**Test and Tag Supplies**

**Guided Digital Sales Assistant**

**Overview: What the Code Does**

v5.html file is a **self-contained interactive product selection tool**.  
It uses **plain HTML, CSS, and JavaScript (no dependencies)** to filter, display, and compare Portable Appliance Tester (PAT) models.

The code builds an **interactive single-page microsite** with three main functions:

**A. Filter Mode**

* Users can apply feature filters (checkboxes, dropdowns, text search) to narrow down available PAT testers.
* Results update **instantly** as filters change.
* A toggle lets users switch between **card view** and **table view**.

**B. Wizard Mode**

* A simplified “quick find” version that asks just a few questions (e.g., *Need Printer?*, *Need Recording?*, *Max Weight?*).
* Generates matching products and explains *why each model was suggested.*

**C. Compare Mode**

* Users can add products to a shortlist for side-by-side comparison.
* Displays all key specifications and direct “View Product” purchase links.

**D. Supporting Features**

* **CSV Export:** Download filtered results in spreadsheet form.
* **Toast Notifications:** Simple feedback messages (e.g., “Added to compare”).
* **Responsive Layout:** Adjusts between desktop, tablet, and mobile.

**How the Code Works (Step-by-Step)**

**1. Data Setup**

* Product data is stored in a JavaScript array named RAW.
* Each object in the array represents a product (e.g., "WAVTITAN").
* A matching BUY\_URLS object maps model IDs to live product pages.

const RAW = [

{id:'WAVTITAN', name:'WAVTITAN', PortableRCD:'YES', FixedRCD:'YES', ... },

...

];

**2. Data Normalization**

Each product entry is processed into a more structured format (PRODUCTS), adding:

* Boolean flags (p.b) for easy filtering.
* Numeric and text metrics (p.metrics) for comparisons.

Example:

const PRODUCTS = RAW.map(r=>({

...r,

b: { Printer: true, Comms: false, ... },

metrics: { outlet: '15A', weight: 1.8, battery: 'Li-Ion' }

}));

**3. User Filtering**

When a user types, checks boxes, or selects dropdowns:

* getFilterReq() collects the current filter settings.
* passes(p, req) checks whether each product meets the criteria.
* Matching results are sent to:
  + renderCards() for grid display, or
  + renderTable() for table view.

**4. Wizard Functionality**

runWizard() reads minimal input (3–4 fields) and filters based on core features only — designed for quick decision-making.

**5. Compare Feature**

Products added to a shortlist (Set) are rendered in a comparison table.  
Users can remove items at any time.

**6. CSV Export**

exportCsv() generates a .csv file client-side using Blob, allowing users to download the current filtered list.

**7. Event Handling**

All interactivity (tab switching, filtering, adding/removing) is handled via event listeners attached to DOM elements when the script loads.

**Main Purpose**

The main purpose of this code is to provide a **lightweight, interactive decision-making tool** that helps customers:

* Narrow down technical products based on features.
* Quickly compare similar PAT testers.
* Transition directly to purchase pages on your main store.

Essentially, it’s a **guided digital sales assistant** for your product range.

**Recommended Improvements**

| **Area** | **Recommendation** | **Benefit** |
| --- | --- | --- |
| **Data Source** | Move RAW and BUY\_URLS into a separate JSON file (e.g., /data/pat-products.json) and load via fetch() | Easier updates without editing HTML |
| **Modularity** | Split CSS and JS into /assets/styles.css and /assets/script.js | Cleaner maintenance, easier integration into CMS |
| **Accessibility** | Add ARIA labels for form controls and improve keyboard navigation | Meets WCAG compliance |
| **Analytics** | Add GA4 or GTM event tracking (filter use, compare actions, product link clicks) | Measure engagement & conversion impact |
| **Performance** | Lazy-load images if product thumbnails are added later | Faster initial load |
| **Scalability** | Refactor filtering logic to allow multiple product categories or brands | Easier to extend beyond PATs |
| **UX** | Add “Clear Compare” and “Back to Filters” buttons | Improves usability for mobile users |
| **Styling** | Use a consistent responsive grid gap (12–16px) and unified shadow intensity | Slight polish for consistency with brand aesthetic |

**Updating the Product Matrix (Easiest Method)**

There are **three easy options** for keeping the matrix current.

**Option 1 — Direct Edit (Quickest)**

Open the HTML file and modify the RAW array directly:

{id:'NEW123', name:'NEW123', PortableRCD:'YES', FixedRCD:'YES', Outlet:'15A', ...}

Add or remove items as needed.  
⚠️ Be sure to also add the corresponding entry to BUY\_URLS.

**Option 2 — External JSON File (Recommended)**

Move the product matrix to /data/products.json:

[

{

"id": "WAVTITAN",

"name": "WAVTITAN",

"PortableRCD": "YES",

"FixedRCD": "YES",

"Printer": "YES",

"Battery": "Li-Ion"

}

]

Then update the script:

fetch('/data/products.json')

.then(res => res.json())

.then(raw => {

const PRODUCTS = processRaw(raw);

filterProducts();

});

This allows non-developers to update the data without touching code.

**Option 3 — Live Data (Future Enhancement)**

Connect to an online data source (e.g., your Shopify API or Google Sheets).  
This makes the selector auto-sync whenever product specs or stock change.

**Summary**

| **Aspect** | **Summary** |
| --- | --- |
| **Code Purpose** | A self-contained interactive tool for finding and comparing PAT testers. |
| **Core Logic** | Client-side filtering of a static dataset, with UI modes for filtering, wizard selection, and comparison. |
| **Ease of Maintenance** | Very high — all logic is local and no dependencies. |
| **Recommended Changes** | Externalize data, modularize code, improve accessibility & analytics. |
| **Matrix Updates** | Edit inline array (short term) or load external JSON (long term). |

**Microsite Project Scope Document (v2)**

**1. Project Overview**

The **PAT Product Selector Microsite** enables customers to quickly identify, filter, and compare Portable Appliance Testers (PATs) based on technical specifications and user needs.  
It’s designed to enhance product discoverability, reduce selection complexity, and increase conversion rates.

**2. Objectives**

* Provide an intuitive product comparison and selection tool.
* Enhance customer confidence and engagement.
* Integrate directly with eCommerce product pages.
* Support exportable data (CSV) for internal and professional use.

**3. Scope of Work**

**Core Deliverables:**

| **Component** | **Description** |
| --- | --- |
| **Interactive Filter Tool** | Sidebar filter with live product results and dual card/table view. |
| **Quick Wizard Mode** | Step-by-step simplified version for rapid recommendations. |
| **Comparison Table** | Dynamic side-by-side comparison of shortlisted items. |
| **Export to CSV** | Downloadable dataset of current filtered results. |
| **Toast Notifications** | Subtle user feedback for actions. |
| **Responsive Design** | Full compatibility for desktop, tablet, and mobile. |
| **Accessibility** | ARIA roles and keyboard-friendly navigation. |

**4. Deliverables**

* Finalized v5.html file (minified and optimized).
* Optionally, separated styles.css and script.js for modular integration.
* Deployment guide for:
  + Shopify (or existing CMS) integration.
  + Static microsite deployment.
* Basic analytics event tracking setup.

**5. Technical Specifications**

| **Area** | **Detail** |
| --- | --- |
| **Frontend Stack** | HTML5, CSS3, Vanilla JavaScript (no external libraries). |
| **Hosting** | Static hosting (Shopify, Netlify, Vercel, or AWS S3). |
| **Responsive Breakpoints** | 700px, 1100px for grid transitions. |
| **Accessibility** | Screen reader roles, keyboard operability. |
| **Performance** | Lightweight (<200 KB total). |
| **Data** | Static JSON-like array embedded in JS, easily editable. |

**6. Integration Paths**

| **Option** | **Integration Summary** | **Recommended For** |
| --- | --- | --- |
| **A. Embedded in Main Site** | Integrate into /pages/product-selector within testandtagsupplies.com.au | Conversion optimization, unified analytics |
| **B. Standalone Microsite** | Deploy on subdomain (e.g., selector.testandtagsupplies.com.au) | Marketing, ads, campaigns, product demos |

**7. Timeline (Estimated)**

| **Phase** | **Deliverable** | **Duration** |
| --- | --- | --- |
| Setup & Integration | Choose hosting path and deploy base HTML | 1–2 days |
| QA & Branding | Match visual style to site, test responsiveness | 2 days |
| Analytics Setup | Add tracking and verify conversions | 1 day |
| Launch | Public go-live | 1 day |

**8. Success Metrics**

* ≥ 90% of users complete filter or wizard flow.
* ≥ 10% CTR from selector to product pages.
* Load time < 1.5s (desktop), < 2s (mobile).
* 100% functional “Export CSV” and “Compare” features.

**9. Future Enhancements**

* JSON-based product feed (live data sync).
* PDF download of comparison report.
* Multi-brand or multi-category expansion.
* Dark mode / accessibility enhancements.
* Analytics event tagging for product-level insights.

**Technical Integration Options**

**Option A — Integrate into Existing eCommerce Site (Recommended)**

**1. Use Case**

Embed the Product Selector as a functional page (e.g., /pages/product-selector) on **testandtagsupplies.com.au**.  
Ideal for seamless shopping flow and shared analytics.

**2. Technical Approach**

| **Step** | **Description** |
| --- | --- |
| **1. Create a new content page** | In your CMS (likely Shopify → Online Store → Pages), create a new page titled “Product Selector”. |
| **2. Add a custom HTML block** | Edit the page’s theme template to include a raw HTML block or a <div> container. |
| **3. Inject microsite HTML** | Copy the body of your v5.html into that block, excluding <html>, <head>, and <body> tags. |
| **4. Include assets** | Move the <style> and <script> contents into your theme: |
|  | • Place <style> in your theme’s custom.css or <style> block in the page. |
|  | • Place <script> content in the theme’s footer, or wrap it in <script> tags in the same page. |
| **5. Adjust styling** | Add a small wrapper class (e.g., .pat-selector) to prevent CSS conflicts with your theme. |
| **6. Test navigation links** | Ensure all “View Product” buttons (BUY\_URLS) open the correct product pages in new tabs. |
| **7. Add tracking** | Use Google Tag Manager or GA4 to log filter use, wizard completions, and outbound product clicks. |

**3. Hosting & Maintenance**

* Hosted entirely within your existing domain and CDN.
* Versioning handled in your theme’s code or through CMS snippets.
* Minimal maintenance — just replace dataset array as products change.

**4. Pros**

* Seamless brand and checkout continuity.
* No duplicate analytics setup.
* Retains main site SEO benefits.
* Easier for customers (no domain hop).

**5. Cons**

* May require developer assistance to fit within CMS constraints.
* Must test CSS isolation to prevent style collisions.

**Option B — Standalone Microsite Deployment**

**1. Use Case**

Deploy the tool as an independent landing page, ideal for campaigns or advertising funnels.  
Example: https://selector.testandtagsupplies.com.au

**2. Technical Approach**

| **Step** | **Description** |
| --- | --- |
| **1. Hosting platform** | Deploy static v5.html file on a host like **Netlify**, **Vercel**, **GitHub Pages**, or **AWS S3 + CloudFront**. |
| **2. Domain setup** | Create a subdomain (selector.testandtagsupplies.com.au) via your DNS provider and point it to your host. |
| **3. SSL & HTTPS** | Ensure SSL certificate (automatic via Netlify/Vercel). |
| **4. Analytics** | Add your existing Google Tag Manager or GA4 snippet into the <head> of the HTML. |
| **5. Performance tuning** | Compress the HTML and enable caching headers (most static hosts do this automatically). |
| **6. Maintenance** | Replace dataset or update design by re-uploading an updated HTML file. |

**3. Pros**

* Fully independent, ultra-fast, easy to update.
* No dependency on CMS restrictions.
* Great for marketing campaigns, QR codes, or social ads.
* Clean testing sandbox for UI/UX experiments.

**4. Cons**

* Separate domain = separate analytics (requires cross-domain tracking).
* Users leave main store to select a product.
* Limited SEO benefit to your main commerce site.

**Does it need to be modular for eCommerce site integration?**

**Short answer:**  
*Not strictly required*, but **strongly recommended** for long-term maintainability and theme compatibility.

Let’s break it down in context of integrating this into **testandtagsupplies.com.au** (very likely Shopify or similar CMS).

**Option 1 — Keep It Self-Contained (No Modular Split)**

You can **embed the current HTML file as-is** inside your store theme or custom page.  
This works because:

* It’s a **standalone file** with inline <style> and <script> blocks.
* There are **no external dependencies** or imports.
* Everything runs in the browser client-side (no backend).

**Advantages**

* **Fastest setup:** copy–paste into a new page template or a “custom HTML” block.
* **Zero dependencies:** no need to manage multiple files or paths.
* **Predictable behavior:** CSS and JS load together in the same scope.

**Considerations**

* Harder to update later (you must edit HTML directly for every data or style change).
* Inline CSS and JS can **clash with your site’s global styles**.
* Editing large inline blocks in Shopify’s code editor is error-prone.

This method is fine if:

* The selector will change infrequently.
* Only a few people will edit or maintain it.
* You’re embedding it once and leaving it alone.

**Option 2 — Modular (Recommended for eCommerce Integration)**

Break the file into **3 small, maintainable parts**:

/product-selector/

├── index.html

├── selector.css

└── selector.js

**Why modularize:**

Your eCommerce platform (like Shopify) already loads:

* Global CSS (theme.css)
* Site-wide JS (for navigation, cart, etc.)

Keeping your selector isolated reduces risk of conflicts and allows easy future updates.

**Advantages of Modular Integration**

| **Area** | **Benefit** |
| --- | --- |
| **Styling Control** | Separate CSS can use a wrapper class (.pat-selector) to isolate styles from your main theme. |
| **Easy Updates** | You can replace or edit selector.js or data.json without touching the theme template. |
| **Cleaner CMS Page** | The page template remains tidy — you just reference the external files. |
| **Reusability** | The selector can be reused on other product categories in the future. |
| **Future-proofing** | Easier to add analytics or data loading without rewriting inline code. |

**Technical Example (Shopify Integration)**

In your Shopify page template (e.g., page.product-selector.liquid):

<div class="pat-selector">

<div id="product-selector-root"></div>

</div>

<!-- Include external assets -->

<link rel="stylesheet" href="{{ 'selector.css' | asset\_url }}" />

<script src="{{ 'selector.js' | asset\_url }}" defer></script>

Then your selector.js file would handle:

* Rendering all HTML (or injecting your current markup),
* Running the filtering and comparison logic,
* Fetching the product data (products.json).

Your dataset can then live in:

/assets/products.json

and be updated via the Shopify admin (or uploaded via theme files).

**Recommended Approach for You**

| **If you want...** | **Do this** |
| --- | --- |
| **Quick embed, minimal changes** | Keep current single-file HTML and paste into a new Shopify Page (good for proof of concept). |
| **Smooth long-term integration, easy updates, no conflicts** | Modularize: split CSS and JS, load via theme assets, and externalize data. |

**Pro Tip**

You can adopt a **hybrid approach**:

* Keep your **HTML and CSS inline** for simplicity,
* Move **only the JS data matrix** (RAW + BUY\_URLS) into an external file like /assets/products.json.

That way:

* You still have a one-file embed for visuals,
* But updating products becomes trivial — upload a new JSON file, done.

**Summary**

| **Factor** | **Inline (Current)** | **Modular (Recommended)** |
| --- | --- | --- |
| Setup time | 🟢 Quick | 🟡 Slightly longer |
| Maintenance | 🔴 Harder | 🟢 Easy |
| Theme safety | 🔴 Risk of CSS conflict | 🟢 Isolated |
| Future scalability | 🔴 Limited | 🟢 Excellent |
| Ideal for | Demo / prototype | Production integration |