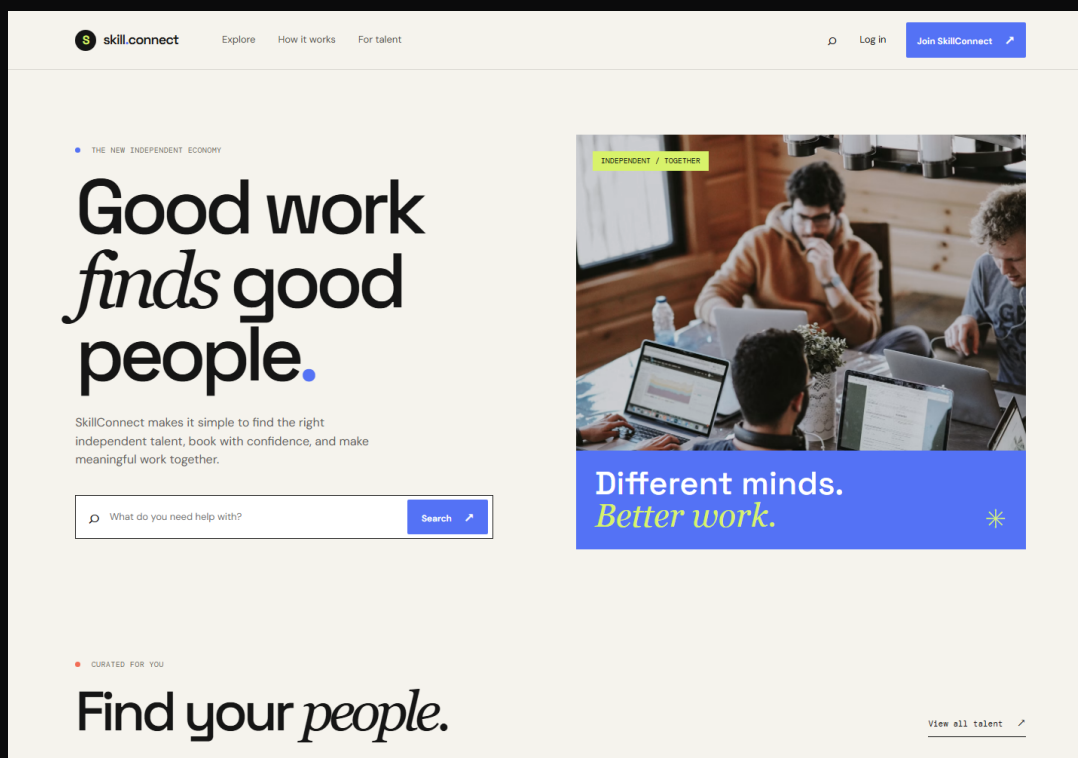


PROJECT DOCUMENTATION

SKILLCONNECT CAPSTONE PROJECT

Syed Roshan Shareq
Project Owner and Developer

Architecture, discovery workflows, local accounts, service management, data design, deployment and maintenance.



SkillConnect connects a fictional talent catalog with searchable discovery, a saved shortlist, booking requests, and a local service-management workspace.

Project source snapshot. These notes describe the implementation in this folder, including its current constraints and the boundary between the working demo and a future server-backed marketplace.

Contents

Project Boundary	3
Files and Dependencies	3
Startup and Application State	4
Catalog Construction	4
Discovery and Shortlist	5
Local Accounts and Session	6
Booking Requests and My Studio	7
Service Management	8
Browser Storage Contracts	9
Reference PostgreSQL Model	10
Presentation and Accessibility	11
Running and Publishing	11
Maintenance and Verification	12
Source Navigation	13

Project Boundary

SkillConnect is a static freelancer discovery and booking capstone. HTML supplies the page and dialogs, CSS supplies the responsive visual system, and vanilla JavaScript owns catalog rendering and browser-local workflows. Vercel serves three public assets; there is no running application server or API.

The 104 baseline talent profiles are fictional. Registration, login, shortlist, bookings, and listing CRUD use this origin's localStorage. Accounts and records are not shared between devices, browsers, or origins. They are also not isolated between accounts within the same browser.

The PostgreSQL schema is a design reference only. Its roles, payments, reviews, booking status history, and relational guarantees do not power the current interface. No payment is collected and a saved booking does not notify a freelancer. Use disposable demo credentials, never a real password.

Files and Dependencies

Location	Responsibility
index.html	Main page, metadata, navigation, booking/auth/studio dialogs
styles.css	Visual system, grids, dialog layout, responsive overrides
app.js	Catalog, browser storage, rendering and all demo actions
build.mjs / dev.mjs	Static asset copy and dependency-free Node preview
database/schema.sql	Unconnected PostgreSQL reference model
package.json / package-lock.json	ES-module tooling and npm command definitions
vercel.json	Build output, catch-all rewrite and security headers
README.md / DEPLOYMENT.md	Scope, setup, submission and hosting notes

Runtime and external assets

There are no installed runtime packages or frontend framework. The browser uses DOM APIs, FormData, Web Crypto, and localStorage. Node.js 18+ is the documented tooling prerequisite. Google Fonts delivers DM Sans, DM Mono, and Space Grotesk; Unsplash supplies remote photographs. These are active third-party asset dependencies, even though there are no third-party business-service integrations.

Startup and Application State

The classic app.js script loads at the end of the document, after its DOM targets exist. Its top-level execution builds the catalog, reads persistent arrays and the saved session, binds event listeners, updates the account controls, and renders the first four matching cards.

```
index.html -> app.js
  catalog + localStorage -> in-memory state
  bind controls -> updateAccountUI() -> renderTalent()
  user action -> mutate state -> persist -> redraw
```

Initialization sequence

1. Combine the four seed profiles with 100 generated profiles.
2. Read saved services and restore only entries marked owner.
3. Reconcile saved listings with baseline profiles by name.
4. Make a shuffled All-talent array and read bookings/users/session.
5. Remove a stored session if its email is absent from the users array.
6. Set category All, authentication mode login, and visible limit four.
7. Register controls and perform the initial account/catalog render.

State	Persistence / behavior
talent / allTalentOrder	Working catalog and separate shuffled All-talent ordering
bookings / users / currentUser	Loaded from localStorage; mutations write back explicitly
activeCategory / visibleLimit	Transient filter and result limit; reset on reload
search / sort / dashboard tab	DOM-backed transient controls; not stored as preferences
Dialog / navigation state	CSS classes plus aria-hidden or aria-expanded attributes

Catalog Construction

Four handwritten profiles plus 100 generated records produce Design 27, Development 26, Writing 26, and Marketing 25. Names combine ten first names with ten surnames; categories cycle across four specialties and locations across eight cities. Prices use $700 + ((\text{index} * 137) \% 2800)$; ratings cycle from 4.6 to 5.0.

Category image URLs are reused across profiles. Ratings, review counts, names, and cities are generated display content, not verified marketplace facts. The All-talent order uses `sort(() => Math.random() - 0.5)`, an unstable, nonuniform shuffle rather than an editorial ranking. Reloads can show different featured people.

Discovery and Shortlist

How a catalog request reaches the cards

renderTalent() reads the trimmed, lowercased search input. It uses the shuffled list for All and the working catalog for a selected category, then intersects the exact category match with a substring search across name, role, category, and location. Service description is not searched separately.

The selected sort is applied after filtering. Top rated converts rating strings to numbers; low-to-high price strips nondigits before numeric comparison. Four records are shown initially. View all sets the limit to Number.MAX_SAFE_INTEGER, revealing every match; there is no paginated API or incremental network loading.

Control	Implemented outcome
Search input	Re-render on every input event; reset visibility to four
Category button	Select category, reset limit, render and scroll to catalog
Sort selector	Apply rating/price sorting and reset limit to four
View all button	Reveal every match; disabled when there are no more results
Filters button	Show an informational toast; no additional filter dialog
No matching talent	Render an empty state with a revised-search suggestion

Saved shortlist

A card's heart button adds or removes the profile name in skillconnect-saved. Saving requires a current local user; otherwise login opens. The original save action is not resumed after authentication. The dashboard resolves each saved name against the current catalog, discards missing matches from the rendered list, and offers booking or removal.

Shortlists are shared browser data, not account-owned records. A listing rename or deletion can leave stale names in storage. Duplicate names are ambiguous because lookup uses the first matching record. Stable service/profile IDs would be needed for reliable references.

Rendering and scale

The full card collection is replaced through innerHTML, then card listeners are rebound. This is simple at the current catalog size, but View all creates every matching image/card at once. There is no virtualization, image-loading policy for catalog cards, or measured performance budget. Rendering user-controlled titles without escaping is a stored-injection risk.

Local Accounts and Session

`openAuth()` switches one dialog between signup and login. It changes the title, submit text, name-field requirement/visibility, and the alternate-mode action. Native form validation enforces email input and a minimum password length of six characters. Names are trimmed in the submit handler.

Registration

1. Normalize email by trimming and lowercasing.
2. Hash the supplied password with `crypto.subtle.digest("SHA-256", ...)`.
3. Reject an email already present in the browser's user array.
4. Create an account with `crypto.randomUUID()` and role `CLIENT`.
5. Persist the users array and a session without its password hash.
6. Swap desktop/mobile account buttons, close the dialog, and show feedback.

Login and logout

Login locates the normalized email and compares its stored hash with the newly computed hash. Unknown emails and mismatches show the same error message. Success persists id, name, email, and role as `skillconnect-user`. Logout removes that session key and closes the dashboard; it does not erase accounts, listings, shortlist, or bookings.

Record	Fields stored
Account	id, name, email, passwordHash, role
Session	id, name, email, role
Session restoration	Email membership check only; no token verification or expiry
Available account roles	The client always registers <code>CLIENT</code> ; no role-selection workflow

Security boundary

This is an authentication simulation. Fast, unsalted SHA-256 is not a production password-storage scheme. Browser JavaScript can read the hashes, and a visitor can alter `localStorage` to impersonate another local account. The saved session is not a trusted credential. No server verifies identity, permissions, password recovery, rate limits, or session revocation.

Web Crypto and UUID generation require a supported browser and an appropriate secure context; `localhost` and `HTTPS` are the intended environments. A future server should use an adaptive password hash or managed identity, server-validated sessions, authorization on every mutation, and account-scoped storage.

Booking Requests and My Studio

Book now opens the booking dialog using the selected record's service text. The current user's email is prefilled when available, but a visitor does not need to log in to request a booking. The form asks for email, project requirements, preferred start date, and budget.

What actually gets saved

The submit listener prevents navigation, reads FormData, prepends a record to the in-memory bookings array, writes skillconnect-bookings, closes the dialog, resets the form, and shows a confirmation toast. Only the three fields below survive.

Persisted field	Source / meaning
service	Displayed service text from #modal-service
email	Email input from submitted FormData
created	Browser locale date from new Date().toLocaleDateString()

Requirements, start date, and budget are collected but discarded. There is no booking ID, service foreign key, client ID, structured timestamp, or persisted booking status. The dashboard displays REQUESTED for every row; this is a fixed presentation label, not a workflow transition.

Workspace panels

Panel	Behavior
Bookings	All browser-local requests, newest first, with service/email/date
Shortlist	Resolvable saved names with booking and removal actions
My services	Create/edit form and all entries marked owner, with edit/delete

Opening My studio runs renderDashboard(), updates its three counts, and rebuilds each panel. Tab buttons switch active styling and hidden panels. The Create your profile CTA opens signup for a visitor; for a signed-in user it opens My services. It is not a freelancer-profile editor and does not resume that destination after signup.

Current boundaries

All local accounts see the same bookings and owned listings. Booking from the shortlist can leave the dashboard behind the new dialog; submission does not refresh the already-open booking panel until the dashboard is rendered again. No email delivery, scheduling, acceptance, cancellation, completion, invoice, or payment occurs.

Service Management

The My services form accepts a required title, one of four categories, and a starting price with HTML minimum one. A hidden service ID selects create versus update. New records are inserted into the same talent array as fictional professionals, so a service title occupies the profile-name slot.

Create, edit, delete

Create assigns a UUID, owner:true, the submitted title/category/price, a category image, a placeholder location, a generic role, rating New, zero reviews, and service text equal to the title. The client then reshuffles All talent, stores the complete catalog, resets the form, and refreshes the catalog and dashboard.

Edit locates a record by ID and fills the form; Save changes modifies name, category, price, and image. Cancel edit clears the ID and restores the create form. Delete asks for browser confirmation, removes the record from both arrays, persists the remaining catalog, and redraws the interface.

Implementation detail	Maintenance consequence
owner is a boolean	No account association; every local account sees the listing
name is used for restore and saves	Renames, duplicates and seeded-name collisions break identity
service text is not updated on edit	Booking a renamed listing can use its original title
price is a formatted string	Numeric sorting/parsing strips nondigits, losing decimal meaning
rating is New for added listings	Number("New") is NaN, so rating sort lacks a defined position
category totals are hardcoded	Displayed category counts do not follow listing mutations

Stored HTML injection

User-controlled titles are interpolated into innerHTML and data/alt/label attributes. A title containing HTML or an event handler can become executable when a catalog, service row, shortlist, or booking is rendered and can persist across reloads. The scope is this browser origin; remote sharing is not implemented, but all same-origin demo records are exposed to injected code.

Use DOM.textContent and safe attribute assignment for untrusted text, with explicit validation of records and URLs. Add server-side ownership enforcement before publishing this as a real marketplace. These are recommended changes; documentation work does not modify the application.

Browser Storage Contracts

`read(key, fallback)` parses JSON and falls back on parsing/storage-access errors or null. `write(key, value)` serializes to JSON and calls `localStorage.setItem` without error handling. Records are scoped to origin, shared by that origin's tabs, and not transmitted to a server.

Key	Stored value
skillconnect-users	Account array: UUID, name, normalized email, SHA-256 hash, role
skillconnect-user	Current session object, removed on logout
skillconnect-bookings	Array of service/email/locale-date request records
skillconnect-saved	Array of talent or listing names, not IDs
skillconnect-services	Whole catalog after CRUD; only owner-marked entries restored

Reload reconciliation

The baseline catalog is reconstructed on every load. Stored owner records form a name-keyed Map. A baseline profile with a matching name is replaced by that saved record, and owned entries with names not present in the baseline are appended. Stored fictional non-owner records are otherwise ignored.

A created title matching a seeded person can replace that person on reload. Multiple owner entries with the same seeded name collapse through the Map. Unmatched duplicate titles can remain as separate rows, but shortlist lookup still returns only the first matching name. UUIDs exist on created services but are not used consistently as references.

Resilience and privacy

A syntactically valid but incorrectly shaped JSON value can still crash filtering or rendering because record shapes are not validated. Storage quota or blocked-write errors can interrupt a submit midway. There is no storage-event synchronization; cached users/bookings/catalog arrays can overwrite another tab's newer changes. A logout in one tab does not invalidate another tab's in-memory session.

Clearing browser/site data deletes all demo accounts and requests for that origin. Moving from localhost to Vercel starts with separate state. Anyone with access to the browser profile can inspect the stored data. Logout is not data deletion, and no export, migration, retention policy, or recovery interface exists.

Future data contract

A server-backed version should separate profiles from services, identify records with stable IDs, tie lists/requests/services to authenticated users, preserve all submitted booking fields, use ISO timestamps, validate input shapes, and define explicit migration/error behavior. The SQL reference supplies a starting point, not a drop-in replacement for these browser objects.

Reference PostgreSQL Model

database/schema.sql declares pgcrypto UUID defaults, four enum types, seven tables, and three indexes. It is not executed by npm build, the preview server, or Vercel. The relationships below describe a proposed design, not deployed persistence.

Table	Relationship and purpose
users	Identity, unique email, password hash, role and timestamps
freelancer_profiles	Unique user_id: at most one profile per user; bio/skills/location/rating
services	Many services per user through freelancer_id; price, delivery and status
bookings	Client + freelancer + service references; brief, price, statuses and dates
booking_status_history	Many history rows per booking; previous/new status and actor
payments	At most one payment per booking; provider reference and outcome
reviews	At most one review per booking; client/freelancer and 1-5 rating

Types and constraints

Roles: CLIENT, FREELANCER, ADMIN. Service states: DRAFT, PUBLISHED, PAUSED, REMOVED. Booking states: PENDING_PAYMENT, PAID, ACCEPTED, IN_PROGRESS, COMPLETED, REJECTED, CANCELLED. Payment states: PENDING, SUCCEEDED, FAILED, REFUNDED.

Service/booking prices are nonnegative and delivery days must be positive. Review ratings are constrained to 1-5. Profile/service removal cascades from a deleted owner; booking history, payment, and review removal cascade from booking deletion. Booking participant/service references do not cascade. Indexes cover service category/status and each booking participant.

What the schema does not enforce

Foreign keys do not guarantee correct user roles, that a booking's freelancer owns its service, that a reviewer is the booking client, or that a booking is complete before review. Enum values do not enforce allowed state transitions. There are no automatic history writes, rating aggregates, updated_at triggers, access policies, or payment reconciliation.

Payment amount lacks a nonnegative check; profile coordinates, ratings, experience, and counts lack bounds. Email uniqueness is case-sensitive. Most type/table creation is not rerunnable without migration handling. Before integration, add a trusted API, transactional business rules, migrations, account authorization, and verified provider events. No SQL execution was performed for this report.

Presentation and Accessibility

The interface uses a light editorial palette, blue primary actions, coral/mint/lavender category accents, remote collaboration photography, and serif-like italic emphasis inherited through CSS. DM Sans, DM Mono, and Space Grotesk are requested from Google Fonts. Later CSS overrides must be read alongside the initial compressed rules.

Desktop uses four talent columns and five categories. At 800px and below, navigation becomes a toggle menu, major sections stack, and talent/categories use two columns. At 420px and below, talent becomes single-column and form rows stack. Dialogs constrain width and scroll vertically.

Implemented foundations include semantic landmarks, labels, named icon buttons, image alt text, dialog roles/names, aria-hidden, mobile aria-expanded, auth role=alert, and toast role=status. These are not a complete accessibility compliance claim.

Missing behavior includes modal focus trapping, background inert handling, reliable focus restoration, ARIA tab selection/keyboard semantics, pressed states for filters/saves, and live result announcements. Search/sort suppress focus outlines without a replacement; smooth scrolling has no reduced-motion handling. Very long titles need compact-row wrapping checks.

Running and Publishing

Local preview

```
# Node.js 18 or later; run in the project folder
npm install
npm run dev
# http://localhost:3000
npm run build
```

There are no npm runtime dependencies. dev.mjs imports the build, then serves four allowlisted paths: /, /index.html, /styles.css, and /app.js. It does not watch source changes. Restart the preview or rebuild after edits. If port 3000 is occupied, PowerShell can use \$env:PORT="3001" before npm run dev.

Static production deployment

build.mjs recreates dist/ with index.html, styles.css, and app.js only. Vercel uses framework null, npm run build, and outputDirectory dist. The catch-all rewrite targets index.html. Headers set X-Frame-Options DENY, X-Content-Type-Options nosniff, and strict-origin-when-cross-origin; no CSP is configured.

Select the correct repository root and Framework Preset Other. The deployment guide warns that an Express entrypoint error can indicate an old commit or preset. No secret or server environment variables are required. The documented public URL is <https://skillconnect-phi-gules.vercel.app/>; this report does not claim a fresh live-site check.

Maintenance and Verification

Routine source changes

Catalog: update seed/generation data and keep static category totals aligned. Prefer stable IDs before supporting renames. Service changes: update both display and booking fields and clear or migrate stale shortlist references. Forms: synchronize visible inputs, validation, and the persisted object; collecting a field does not make it durable.

Styles: inspect the full cascade, especially the 800px and 420px rules and later hero overrides. Deployment: keep only public assets in the build allowlist; the documentation PDF is not automatically published. Database: treat schema changes as future design until an API and migration workflow are implemented.

Symptom	Where to start
Saved data is missing on another device	Expected browser/origin-local storage; not server sync
Login fails after browser data is cleared	The local users array was removed; create a new demo account
Booking lost its brief/date/budget	app.js:70 persists only service, email and created date
Renamed service books its old title	app.js:74 does not update the service field on edit
Category counts do not match listings	index.html:44 contains hardcoded category totals
Another account sees my requests	Shared arrays have no authenticated owner identifiers
Edits are absent in local preview	Restart/rebuild: dev.mjs has no file watcher

Release checklist

Test registration, duplicate emails, logout/login, wrong passwords, and reload persistence with disposable credentials. Test every category, search, price/rating sorting, empty results, View all, shortlist add/remove, and unauthenticated saves. Submit a booking and inspect both the dashboard and stored fields.

Create, edit, cancel, delete, and reload listings. Exercise duplicate/seeded names, long titles, hostile HTML, multiple accounts, simultaneous tabs, blocked storage, and malformed JSON. Check desktop and narrow layouts, keyboard focus, stacked-dialog closing, external-image/font failures, and HTTPS Web Crypto support.

Verification during documentation: source inspection, `node --check app.js`, `npm run build`, and a headless desktop cover capture. No automated test script exists in `package.json`. Full interaction/security/accessibility tests and PostgreSQL/live-host checks remain unperformed; the checklist is guidance, not reported pass results.

Source Navigation

References point to the inspected project snapshot and may shift after edits. Much of app.js and styles.css is compressed into long lines; a line can contain an entire workflow. Use the named functions and selectors to narrow a search.

Location	What to read
index.html:8 / :26	External fonts and hero/search markup
index.html:44 / :47	Category totals, catalog grid and controls
index.html:62 / :63 / :64	Booking, authentication and studio dialogs
app.js:1 / :22 / :30	Seed profiles, generated catalog and baseline merge
app.js:28 / :31 / :38	Storage helpers, service restoration and session check
app.js:43	renderTalent: search/filter/sort/cards and shortlist actions
app.js:44 / :46 / :47	Dialog lifecycle, booking opening and auth-mode switch
app.js:48 / :49	SHA-256 helper and account-navigation visibility
app.js:50 / :51 / :52	Dashboard, shortlist and owned-service rendering
app.js:53 / :54	Service edit population and deletion
app.js:55 / :56 / :67	Category/search/sort listeners and reveal-all action
app.js:59 / :65 / :66	Mobile menu and studio entry routes
app.js:70 / :71 / :72	Booking persistence, account submit and logout
app.js:73 / :74 / :76	Studio tabs, service submit and Escape/initial render
styles.css:1 / :2 / :13	Base styling, responsive rules and later overrides
database/schema.sql:1-5 / :6-12 / :13	Enums, proposed tables and indexes
build.mjs:3 / dev.mjs:1	Public-asset build and local HTTP preview
vercel.json:2-13	Framework/build/output, rewrite and response headers
README.md:15 / :28 / :69	Demo scope, reference schema and production next steps
DEPLOYMENT.md:18 / :25	Hosting setup and manual live-release checklist

The current project demonstrates discovery, local persistence, dialog workflows, and service CRUD. A production marketplace requires a trusted identity/data layer and explicit transactional business rules; the directory name does not establish production readiness.