MedLearn Project Overview

MedLearn is a training and onboarding LMS for pharmacies. The app delivers public-facing marketing and inquiry capture, plus an authenticated admin dashboard for approvals, invitations, and course management prototypes.

Tech stack

Frontend Parcel bundler, vanilla JS modules, Tailwind CDN, modular CSS partials Backend Node.js + Express, cookie-session, express-validator, mysql2 Database MySQL (schemas for inquiries, invites, users, companies) Email Mailgun (transactional: approvals, invites)

Project layout

```
src/
  client/
    public/
      assets/
        css/
          styles.css           entry point
          settings.css         CSS variables
          base.css             resets + base type
          layout.css           layout, grid, header, footer
          components.css       buttons, cards, nav, forms
          utilities.css        helpers
          vendor.css           vendor overrides + imports
        js/
          lib/                 shared modules
            main.js            shared nav, alerts, authFetch, bootstrapping
          public/              public pages
            index.js
            inquiries.js
            signin.js
          secure/              secure pages
            dashboard.js
      index.html
      courses.html
      inquiries.html
      signin.html
    secure/
      dashboard.html
  server/
    server.js                  Express app + static serving
    routes/
```

```
      inquiriesRoute.js
      usersRoute.js
      companiesRoute.js

dist/                         Parcel output (gitignored)
```

Static serving and URLs

Express serves `src/client/public` as the web root, so runtime paths do not include `/public`. Examples

`/` → `index.html` `/inquiries.html` → `src/client/public/inquiries.html` `/assets/...` → `src/client/public/assets/...`

CSS organization

`styles.css` uses ordered imports to keep cascade predictable

1 settings 2 base 3 layout 4 components 5 utilities 6 vendor

Each selector in partials includes a brief line comment explaining intent. The look and behavior were kept identical to the original single-file stylesheet.

Vendor CSS and intl-tel-input assets

Goal avoid Parcel ambiguity for custom properties that reference images.

Final approach

1 Import the package CSS from JS so Parcel owns module resolution

```
// in assets/js/lib/main.js (or the first page that needs phone input)
import 'intl-tel-input/build/css/intlTelInput.css';
```

2 Place the flag and globe images in public assets so absolute URLs are unambiguous

```
src/client/public/assets/vendor/intl-tel-input/img/
  flags.webp
  flags@2x.webp
  globe.png
  globe@2x.png
```

3 Override the package CSS custom properties once, in `assets/css/vendor.css`

```
:root {
  --iti-path-flags:     url("/assets/vendor/intl-tel-input/img/flags.webp");
  --iti-path-flags-2x:  url("/assets/vendor/intl-tel-input/img/flags@2x.webp");
  --iti-path-globe-1x:  url("/assets/vendor/intl-tel-input/img/globe.png");
  --iti-path-globe-2x:  url("/assets/vendor/intl-tel-input/img/globe@2x.png");
}
```

This keeps all vendor assets stable under `/assets/vendor/...` and avoids relative URL resolution inside CSS variables.

JavaScript organization

Shared code lives in `assets/js/lib/` and is imported by per-page scripts under `assets/js/public` or `assets/js/secure`. Examples

`main.js` nav/menu, toast/alert helpers, authFetch, any global bootstrapping `public/inquiries.js` form validation, NPI lookups, submission UX `secure/dashboard.js` admin controls, table actions

Parcel tree-shakes and bundles per HTML entry, so you keep shared logic DRY while isolating page-only code.

Backend overview

Express server

serves static files from `src/client/public` at `/` exposes REST routes under `/api/*` uses cookie-session for auth, express-validator for inputs

Key flows

Inquiry approval and Invite creation

Admin approves an inquiry → server creates an `invites` row with a random plain token and a hashed token stored in DB → email contains the plain token in a one-time URL → onboarding endpoint verifies by comparing hashed values and checks expiration.

Email

Mailgun is used for confirmations and invite delivery. HTML templates are kept simple and tested via dev mailboxes or your own email.

Database notes

Tables of interest

inquiries capture business inquiries, statuses pending|approved|rejected invites one-time onboarding links tokenUsed flag, expirationTime default now() + 72 hours

Quick maintenance

```
SET FOREIGN_KEY_CHECKS = 0;
TRUNCATE TABLE invites;
TRUNCATE TABLE inquiries;
SET FOREIGN_KEY_CHECKS = 1;
```

Build and run

Dev

`npm run dev` runs clean, Parcel watch, and Express server concurrently

Output

Parcel writes to root `dist/` only. `dist/` is gitignored. There should be no `dist/` inside `src/client/public`.

Coding conventions

Use absolute root paths in HTML for assets `/assets/...` Keep shared JS in `lib/`, page code under `public/` or `secure/` Keep vendor tweaks in `vendor.css` only One stylesheet link per page `/assets/css/styles.css`

Open tasks and ideas

Finish email template polish for approvals and invites Welcome page content checklist after invite link land explain org, required profile fields, first password set Admin dashboard highlights filter/search inquiries, invite resend, simple metrics Optional add a small postcss step later if you want autoprefixing from npm instead of Tailwind CDN

How to propose changes

Create a branch named `feature/<topic>` Keep changes scoped and cohesive Submit a single PR with a clear description and screenshots for UI touches

End of overview