CarbonBalance Frontend – Technical Doc.

Prepared for: CostPlan Group / Tech-Titans (QUT IFB399 Capstone)
Scope: React frontend only (API consumed: CarbonBalance backend)

Executive Summary

What it is. A React SPA for construction sustainability planning: set up a project, weight sustainability themes, get ranked interventions, and view results.

Who uses it. Project teams and admins (admin can create users).

Core flows. Login → Create Project (+metrics) → Theme Rating → Intervention Selection (batch apply) → Results (matrix/report) → View Existing Projects.

Run & deploy (2-minute).

- Install: `npm ci` (or `npm install`)
- Dev: `npm start` (proxy to backend).
- Prod build: `npm run build`
- API base URL: set `REACT_APP_API_BASE` for prod; dev uses `/api` proxy.
- Deploy (GH Pages): `npm run deploy` (requires repo Pages + `homepage` set).

API cheatsheet (frontend calls).

- Auth: `POST /auth/login`
- Admin: `POST /admin/users`
- Projects: `POST /projects`, `GET/PATCH/DELETE /projects/:id`, `GET /users/:id/projects`
- Themes: `GET /themes`, `GET /projects/:id/theme-scores`, `POST /projects/:id/themes`
- Metrics: `POST /projects/:id/metrics`
- Recommendations: `GET /projects/:id/recommendations`
- Interventions: `POST /projects/:id/apply-batch` (and legacy `POST /projects/:id/apply`)
- Implemented: `GET /projects/:id/implemented`

Pages & routes (snapshot).

- `/login` (Login) • `/admin` (AdminCreateUser) • `/dashboard` (Dashboard)
- `/create-project` (CreateProject) • `/theme-rating` (ThemeRating)
- `/intervention-selection` (InterventionSelection) • `/results` (ResultsMatrix)
- `/data-ingestion` (DataIngestion) • `/view-existing` (ViewExistingProjects)

Ownership. Frontend: CarbonBalance (Tech-Titans). Backend/API: CostPlan/CB.
Contacts: Add repo URL, API base(s), and on-call emails here before handover.

Known gaps / next steps.

- Results/reporting page: currently sample data + placeholder export; bind to backend & generate PDF/Excel.
- Data ingestion: wire upload/download to real endpoints with validation & progress.
- Auth UX: add logout and protected routes/role-based guards.
- Testing: extend Jest/RTL; add E2E (Playwright) and API mocks.
- Design system: migrate inline styles to Tailwind utilities; shared tokens.
- A11y & perf: focus rings, aria labels, code-split heavy pages, image optim.
- Ops: environment documentation, Sentry/analytics, CSP, 404/500 UX.

1) System Overview

CarbonBalance is a web SPA that guides construction teams from project setup → theme prioritisation → intervention selection → results. The frontend is built with Create React App and modern React, styled with Tailwind-compatible PostCSS and custom brand system fonts (Arquitecta). It consumes a REST API for authentication, project data, scoring, and recommendations.

Primary user flows

1. Authenticate → Login with email/password.
2. Create Project → Enter project metadata and quantitative metrics.
3. Rate Sustainability Themes → Weight each theme (0–100).
4. Get Recommendations → Review/select recommended interventions and apply them (batch).
5. Results → View matrix (dependencies/report placeholders) and export report.
6. Manage Projects → Browse existing projects (filter/sort), open or create new.

2) Technology Stack

- Runtime: Node.js (local dev), Browser (production)
- Framework: React (CRA bootstrapped)
- Routing: react-router-dom
- HTTP: Fetch-based client wrapper (src/api.js)
- State: Local component state + localStorage for auth/session
- Styling: Tailwind-compatible PostCSS + custom CSS (Arquitecta fonts)
- Testing: Jest + React Testing Library (CRA defaults)
- Performance: Web Vitals reporting utility
- Deployment: GitHub Pages (via gh-pages)

3) Repository Layout (Frontend)

```
src/
  api.js              # API client & auth helpers
  App.js               # Router + route map
  App.css               # Global brand styles (Arquitecta)
  index.js / index.css     # CRA entry & base CSS
  reportWebVitals.js      # Performance metrics (optional)
  setupTests.js          # RTL/Jest DOM matchers
  components/
    common/
      Layout.jsx         # Shell, header/footer control per route
      Header.jsx         # Top nav (hidden on login/admin)
    Footer.jsx          # Footer (always on)
  pages/
    Login.jsx           # Auth page
    Dashboard.jsx         # Post-login landing & CTAs
    CreateProject.jsx      # Project creation + metrics submit
    ThemeRating.jsx        # Theme weights (0–100), save
    InterventionSelection.jsx  # List + select + apply interventions
    ResultsMatrix.jsx      # Dependency matrix + export placeholder
    DataIngestion.jsx      # Download/upload placeholder (Excel)
    ViewExistingProject.jsx # User's projects list (filter/sort)
    AdminCreateUser.jsx     # Create user (admin)
```

Assets: multiple pages reference background images (e.g., newbg.jpg, newdashbg.jpg) via process.env.PUBLIC_URL. Ensure these exist under public/ in the deployed app.

4) Build, Install & Run

Prerequisites

- Node.js LTS (18+) and npm 8+
- Backend reachable at development proxy or via REACT_APP_API_BASE

Environment

- Development (default): uses CRA dev server with a local proxy to backend.
- Production: set REACT_APP_API_BASE to your backend URL at build time.

Example .env (optional):

```
REACT_APP_API_BASE=https://api.example.com
```

Install

```
npm install
# or for reproducible installs
npm ci
```

Run (development)

```
npm start
# opens http://localhost:3000 with proxying to the backend
```

Test

```
npm test
```

Build (production)

```
npm run build
```

Deploy (GitHub Pages)

```
# ensure repository settings & homepage configured
npm run deploy
```

Note: A CRA "proxy" is configured for local development. For production, you must use REACT_APP_API_BASE so the compiled bundle calls the correct backend.

5) Application Routing & Pages

Top-level router is defined in App.js. The Layout component shows/hides the Header for login/admin routes and always renders the Footer.

Route Map

Route	Component	Purpose	Notes
/login	Login.jsx	Email/password sign-in; persists user + token; redirect to Dashboard	Hides header (Layout), shows footer
/admin	AdminCreateUser.jsx	Admin creates a user with role + default access level	Hides header (Layout), shows footer
/dashboard	Dashboard.jsx	Post-login landing; hero content; CTA to create/view projects	Background with hero copy
/create-project	CreateProject.jsx	Form for new project; submits metrics; navigates to Theme Rating	Persists current project id
/theme-rating	ThemeRating.jsx	Fetch themes & any saved scores; save sliders (0–100)	Moves to Intervention Selection
/intervention-selection	InterventionSelection.jsx	Displays recommended interventions; multi-select; apply batch	Auto-refresh/redirect when done
/results	ResultsMatrix.jsx	Dependency matrix of interventions; export placeholder	Future: full report/export
/data-ingestion	DataIngestion.jsx	Download sample report; upload staged files (placeholder)	Future: integrate backend ingest
/view-existing	ViewExistingProject.jsx	List user's projects w/ filter/sort; open or create	Uses token to fetch "my projects"
/	redirect → /login	Default entry	Unknown paths → /login

6) API Client & Endpoints

The API client (src/api.js) centralizes authentication and request logic, with JSON fetch, 401 handling, and helpers to read/write localStorage (token, auth, user). All methods return JSON data or throw a descriptive Error when non-2xx.

Auth Helpers

- Token management: setToken(t), getToken(), clearToken()
- Auth object: setAuth(obj), getAuth(), getUserId()
- On 401 responses, the client auto-clears token to avoid stale sessions.

Request Helper

```
request(path, { method = 'GET', body, headers })
# Adds JSON headers, Authorization if token present, parses response JSON.
# Throws with server error/message/statusText when !res.ok.
```

API Methods (with typical call sites)

Area	Method	HTTP	Path	Called From
Auth	login(email, password)	POST	/auth/login	Login.jsx
Admin	createUser({ name, email, password, role, default_access_level })	POST	/admin/users	AdminCreateUser.jsx
Projects	createProject(payload)	POST	/projects	CreateProject.jsx
	getProject(projectId)	GET	/projects/:id	(future detail pages)
	patchProject(projectId, partial)	PATCH	/projects/:id	(future edits)
	deleteProject(projectId)	DELETE	/projects/:id	(future)
	listProjectsByUser(userId)	GET	/users/:id/projects	(internal)
	listMyProjects()	GET	/users/{current}/projects	ViewExistingProject.jsx
Themes	listThemes()	GET	/themes	ThemeRating.jsx
	getProjectThemeScores(projectId)	GET	/projects/:id/theme-scores	ThemeRating.jsx
	saveProjectThemes(projectId, weights, {dryRun})	POST	/projects/:id/themes	ThemeRating.jsx
Metrics	sendBuildingMetrics(projectId, metrics, {dryRun})	POST	/projects/:id/metrics	CreateProject.jsx
Interventions	applyIntervention(projectId, interventionId, {dryRun})	POST	/projects/:id/apply	(kept for compatibility)
	applyInterventionsBatch(projectId, ids, {dryRun})	POST	/projects/:id/apply-batch	InterventionSelection.jsx
Recommendations	getRecommendations(projectId)	GET	/projects/:id/recommendations	InterventionSelection.jsx
Implemented	getImplementedInterventions(projectId)	GET	/projects/:id/implemented	(future Results page)

Base URL resolution

- API_BASE = process.env.REACT_APP_API_BASE || '/api'
  In dev, CRA proxy can forward /api to the backend; in prod, set REACT_APP_API_BASE.

7) Page Responsibilities & Data Flow

Login

- Goal: authenticate, persist session, and redirect to the dashboard.
- Flow: submit → api.login → save { user, access_token } to storage → /dashboard.
- UX: password toggle, loading state, inline error presentation.

Dashboard

- Goal: post-login landing & CTAs; hero message; neutral page (no API fetch).
- Next steps: create project / view existing.

Create Project

- Goal: create a project and immediately send quantitative metrics to backend.
- Flow: submit base info + numeric fields → api.createProject → api.sendBuildingMetrics → store projectId → /theme-rating.
- Validation: numeric casting only when fields provided; lenient defaults for optional fields.

Theme Rating

- Goal: set or update theme weights (0–100) per project.
- Flow: load theme catalog → (optionally) load existing theme-scores for project → submit → /intervention-selection.
- Extras: "No Preference" sets all sliders to 100.

Intervention Selection

- Goal: view ranked recommended interventions, select multiple, and apply in batch.
- Flow: read projectId from router state/URL/localStorage → getRecommendations → sort by theme_weighted_effectiveness → user selects → applyInterventionsBatch → if next_recommendations present, refresh; when complete, redirect to /results.
- UX: mobile-friendly list; action buttons; helpful toasts/alerts.

Results Matrix

- Goal: visualise intervention dependencies; prepare for export/reporting.
- Status: sample data + placeholder export (alert). Future: integrate server data and reporting pipeline.

Data Ingestion

- Goal: provide a download artifact and upload channel for batch intervention data.
- Status: placeholders for download and file-select; future: wire up to backend endpoints.

View Existing Projects

- Goal: quick access to recent projects with filter and sort.
- Flow: listMyProjects() uses current token to infer user → shows top 6 by recency; clicking row: navigate to project (route TBD).

Admin Create User

- Goal: allow Admin to create new users with role + default access level.

- Flow: validate email → createUser() → success alert → redirect to /login.
- Error handling: surface email_exists or API error messages.

8) Authentication, Session & Security

- Token: stored in localStorage and attached as Authorization: Bearer <token> for API requests.
- Auth object: entire server response persisted to localStorage for quick access to user.id and role.
- 401 handling: API client clears token automatically → client can redirect to login.
- Considerations (future hardening):
    - Prefer HTTP-only cookies for tokens where feasible.
    - Add logout control to clear token & auth.
    - Centralise route guarding (e.g., ProtectedRoute) to prevent unauthenticated access to protected pages.

9) Styling & Design System

- Fonts: Arquitecta (Black/Light) declared via @font-face and applied globally.
- Brand: Primary blue rgb(50, 195, 226), greyscale backgrounds, orange accent; glass-blur card motif.
- Responsiveness: inline window.innerWidth <= 768 checks for mobile breakpoints across pages.
- Future: migrate inline styles to Tailwind utility classes; establish shared tokens (spacing, colors, radii) and components.

10) Error Handling & UX States

- API layer throws descriptive Errors (message derived from server/body/status).
- Pages commonly present inline banners/alerts for error states.
- Common patterns:
    - Loading: skeleton text or "Loading …" messaging.
    - Validation: simple regex for email; numeric casting for project metrics.
    - Recovery: selection reset after applying interventions; re-fetch on failure.

11) Testing & Quality

- Jest + React Testing Library are configured.
- Included sample test (App.test.js) to verify initial render; extend with page-level tests:
    - Login submission & error case
    - Create Project form validation (numbers present → numeric casting)
    - Theme slider interactions
    - Intervention selection flows (apply; next recommendations)

Web Vitals: reportWebVitals.js can be wired to analytics for CLS, LCP, FID, etc.

12) Configuration & Deployment

- CRA Proxy: development requests to /api are forwarded to the backend (see backend dev server and port). This avoids CORS in dev.
- Production Base URL: set REACT_APP_API_BASE at build time to production API.
- GitHub Pages: deploy via npm run deploy; ensure homepage is set to the repository pages URL and repository permissions allow Pages deployment.
- Browserslist: CRA default targets for modern evergreen browsers.

13) Troubleshooting

- 401 Unauthorized: token expired/invalid → login again; ensure backend CORS and API base URL are correct.
- Network errors in dev: check that the backend is running and the proxy target matches your backend origin.
- Blank screen after deploy: confirm homepage path, ensure static assets (e.g., newbg.jpg, fonts) exist in public/, and that the repo Pages environment is active.
- Fonts not loading: verify font files exist under src/assets/fonts with correct filenames and that bundling includes them; confirm correct relative URLs.

14) Roadmap / Future Improvements

- Results/reporting: bind matrix to real project interventions and generate downloadable PDF/Excel.
- Data ingestion: implement backend endpoints for secure upload & validation.
- Routing guards: centralised protected routes; role-based visibility in nav.
- Design system: Tailwind utilities + component library to replace inline styles; consistent breakpoints.
- Accessibility: keyboard focus styles, semantic landmarks, aria labels for controls.
- Performance: code-split heavy pages; memoize lists; image optimisation.
- Testing: expand to E2E (Playwright), API mocks, and coverage targets.