# Phase 5 Build Plan – Complete Implementation Guide

This document outlines how to implement all remaining features for **Phase 5** of the Google Dorking Station, building upon the Phase 5.0 “Community Packs” baseline. The goal is to provide clear guidance on where to add new state, helper functions, and UI elements. Code snippets are designed to be dropped into the existing React component in GoogleDorkingStationPhase50.

## 1. Expand pack schema and import/export

Phase 5.0 already supports community packs stored in localStorage. To fully support user‑generated packs, you should extend the pack schema to include metadata (author, description, tags, version, created/updated timestamps, and optional placeholders), and validate that imported packs conform to this structure.

// At the top of the file near existing constants:  
const COMMUNITY\_PACKS\_KEY = "gds\_community\_packs\_v1";  
  
// A helper for validating an imported pack (drop this near other helpers):  
function validatePack(p) {  
 if (!p || typeof p !== 'object') return false;  
 if (!p.name || typeof p.name !== 'string') return false;  
 const cats = ['Person','Place','Thing'];  
 for (var i=0;i<cats.length;i++) {  
 var arr = p[cats[i]];  
 if (arr && !Array.isArray(arr)) return false;  
 if (arr) {  
 for (var j=0;j<arr.length;j++) {  
 var t = arr[j];  
 if (!t || typeof t.template !== 'string' || typeof t.label !== 'string') return false;  
 }  
 }  
 }  
 return true;  
}  
  
// When importing a community packs JSON file, call validatePack() on each element and skip invalid ones.

### Creating a pack from favorites

You can let users bundle a new pack directly from their favorites. Below is a helper function that converts the favorites array into a pack object; insert it near the state hooks in your component.

function favoritesToPack(packName, author) {  
 var byCat = { Person: [], Place: [], Thing: [] };  
 for (var i=0; i < favorites.length; i++) {  
 var f = favorites[i];  
 if (byCat[f.category]) {  
 byCat[f.category].push({  
 label: f.label,  
 template: f.template,  
 hint: 'Saved favorite'  
 });  
 }  
 }  
 return {  
 name: packName.trim() || 'Untitled',  
 author: author || 'Local',  
 description: 'Generated from favorites',  
 version: 1,  
 createdAt: Date.now(),  
 updatedAt: Date.now(),  
 Person: byCat.Person,  
 Place: byCat.Place,  
 Thing: byCat.Thing  
 };  
}

Create a simple UI in the Community tab to name a pack and export it using this helper.

## 2. Collections (shared favorites)

Collections let users organise their saved queries into named sets that can be exported/imported. Define a new localStorage key and state to hold collections:

const COLLECTIONS\_KEY = "gds\_collections\_v1";  
const [collections, setCollections] = useLocalJSON(COLLECTIONS\_KEY, []);

Each collection is an object { id, name, description?, items: [], createdAt, updatedAt }. Implement helper functions to create a collection and add a favorite into a collection:

function createCollection(name, description) {  
 const c = {  
 id: 'COLL\_' + Math.random().toString(36).slice(2, 8),  
 name,  
 description: description || '',  
 items: [],  
 createdAt: Date.now(),  
 updatedAt: Date.now(),  
 };  
 setCollections(prev => prev.concat([c]));  
}  
  
function addFavoriteToCollection(collId, fav) {  
 setCollections(prev => prev.map(coll => {  
 if (coll.id !== collId) return coll;  
 // Avoid duplicates by label + template  
 const exists = coll.items.some(it => it.label === fav.label && it.template === fav.template);  
 if (exists) return coll;  
 return {  
 ...coll,  
 items: coll.items.concat([{  
 label: fav.label,  
 template: fav.template,  
 category: fav.category  
 }]),  
 updatedAt: Date.now()  
 };  
 }));  
}

Expose a **Collections** tab (next to **Community**) that lists existing collections and allows new ones to be created, exported, or imported from a JSON file. In the **Favorites** tab, add an “Add to Collection” dropdown to each row.

## 3. Substitutions and nested builder

### Substitutions

Implement support for placeholders like {date}, {location}, and {filetype} by adding a new state and helper function:

const [subs, setSubs] = useState({ q: '', date: '', location: '', filetype: '' });  
  
function applySubs(str, map) {  
 var out = String(str || '');  
 for (var k in map) {  
 if (Object.prototype.hasOwnProperty.call(map, k)) {  
 out = out.split('{' + k + '}').join(map[k] || '');  
 }  
 }  
 return out;  
}

Modify the computed renderedQuery to apply substitutions and fall back to {q} if a placeholder is missing:

const renderedQuery = useMemo(() => {  
 var t = builtTemplate;  
 t = applySubs(t, subs);  
 // Always replace {q} at the end  
 t = t.split('{q}').join(q.trim());  
 return t;  
}, [builtTemplate, subs, q]);

In the UI, add a small form beneath the builder to edit the substitution values. Use a simple grid of inputs; label each input with its placeholder name:

{/\* Substitution form \*/}  
<div className="mt-4 grid md:grid-cols-4 gap-2">  
 {['q','date','location','filetype'].map(k => (  
 <div key={k}>  
 <label className="block text-xs text-gray-600 mb-1">{'{' + k + '}'}</label>  
 <input value={subs[k]} onChange={e => setSubs(prev => ({ ...prev, [k]: e.target.value }))}   
 className="w-full rounded-xl border px-2 py-1 text-sm" placeholder={k === 'q' ? 'keyword' : k} />  
 </div>  
 ))}  
</div>

### Nested builder (optional)

The baseline uses a linear array of chips. For advanced grouping, define a tree structure with GROUP nodes (AND/OR) and allow chips to be dragged into these groups. Because this requires substantial UI work, you can keep the existing linear builder as “Simple mode” and implement a second “Group mode” as a separate component. See the plan for details on the node schema.

## 4. Learning Mode 2.0

Extend Learning Mode with quiz questions and a sandbox view. Define a small quiz bank:

const QUIZ\_BANK = [  
 { id: 'q1', prompt: 'Which operator excludes Pinterest results?', options: ['site:', '-', 'filetype:', 'intitle:'], correctIndex: 1, explain: 'Use a leading minus ( -example.com ) to exclude hosts or terms.' },  
 { id: 'q2', prompt: 'How do you restrict to PDF files?', options: ['inurl:', 'filetype:', 'intitle:', 'site:'], correctIndex: 1, explain: 'Use filetype:pdf to limit results to PDFs.' },  
];

In the Learning Mode drawer, add a tabbed section called “Practice” that cycles through these questions. Display radio buttons and show feedback when the user checks an answer. Consider awarding badges after a user answers three questions correctly.

For the sandbox view, you can either embed the actual engine results in an <iframe> or simulate them by generating dummy results. A simulated approach keeps the app offline and avoids terms‑of‑service issues.

## 5. Suggestion engine

Create a rule‑based helper that examines the keyword (q) and the current template to propose extra operators. This helper returns an array of suggestion objects with a type (add, wrap, exclude), a chip definition, and a reason:

function suggestFor(q, built) {  
 var S = [];  
 var ql = String(q || '').toLowerCase();  
 // Device‑like tokens suggest PDF manuals  
 if (/\b(model|router|xr|mk|pro|plus|series|v\d)\b/.test(ql) && !/filetype:pdf\b/i.test(built)) {  
 S.push({ type: 'add', chip: { kind: 'FILETYPE', value: 'pdf' }, reason: 'Looks like a device; try manuals/specs.' });  
 }  
 // Place names suggest .gov  
 if (/\b(city|county|dept|department|university|school|district)\b/.test(ql) && !/site:\.gov\b/i.test(built)) {  
 S.push({ type: 'add', chip: { kind: 'SITE', value: '.gov' }, reason: 'Government domains often have official docs.' });  
 }  
 // Multi‑word names benefit from quotes  
 if (!/"/.test(built) && /\s/.test(q)) {  
 S.push({ type: 'wrap', reason: 'Use quotes for exact full‑name matching.' });  
 }  
 // Remove Pinterest for design/inspiration keywords  
 if (!/-pinterest\.com/.test(built) && /photo|design|inspiration/i.test(ql)) {  
 S.push({ type: 'exclude', value: 'pinterest.com', reason: 'Remove Pinterest results.' });  
 }  
 return S.slice(0, 4);  
}

In the UI, show a suggestions bar above the builder. Each suggestion renders as a pill; clicking a pill inserts or modifies the appropriate chip in the builder:

{/\* Suggestion bar \*/}  
<div className="mt-2">  
 {suggestFor(q, renderedQuery).map((s, i) => {  
 let label = '';  
 if (s.type === 'add') label = '+' + (s.chip.kind === 'FILETYPE' ? 'filetype:' : 'site:') + s.chip.value;  
 if (s.type === 'wrap') label = 'Wrap in "…"';  
 if (s.type === 'exclude') label = '-' + s.value;  
 return (  
 <button key={i}  
 onClick={() => {  
 if (s.type === 'add') {  
 setChips(prev => prev.concat([{ id: 'c' + Math.random().toString(36).slice(2, 6), type: s.chip.kind, value: s.chip.value }]));  
 }  
 if (s.type === 'wrap') {  
 setChips(prev => prev.map(c => c.type === 'PHRASE' ? { ...c, value: c.value || q } : c));  
 }  
 if (s.type === 'exclude') {  
 setChips(prev => prev.concat([{ id: 'c' + Math.random().toString(36).slice(2, 6), type: 'EXCLUDE', value: s.value }]));  
 }  
 }}  
 title={s.reason}  
 className="text-xs mr-2 mb-2 px-2 py-1 rounded-lg border bg-white hover:bg-gray-100">  
 {label}  
 </button>  
 );  
 })}  
</div>

## 6. Safety & guardrails

To discourage misuse, detect potentially sensitive search patterns and warn the user. Write a helper that scans a template and returns a list of triggers:

function detectSensitive(tpl) {  
 const t = String(tpl || '').toLowerCase();  
 const hits = [];  
 if (/\binurl:\s\*(admin|login|signup|register|dashboard)\b/.test(t)) hits.push('admin/login paths');  
 if (/\b(password|passwd|api\_key|token)\b/.test(t)) hits.push('credential terms');  
 if (/\bindex\.of\b/.test(t)) hits.push('directory listing');  
 return hits;  
}

Render a banner near the builder when builtTemplate contains any sensitive operators:

{(() => {  
 const hits = detectSensitive(builtTemplate);  
 if (!hits.length) return null;  
 return (  
 <div className="mt-3 rounded-xl border border-red-200 bg-red-50 text-red-800 text-xs p-3">  
 <div className="font-semibold mb-1">Heads‑up: sensitive operators detected</div>  
 <div>Flags: {hits.join(', ')}. Use these queries only for legitimate research and never to access systems without permission.</div>  
 </div>  
 );  
})()}

## 7. UX enhancements

### Dark mode toggle

Persist a theme setting in localStorage and apply it by toggling a dark class on document.documentElement. Define a constant and state for the theme:

const THEME\_KEY = 'gds\_theme';  
const [theme, setTheme] = useState(() => {  
 try { return localStorage.getItem(THEME\_KEY) || 'system'; } catch (e) { return 'system'; }  
});  
  
useEffect(() => {  
 try { localStorage.setItem(THEME\_KEY, theme); } catch (e) {}  
 const el = document.documentElement;  
 if (theme === 'dark' || (theme === 'system' && window.matchMedia('(prefers-color-scheme: dark)').matches)) {  
 el.classList.add('dark');  
 } else {  
 el.classList.remove('dark');  
 }  
}, [theme]);

Place a small button in the header to switch between light and dark modes:

<button onClick={() => setTheme(prev => prev === 'dark' ? 'light' : 'dark')}  
 className="ml-2 text-xs rounded px-2 py-1 border bg-white hover:bg-gray-100">  
 {theme === 'dark' ? 'Light Mode' : 'Dark Mode'}  
</button>

### Command palette and split layout

While not required immediately, you can implement a command palette (triggered by Ctrl/Cmd+P) to quickly run actions (new collection, toggle dark mode, open all queries, etc.) and a toggle for a split layout where favorites and the builder are side‑by‑side. Persist the chosen layout in localStorage under a new key (e.g. gds\_layout).

## Summary

Implementing these features completes the Phase 5 build by adding shared collections, richer substitution support, operator suggestions, safety warnings, and a theme toggle. Use the code snippets above to modify your existing GoogleDorkingStationPhase50 component. For a more detailed step‑by‑step patch, refer back to the comprehensive plan delivered earlier. Once these hooks are in place, you can further refine Learning Mode and the nested builder in future updates.