Overview

Main Tab – Project Overview

Echoes of Pharloom

Echoes of Pharloom is a focused Pomodoro study companion that helps users stay engaged, mindful, and productive through guided study sessions.

It balances structure and creativity: offering personalized session plans, immersive ambience, and progress tracking powered by modern web technologies.

---

1. Goal of the App

Echoes of Pharloom aims to support consistent focus through:

- **Structured Study Plans** – Users create or randomize session layouts with breaks.
- **Motivational Feedback** – Audio, visual, and streak indicators encourage daily consistency.
- **Frictionless Flow** – Session persistence, focus detection, and background ambience provide a calm, distraction-free environment.

Target users are students and professionals seeking a guided study environment with mood and accountability components.

Design principles:

- **Simplicity** – Minimal clicks and clean layout.
- **Resonance** – Audio and visuals enhance the user's mindset.
- **Transparency** – Every feature serves user agency and mental flow.

---

2. High-Level Architecture

Echoes of Pharloom is composed of three primary layers:

1. **Frontend (React + TypeScript)** – Handles user experience, authentication, and local state management.

2. **Services (Node.js AWS Lambdas)** – API layer that stores user data, handles sessions, profiles, and feedback.

3. **Infrastructure (AWS CDK)** – Automates the provisioning of Cognito, API Gateway, S3, and DynamoDB.

Data flows primarily through **HTTP JSON APIs** with token-based authentication via Cognito.
Frontend → API Gateway → Lambda → DynamoDB/S3.

---

3. Core Data Flow

1. **Authentication:**
   User signs in via Cognito hosted UI (PKCE). Tokens stored in localStorage.

2. **Session Creation:**
   The frontend constructs or randomizes a study plan.
   Authenticated users' sessions are stored in DynamoDB via createSession.

3. **Study Runtime:**
   App.tsx manages timer phases, events, and ambience.
   Events are appended locally and optionally sent to /sessions/{id}/events.

4. **Profile Management:**
   The user can upload a profile photo → frontend requests a pre-signed S3 URL → uploads file → updates profile record via /profile.

5. **Feedback Flow:**
   FeedbackModal → POST /feedback → backend validates + sends email via Resend → returns success response.

---

4. Tech Stack Summary

Layer	Technologies	Notes
Frontend	React, TypeScript, MSW, Amplify Hosting	SPA, local storage caching, Cognito auth
Backend	Node.js (Lambda), API Gateway, DynamoDB, S3	Stateless microservice design
Infra	AWS CDK (TypeScript)	One-stack deployment managing all resources
Auth	AWS Cognito (PKCE Flow)	Handles tokens, password reset, hosted UI
Email	Resend API	Handles feedback email delivery
Hosting	AWS Amplify	Automatic builds and deploys for frontend

---

5. Quick Links

- **GitHub Repository:** Link
- **Technical Documentation:** Google Doc – Echoes of Pharloom Overview
- **Amplify App URL:** *(to be added post-deploy)*
- **AWS Console Shortcuts:**
    - DynamoDB Table
    - S3 Bucket (ProfilePhotos)
    - Cognito User Pool
    - API Gateway Console

Frontend

Overview

The frontend is a **React + TypeScript** single-page application hosted on **AWS Amplify**.
It manages user authentication, study sessions, streak tracking, and interactive ambience through modular components.

---

1. Routing and Entry Point

File: `src/index.tsx`

- Bootstraps React in `StrictMode`.
- Enables Mock Service Worker (MSW) in development (`enableMocks`).
- Mounts `BrowserRouter` with the following routes:
  - `/` → **Home**
  - `/study` → **Study Page**
  - `/create` → **Create Session**
  - `/profile` → **Profile**
  - `/info` → **Info**
  - `/auth/callback` → **AuthCallback**

---

2. Authentication System

File: `src/auth/AuthContext.tsx`

- Wraps the app with `AuthProvider`.
- Implements **PKCE OAuth2 flow** for Cognito Hosted UI.

- Stores `id_token` and `access_token` in localStorage.
- Provides `signIn()`, `signOut()`, and `getUser()` helpers.
- Supports a **mock local user** mode in development.

---

3. Study Engine

File: `src/App.tsx`

Handles the full lifecycle of a study session:

- Loads session plan from `location.state`, `localStorage`, or fallback.
- Persists `activePlan` locally to survive reloads.
- Manages phases: running, break, completed.
- Records events (`startSession`, `appendEvent`, `completeSession`) both locally and optionally via backend.
- Detects focus loss and displays toasts.
- Integrates `audioManager.ts` and `videoManager.ts` for ambient feedback.
- Interfaces with `TimerControls`, `HUD`, and `VolumeControl`.

---

4. Create Session

File: `src/pages/CreateSession.tsx`

- Manual or randomized session creation modes.
- Inputs total session length and number of segments.
- Enforces minimum break length (1 minute).

- Generates session plan and stores it for `/study` navigation.
- Optionally syncs with backend if user is authenticated.

---

5. Home Page

File: `src/pages/Home.tsx`

- Fetches area data and user summaries from `/areas` and `/home`.
- Displays streaks and study summary tiles.
- Provides navigation to CreateSession, Profile, and Info pages.

---

6. Profile Page

File: `src/pages/Profile.tsx`

- Tabs: **Overview**, **Personal Info**, **Data & Privacy**, **All Sessions**.
- Profile photo uploads:
    - POST `/profile/photo/uploadurl`
    - PUT file to S3 using pre-signed URL
    - PUT `/profile` with public photo URL
- Supports in-app reset and password change via Cognito IDP.

---

7. Utilities and Components

Notable utilities:

- `api/index.ts` – fetch wrapper adding `Authorization: Bearer id_token` header.
- `local/data.ts` – handles offline session data and streak computation.
- `utils/VolumeControl.tsx` – stylized volume slider and animations.

Core UI components:

- **TimerControls** – play/pause/reset/fullscreen, contextual actions.
- **FeedbackModal** – collects and sends user messages.
- **BottomCredits** – adaptable overlay or inline credits with safe-area awareness.

Backend

Overview

The backend consists of **Node.js AWS Lambda** functions exposed via **HTTP API Gateway**. Each Lambda handles a specific route, with shared logic abstracted in `services/lib`.

---

1. API Endpoints

Lambda	Route	Auth	Purpose
`listAreas.ts`	GET `/areas`	Public	Returns available study areas.
`getHome.ts`	GET `/home`	Protected	Fetches user streaks and recent sessions.
`createSession.ts`	POST `/sessions`	Protected	Creates new study session records.
`appendEvent.ts`	POST `/sessions/{id}/events`	Protected	Appends session event (focus lost, completion).
`getProfile.ts`	GET `/profile`	Protected	Retrieves user profile info.
`putProfile.ts`	PUT `/profile`	Protected	Updates user profile record.
`getUploadUrl.ts`	POST `/profile/photo/uploadurl`	Protected	Generates pre-signed S3 upload URL.
`feedback.ts`	POST `/feedback`	Public	Sends user feedback via Resend API.

---

2. Data Model

DynamoDB single-table structure:

PK	SK	Entity	Attributes

`USER#<sub>`	`PROFILE`	Profile	`{ name, photoUrl, updatedAt }`
`USER#<sub>`	`SESSION#<id>`	Session	`{ plan, createdAt, completedAt }`
`USER#<sub>`	`EVENT#<id>`	Event	`{ type, timestamp, duration }`

- Uses `PAY_PER_REQUEST` billing.
- Simple key-based access patterns.

---

3. Auth & Security

- Protected routes require a valid **JWT token** verified by the API Gateway's **Cognito authorizer**.
- Public routes (`/areas`, `/feedback`) explicitly omit auth.
- All endpoints return **CORS-friendly responses** for browser access.

---

4. Runtime Configuration

Each Lambda is configured with:

- `TABLE_NAME` – DynamoDB table reference
- `PROFILE_PHOTO_BUCKET` – S3 bucket name
- `RESEND_API_KEY`, `CONTACT_FROM_EMAIL`, `CONTACT_TO_EMAIL` – feedback integration

Subtab – Infrastructure (CDK)

Overview

Infrastructure is managed via **AWS CDK (TypeScript)**, defined under `infra/lib/echoes-infra-stack.ts`.
The CDK stack provisions all core services declaratively, ensuring reproducible environments.

---

1. Resources

Resource	Description
DynamoDB Table	Single table (`PK`, `SK`), PAY_PER_REQUEST, dev removal policy.
Cognito User Pool & Client	Handles user registration, login, password resets.
API Gateway (HTTP)	Connects to backend Lambdas; CORS enabled.
S3 Bucket (ProfilePhotos)	Public photo hosting; pre-signed PUT access.
Lambda Functions	Deployed from `/services/api`, Node.js 20 runtime.

---

2. Environment Variables

Each Lambda automatically receives environment variables for:

```
TABLE_NAME
RESEND_API_KEY
CONTACT_FROM_EMAIL
CONTACT_TO_EMAIL
PROFILE_PHOTO_BUCKET
```

---

3. Outputs

CDK exports:

`ApiUrl`
`UserPoolId`
`UserPoolClientId`
`UserPoolDomain`
`Region`

---

4. Deployment Behavior

- CDK synthesizes (`cdk synth`) → deploys (`cdk deploy`) the entire stack.
- Frontend uses Amplify for separate hosting and build.
- Environments are isolated: `dev`, `prod`, etc.
- IAM roles are auto-generated with least privilege.

Runtime and Behavior

1. Frontend Initialization

- App loads MSW mocks in dev mode.
- AuthProvider hydrates Cognito tokens.
- BrowserRouter loads initial route.

2. Session Lifecycle

1. User creates a session (`CreateSession`).
2. Navigates to `/study` → session begins.
3. Events (focus lost, completion) appended locally and optionally to backend.
4. On completion, app plays success audio and updates streaks.

3. Profile Management

- Upload photo via pre-signed S3 URL → save URL → instantly visible.
- Change password via Cognito IDP APIs.
- Reset app data locally if needed.

4. Feedback Flow

- Modal POSTs to `/feedback`.
- Lambda validates input and relays via Resend.
- Returns `{ ok: true }` with CORS headers.
- Frontend shows success banner.

5. Error Handling & Offline Mode

- Local persistence in `local/data.ts` ensures no session data loss.
- Missing auth → local fallback mode.
- Network errors → user toasts and cached recovery.