labelled. Colours are readable. Navigation is obvious. The site works cleanly across desktop, tablet, and mobile when hosted .

4.1 Layout, Navigation, and Consistency

Task: Design your site so users can understand it without instructions.

Your site should:

- Use a consistent layout across pages
- Place navigation in a predictable location
- Use clear headings and sections
- Avoid cluttered or confusing layouts

Users should always know:

- Where they are
- What to do next
- How to return or continue

Evidence to Show

- Screenshots of multiple pages
- Evidence of consistent navigation and layout

Band Signal

- Band 2: Layout works but may be basic
- Band 3: Layout is clear and consistent
- Band 4: Layout is intuitive and professional

4.2 User Feedback and Messages

Task: Your system must communicate clearly with users.

This includes:

- Success messages when actions work

- Error messages when something goes wrong
- Clear instructions near inputs

Messages should:

- Be written in plain language
- Explain what went wrong
- Explain what the user should do next

Avoid technical or vague messages.

Evidence to Show

- Screenshots of success messages
- Screenshots of handled errors
- Testing-log entries referencing feedback

Band Signal

- Band 2: Some feedback exists
- Band 3: Feedback is helpful and clear
- Band 4: Feedback is consistent and user-focused

4.3 Accessibility and Inclusive Design

Task: Ensure your site can be used by as many people as possible.

You must apply accessibility principles, including:

- Labels correctly linked to inputs
- Clear, readable text
- Sufficient colour contrast
- Logical heading order
- Keyboard-friendly interaction

These improvements are based on recognised accessibility guidance and W3C principles.

Evidence to Show

- Screenshot of labelled form inputs
- Screenshot showing readable layout and contrast
- Brief explanation of accessibility choices

Band Signal

- Band 2: Some accessibility features present
- Band 3: Accessibility considered across the site
- Band 4: Accessibility is thorough and deliberate

4.4 Responsive Behaviour and Hosting

Task: Your site must behave properly when hosted on GitHub Pages.

The examiner will check:

- CSS loads correctly
- JavaScript loads correctly
- Layout adapts to different screen sizes
- Nothing breaks on the hosted version

You should test:

- Desktop
- At least one other screen size (tablet or mobile)

Evidence to Show

- Screenshot of hosted site
- Screenshot of site at different screen widths
- Testing-log entries for hosted testing

Band Signal

- Band 2: Hosted site partly works

- Band 3: Hosted site mostly works and adapts
- Band 4: Hosted site works perfectly across devices

Final Reminder for User Experience

User Experience is often where:

- Easy marks are lost
- Distinction marks are secured

You do not need advanced design.

You need **clear layout, helpful feedback, and accessible choices**.

A simple, friendly system will always score higher than a complex but confusing one.

5. Security, Legal and Standards (Safe, Responsible Development)

This section focuses on whether your system has been built **responsibly and safely**, with care taken to protect data, users, and the project itself.

The examiner is looking for clear evidence that you understand:

- How to protect user data
- How to avoid common security mistakes
- How to follow basic legal and accessibility expectations
- How to apply standards in a sensible way

This section is not about adding complex security features. It is about **good judgement and safe practice**.

What You Are Being Marked On

- Safe handling of Supabase keys and data
- Input validation and protection against misuse
- Awareness of privacy and GDPR principles
- Use of recognised coding and data standards
- Evidence that unsafe practices were avoided

Band Expectations (Very Important)

- **Band 2 (Pass level):** Some awareness of safety and standards. Inputs are validated. Basic accessibility features are present.
- **Band 3 (Merit level):** Most safety measures are in place. Supabase is used correctly. Privacy considerations are clear. Code follows recognised best practices.
- **Band 4 (Distinction level):** Strong and consistent safety throughout. Only safe keys are used. No sensitive data is exposed. Clear evidence of responsible, professional development .

5.1 Supabase Security and API Key Handling

Task: Ensure Supabase is used safely and correctly.

You must:

- Use **only the public anon API key**
- Never upload secret keys to GitHub
- Avoid exposing sensitive information in JavaScript or SQL
- Ensure Supabase rules and table access are appropriate

Using the wrong key is a major error and will limit marks.

Evidence to Show

- Screenshot of `app.js` showing anon key usage
- Confirmation that no secret keys appear anywhere

- Screenshot of Supabase project settings if appropriate

Band Signal

- Band 2: Supabase connected but safety may be basic
- Band 3: Supabase used correctly and safely
- Band 4: No unsafe keys or data exposed anywhere

5.2 Input Validation and Data Protection

Task: Protect your system from invalid or harmful data.

You must:

- Validate user input using JavaScript and/or SQL constraints
- Prevent empty, invalid, or extreme values being stored
- Ensure the system does not crash or behave unpredictably

This protects both the system and the data.

Evidence to Show

- Screenshots of validation rules
- Screenshots of rejected input
- Testing-log entries showing invalid and extreme data tests

Band Signal

- Band 2: Some validation present
- Band 3: Most invalid data handled correctly
- Band 4: Validation is thorough and reliable

5.3 Privacy and GDPR Awareness

Task: Demonstrate awareness of basic data protection principles.

You should:

- Only collect data that is necessary
- Avoid storing unnecessary personal information
- Handle user data responsibly
- Be aware that data belongs to the user

You are not expected to be a legal expert. You are expected to show common-sense awareness.

Evidence to Show

- Short written explanation of how data is handled responsibly
- Evidence that unnecessary personal data is not collected

Band Signal

- Band 3 and 4 depend on clear privacy awareness

5.4 Standards and Professional Practice

Task: Apply recognised standards consistently across your project.

This includes:

- W3C-informed practices for HTML, CSS, and JavaScript
- SQL best practices for table design and data integrity
- Consistent naming and structure
- No deprecated or unsafe approaches

These standards support:

- Safety
- Maintainability
- Professional quality

Evidence to Show

- Screenshot of clean HTML, CSS, and JS
- Screenshot of SQL schema
- Brief explanation of standards followed

Band Signal

- Band 2: Some standards followed
- Band 3: Standards followed consistently
- Band 4: Standards applied deliberately and professionally

Final Reminder for Security, Legal and Standards

- You do not gain marks for being complex.
- You gain marks for being **careful, sensible, and responsible**.
- One exposed secret key or unsafe practice can cap your marks, even if everything else works.
- Think like a professional developer, not just a coder.

6. Testing (Proving That Your System Works)

This section focuses on how well you have **tested your system and recorded the results**.

The examiner is not only interested in whether your system works. They want to see:

- What you tested
- What went wrong
- What you fixed
- What you re-tested

Good testing shows control, care, and professional thinking.

What You Are Being Marked On

- Range of test data used
- Clarity of test records
- Evidence of bugs and fixes
- Re-testing after changes
- Testing of the hosted version

Testing evidence must be clear and written as you go, not added at the end.

Band Expectations (Very Important)

- **Band 2 (Pass level):** Basic testing carried out. Mostly normal data used. Limited evidence of fixes.
- **Band 3 (Merit level):** Good range of tests used, including invalid and blank data. Bugs are recorded and fixed.
- **Band 4 (Distinction level):** Full testing carried out, including hosted testing. Errors are logged, fixed, and re-tested clearly. Validation and database constraints are tested thoroughly.

6.1 Test Planning and Test Data

Task: Plan and carry out testing using a range of data types.

You must test:

- Normal data
- Invalid data
- Blank or null data
- Extreme or boundary values

Testing must reflect how real users might misuse the system.

Evidence to Show

- Completed testing-log.md
- Evidence of multiple test types

Band Signal

- Band 2: Mostly normal data tested
- Band 3: Normal, invalid, and blank data tested
- Band 4: Normal, invalid, blank, and extreme data tested

6.2 Recording Test Results

Task: Record each test clearly and consistently.

Each test record should include:

- Test number
- Input used
- Expected outcome
- Actual outcome
- Fix made (if required)
- Re-test result

Tests should be written so someone else can understand what was checked.

Evidence to Show

- Screenshot of testing-log.md
- Clear table entries showing outcomes

Band Signal

- Band 3 and 4 depend on clear, readable test records

6.3 Debugging and Fault Resolution

Task: Show evidence of problems and how you resolved them.

This includes:

- Console errors
- Logic errors
- Validation failures
- Hosting issues

You must show:

- The problem
- The fix
- The successful re-test

Simply fixing issues without recording them will limit marks.

Evidence to Show

- Screenshots of errors
- Screenshots of fixes
- Updated test results

Band Signal

- Band 2: Few fixes shown

- Band 3: Bugs fixed and recorded
- Band 4: Bugs fixed, re-tested, and verified

6.4 Hosted Version Testing

Task: Test the GitHub Pages version of your site.

You must confirm:

- JavaScript loads correctly
- CSS loads correctly
- Supabase interaction still works
- CRUD still functions

Local testing alone is not enough for higher marks.

Evidence to Show

- Screenshots of hosted site
- Testing-log entries labelled as “hosted test”

Band Signal

- Band 3: Hosted version tested
- Band 4: Hosted version tested thoroughly and reliably

6.5 Performance and Reliability Checks (Good Evidence)

This section is about *showing the examiner you have thought about how your site behaves under repeated use and how stable it is when hosted*.

You do NOT need to do expensive or professional load testing. Instead, you will perform **lightweight checks** that are realistic for a simple web app and capture clear evidence.

Why this matters

Even if your functionality works, the examiner can award extra confidence marks when they see that:

- Your hosted site loads reliably
- Repeated actions still behave correctly
- You have documented any issues and re-tested after fixes

This will help support **higher bands (Merit & Distinction)** in Testing and User Experience.

6.5.1 Repeated CRUD Action Testing

What to do

Perform many create, read, update, and delete actions in one session to check your system's behaviour under repeated use.

You should try:

- Adding **20 records in a row**
- Editing **10 records in a row**
- Deleting **10 records in a row**
- Refreshing the hosted page after these actions

Do this *on the hosted version* (GitHub Pages), not just locally.

How to record evidence

1. Screenshot the list of records **before** testing begins
2. Screenshot the list of records **after** repeated actions
3. For each action type, record:
    a. What you did (e.g. "Added 20 records quickly")
    b. What happened (any errors?)
4. Note any broken behaviour and how you fixed it

Where to record it

- Add these notes and screenshots to your **testing-log.md**

Example entry format

```
Test 6.5.1 – Repeated CRUD
Action: Add 20 records rapidly
Expected: All records added correctly
Actual: 19 records added, 1 error on #15
Fix: Adjusted validation to prevent blank field
Re-test Result: All 20 added successfully
```

6.5.2 Simple Performance Check (Page Load)

You can use free online tools to check if your hosted site loads quickly and reliably.

Recommended free tool

- **WebPageTest** — https://webpagetest.org/

Short instructions

1. Open your hosted GitHub Pages link (e.g. `https://yourusername.github.io/yourproject/`)
2. Go to https://webpagetest.org/
3. Paste your hosted site URL
4. Run a **Quick Test** on "Desktop" and another on "Mobile"
5. Wait for the results (usually ~30–60 seconds)

What results to record

- Overall load time
- Largest contentful paint (LCP)
- Any warnings about resources

You do NOT need perfect scores. The aim here is **evidence that you checked performance**, not that you became a performance expert.

How to record evidence

- Screenshot the top part of your WebPageTest results page
- Add a short comment in your testing log:

```
Test 6.5.2 – Page Load (Desktop)
Load time: 1.8s
Notes: Site loads reliably. No major warnings.
```

6.5.3 Browser Console Stress-Check

You can run a small JavaScript snippet in the browser console to **simulate repeated requests** and check for errors.

What to do

1. Open the hosted version of your site
2. Press **F12** to open Developer Tools
3. Go to the **Console** tab
4. Paste a repeat loop like this (example only, adjust table/field names to yours):

```
async function runQuickInserts() {
  for (let i = 1; i <= 20; i++) {
    await supabase
      .from("your_table")
      .insert([{ name: "Test"+i, value: i }]);
  }
  console.log("Insert loop completed");
}
runQuickInserts();
```

5. Press Enter and wait
6. Watch the console for any error messages

What you should screenshot

- The console before running

- The console after running (especially if there are errors)
- A screenshot of the updated data list

How to record it: Add a testing-log entry like:

```
Test 6.5.3 – Browser Console Inserts
Action: Ran insert loop via console
Expected: 20 records added
Actual: 20 records added successfully, no errors
Re-test: Confirmed list updates reflect new entries
```

Important Safety Note: Do this only on your **hosted version** and with **test data**, not real personal data.

Final Reminder for Testing

Testing is **not a formality**. It is one of the clearest ways to show higher-level thinking.

A system with:

- Clear tests
- Honest bugs
- Visible fixes
- Proper re-testing

Will always score higher than a system that "just seems to work".

7. Documentation (Explaining and Evidencing Your Work)

This section focuses on how clearly you have **explained your decisions and recorded your development process**.

The examiner is checking whether:

- Your project can be understood by someone else
- Your decisions make sense
- Your development process is visible and logical

Good documentation supports higher marks across **all sections**, not just this one.

What You Are Being Marked On

- Clarity of explanations
- Justification of technology choices
- Evidence of version control
- Quality of written documentation

Documentation does not need to be long. It needs to be **clear and purposeful**.

Band Expectations (Important)

- **Band 2 (Pass level):** Basic documentation is present. Some explanations exist, but they may be brief or unclear.
- **Band 3 (Merit level):** Documentation explains key decisions and shows clear development progress.
- **Band 4 (Distinction level):** Documentation is clear, concise, and professional. Decisions are justified sensibly, and progress is easy to follow.

7.1 Rationale for Technology Choices

Task: Explain why you chose the technologies used in your project.

You should briefly explain:

- Why HTML, CSS, and JavaScript were suitable
- Why Supabase and SQL were appropriate for data storage
- Why GitHub Pages was used for hosting

This should be **2–4 short sentences**, written in plain English.

You are not being marked on technical depth. You are being marked on **reasonable justification**.

Evidence to Show

- Short written paragraph included in documentation or README.md

Band Signal

- Band 2: Technologies listed
- Band 3: Technologies explained
- Band 4: Technologies justified clearly

7.2 Version Control and Change Log

Task: Show how your project developed over time using GitHub.

You must:

- Commit regularly
- Use meaningful commit messages
- Show progress, not just a final upload

Commit messages should describe what changed and why.

Evidence to Show

- Screenshot of GitHub commit history
- Examples of clear commit messages

Band Signal

- Band 2: Some commits exist

- Band 3: Regular commits show progress
- Band 4: Commit history clearly tells the story of development

7.3 README.md Completion

Task: Ensure your README.md fully explains your project.

It should include:

- What the project does
- How to use it
- Any setup steps needed
- The live GitHub Pages link

The README may be the **first thing the examiner reads**.

Evidence to Show

- Screenshot of completed README.md

Band Signal

- Band 2: README exists
- Band 3: README is clear and helpful
- Band 4: README is complete and professional

Final Reminder for Documentation

Documentation is not about writing lots.
It is about making your work **understandable, traceable, and credible**.

Clear documentation:

- Protects marks
- Makes marking easier

- Reinforces professional standards

Final Checklist for Task 2: Development

You should complete this checklist **before submitting**.
Tick items only when you can provide **clear evidence** (code, screenshots, testing, or commits).

Section 2: Functionality

- ☐ JavaScript connects correctly to Supabase using the public anon key
- ☐ Supabase tables are created correctly with appropriate column names and data types
- ☐ CREATE (Add) functionality works
- ☐ READ (View/display) functionality works
- ☐ UPDATE (Edit) functionality works
- ☐ DELETE functionality works
- ☐ Errors are handled with clear messages
- ☐ The hosted GitHub Pages version works the same as the local version

Band check:

- Pass: Some CRUD works
- Merit: Most CRUD works
- Distinction: Full CRUD works smoothly with no major bugs

Section 3: Code Organisation

- ☐ HTML, CSS, and JavaScript are in separate files
- ☐ Folder structure matches expected VSC layout
- ☐ JavaScript is organised into clear functions
- ☐ Code is readable and consistently named
- ☐ Helpful comments explain complex logic
- ☐ SQL schema is clear and matches JavaScript
- ☐ Coding standards followed (HTML, CSS, JS, SQL)
- ☐ GitHub commits are regular and meaningful
- ☐ README.md exists and explains the project

Band check:

- Pass: Basic separation and some comments
- Merit: Clear structure and steady commits
- Distinction: Professional organisation throughout

Section 4: User Experience

- ☐ Layout is consistent across pages
- ☐ Navigation is clear and predictable
- ☐ Headings and sections are easy to follow
- ☐ Success messages are clear and helpful
- ☐ Error messages explain what went wrong and what to do next
- ☐ All form inputs have labels
- ☐ Text is readable with good contrast
- ☐ Site works on desktop and at least one other screen size
- ☐ Hosted version displays correctly

Band check:

- Pass: Site usable but basic
- Merit: Clear layout and helpful feedback
- Distinction: Fully accessible and polished

Section 5: Security, Legal and Standards

- ☐ Only the public Supabase anon key is used
- ☐ No secret keys appear in GitHub or code
- ☐ User input is validated properly
- ☐ Invalid, blank, and extreme data is handled safely
- ☐ Only necessary data is collected
- ☐ Privacy and GDPR awareness explained briefly
- ☐ Standards followed across HTML, CSS, JS, and SQL

Band check:

- Pass: Some awareness of safety
- Merit: Mostly safe and responsible
- Distinction: Strong, consistent safety throughout

Section 6: Testing

- ☐ Normal data tested
- ☐ Invalid data tested
- ☐ Blank/null data tested
- ☐ Extreme or boundary data tested
- ☐ At least 30 test cases recorded
- ☐ Bugs are logged clearly
- ☐ Fixes are recorded
- ☐ Re-tests are shown after fixes
- ☐ Hosted version is tested and logged
- ☐ Performance and reliability checks

Band check:

- Pass: Mostly normal data tested
- Merit: Wide range of tests with fixes
- Distinction: Full testing including hosted behaviour

Section 7: Documentation

- ☐ Technology choices explained clearly
- ☐ Development process is easy to follow
- ☐ Commit history shows progress over time
- ☐ README.md includes purpose, usage, setup, and live link
- ☐ Screenshots collected for all sections

Band check:

- Pass: Documentation exists
- Merit: Documentation explains decisions
- Distinction: Documentation is clear, concise, and professional

Final Student Reminder

A **simple system that works, is tested properly, and is clearly documented** will always score higher than a complex system that is unfinished or poorly evidenced.

- Do not rush the final stages.
- Check the hosted version.
- Check your testing log.
- Check your commits.
- Then submit with confidence.