Waves Learning Platform — Approach and Design Decisions

1) Goals and Scope

- Build a formal, professional educational web app focused on the topic of Waves.
- Provide authenticated access, a dashboard, rich interactive content, and a 10-question quiz with explanations.
- Ensure persistence across reloads (stay on the same page if logged in; reset quiz answers on quiz reload).
- Deploy as a modern SPA with correct routing on Vercel and other static hosts.

2) Tech Stack and Rationale

- **React 18 + TypeScript + Vite**: Fast DX, type safety, efficient dev/build, great HMR.
- **react-router-dom**: Client-side routing and protected routes.
- **HTML5 Canvas**: High-performance 2D rendering for wave simulations using `requestAnimationFrame`.
- **LocalStorage**: Lightweight persistence for session and UI state (no backend in scope).
- **Vercel**: Zero-config, global CDN, and SPA routing rewrites.

3) Architecture Overview

- **Project Structure** (core):
  - `src/App.tsx`: App shell, routes, auth state, persistence.
  - `src/components/Login.tsx`: Email/password validation; triggers login.
  - `src/components/Dashboard.tsx`: Course cards, sign-out.
  - `src/components/WavesCourse.tsx`: Two tabs (Content, Quiz); content side-nav (Overview, Transverse, Longitudinal).
  - `src/components/WaveLab.tsx`: Canvas-based simulations (single, interference, standing), controls.
  - `src/App.css`: Systematic, formal UI theme and responsive layout.
- **Routing**:
  - `/` → Login (redirect to `/dashboard` if already authenticated)
  - `/dashboard` → Protected dashboard
  - `/waves-course` → Protected course content + quiz
  - Catch-all redirects appropriately based on auth

4) Authentication and Persistence

- **Client-side validation**:
  - Email: contains `@` and `.`
  - Password: ≥ 8 chars, ≥ 1 uppercase, ≥ 1 digit
- **Session state**: `localStorage.user` holds `{ email, name }`.
- **Route persistence**: On reload, current route remains; protected routes redirect only if no `user`.
- **Course tab persistence**: `localStorage['waves-course-tab']` stores `content` or `quiz`.
- **Quiz reset rule**: Entering the Quiz tab clears selected options and score (per requirement).

5) Content Architecture

- **Side navigation** inside Content tab:
  - Overview → conceptual intro, parameters, visuals using `WaveLab`.

    - Transverse → definition, properties, math (e.g., (y(x,t)=A\sin(kx-\omega t+\phi))).
    - Longitudinal → compression/rarefaction, particle displacement, relevant equations.
- **Each section** combines short theory, key ideas, parameter controls, and live simulation.

6) Simulations (WaveLab)

- **Canvas loop** via `requestAnimationFrame` with controlled delta-time.
- **Modes**:
    - Single wave
    - Interference (superposition of two waves)
    - Standing wave (counter-propagating waves)
- **Parameters**: amplitude, frequency, phase, speed, wavelength; toggles for grid, axes, markers.
- **Design choices**:
    - Canvas instead of SVG for better performance at high sample counts.
    - Centralized draw loop; pure math functions for waveforms; input debouncing where appropriate.
    - Guard clauses for null canvas context; cleanup of animation frame on unmount.

7) Quiz Design

- 10 fixed multiple-choice questions covering concepts and math.
- Per-question explanations shown after submission.
- Score computed client-side; UI disables editing post-submit until the quiz tab is re-entered (which resets state).

8) UI/UX System

- **Formal theme**: Muted primary, neutral surfaces, strong contrast for text.
- **Layout**: Clear typographic hierarchy; generous spacing; consistent card and button styles.
- **Responsive**: Flex/grid layouts adapt down to mobile; simulations scale with container.
- **Accessibility**: Sufficient color contrast; focusable controls; semantic headings and landmarks.

9) Performance Considerations

- **Build-time splitting**: `manualChunks` for `vendor` and `router` to improve caching.
- **Vite optimizations**: Modern ESM build; no source maps in prod.
- **Canvas**: Efficient sampling, avoids unnecessary reflows; only recompute when parameters change.
- **React**: Keep state localized; avoid unnecessary re-renders with stable props and effects.

10) Security Considerations

- Client-only auth (no backend) by design: suitable for demos/learning, not for sensitive data.
- Inputs validated on client; avoid storing sensitive info in `localStorage`.
- Future hardening would include server-side auth (tokens), HTTPS-only cookies, and rate-limited APIs.

11) Deployment Design (Vercel-first SPA)

- `vercel.json` rewrites all paths to `index.html` so React Router can handle deep links.
- **Vite** `base: '/'` ensures correct asset paths.
- `public/_redirects` included for cross-host SPA compatibility.
- **Scripts**: `build` compiles TS and bundles with Vite; Vercel uses `npm run build`.

12) Trade-offs and Alternatives

- **LocalStorage vs Backend**: Faster to implement and meets persistence goals; not secure for real auth.
- **Canvas vs SVG**: Canvas chosen for performance under continuous animation; SVG better for static/DOM-integrated visuals.
- **Fixed Quiz vs CMS**: Fixed set provides reliability and simplicity; a CMS/API would enable updates without redeploy.

13) Future Enhancements

- Real backend auth (JWT/cookies), user profiles, and progress tracking.
- Authoring tools or CMS for content/quiz management.
- More wave types (surface, electromagnetic) and measurement tools (cursors, FFT).
- Internationalization (i18n) and richer accessibility testing.

14) Build and Deploy (Quick)

1. Install deps: `npm install`
2. Dev server: `npm run dev`
3. Production build: `npm run build` → outputs to `dist/`
4. Vercel deploy: Upload `dist/` or connect the repo; ensure `vercel.json` is present.