# 1 — High level technical architecture (overview)

**Goal:** A mobile-first React Native app with an Express/Node.js API, mobile offline-first logging & cloud sync, user profiles/goals/social features, a natural-language chatbot, and analytics.

**Primary components**

- **React Native app (mobile):** UX, offline persistence, habit logging, chatbot UI, motivation hub, social sharing, onboarding quiz. (4 frontend components divided among 4 members)
- **Backend API (Express/Node.js):** auth, business rules, data validation, sync endpoints, chatbot orchestration, notifications, shareable image/card generation.
- **Database:** Primary relational DB (Postgres) for core data; Redis for caching/session/locks; S3 (or compatible object store) for photos and generated share cards.
- **Background workers / async services:** queue (BullMQ with Redis) for heavy tasks: image rendering, notification scheduling, badge computation, chatbot model calls.
- **Third-party services:** Push notifications (FCM/APNS via a push service), email, SMS provider (optional), external ML/LLM for chatbot (if any), analytics (Mixpanel/Amplitude), and Sentry for errors.
- **CI/CD:** GitHub Actions for tests and deploys; staging & production environments; mobile builds via EAS (Expo) or Fastlane.
- **Hosting:** Backend containerized (Docker) on serverless or Kubernetes, or a managed container host (e.g., AWS ECS/EKS, DigitalOcean App Platform). Postgres managed (RDS / DigitalOcean Managed DB).

**Data flow (user habit logging example):**

1. User adds habit in app → stored locally in SQLite/AsyncStorage.
2. App tries to send `/sync/habits` to backend; server validates & persists to Postgres.
3. Server enqueues badge recalculation + notification via Redis/Bull.
4. Worker updates user badges and sends push notification.

---

# 2 — Responsibilities & team split (5 members)

You said you'll do the full backend. The 4 frontend members each take a component:

1. **You (Backend — Full stack backend owner)**

    - Build Express API, DB schemas, workers, authentication, sync logic, admin endpoints, deployments.
2. **Frontend Member A — Onboarding & Profile + Motivation Hub**
    - Lifestyle quiz, baseline profile, profile editing, Motivation feed, weekly personalized suggestions. (Onboarding & motivation features).

      KeyFeatures&UserFlow

3. **Frontend Member B — Daily Habit Tracking**
    - Habit list, quick-add UI, voice-to-text integration, photo evidence upload, history & trending graphs, offline-first features, local persistence & sync.

      KeyFeatures&UserFlow

4. **Frontend Member C — Chatbot**
    - Chat UI, message history, NLU integrations, fallback flows for logging activities via conversation. Integrates with backend chatbot endpoints.

      KeyFeatures&UserFlow

5. **Frontend Member D — Social, Goals & Sharing**
    - Goals UI, badges & streaks, private leaderboards, progress reports, shareable card generator and share flows (social media).

      KeyFeatures&UserFlow

---

# 3 — Database design (recommended Postgres schema)

Key design notes:

- Use normalized relational model for users, habits, goals. Store time-series habit logs with good indexes for time-range queries.
- Use UUIDs as primary keys (pgcrypto or uuid-ossp).
- Store photo metadata (S3 keys) not binary in DB.
- Soft deletes via `deleted_at` timestamps.
- Add partitioning for `habit_logs` by date if scale grows.

## Tables (core)

**users**

- `id` UUID PK
- `email` text UNIQUE
- `phone` text NULL
- `password_hash` text (bcrypt)
- `name` text
- `timezone` text
- `locale` text
- `avatar_s3_key` text NULL
- `onboarding_completed` boolean default false
- `baseline_profile` jsonb (quiz result snapshot)
- `created_at`, `updated_at`

### sessions / refresh_tokens

- `id` UUID PK
- `user_id` FK -> users(id)
- `refresh_token_hash` text
- `device_info` jsonb (os, device id)
- `expires_at`, `created_at`

### habits (catalog / user-defined)

- `id` UUID PK
- `user_id` FK
- `title` text
- `category` text (enum: transport, food, energy, waste, consumption)
- `unit` text (km, kg, trips, meals, etc.)
- `co2_per_unit` numeric (kg)
- `is_default` boolean (pre-set suggestions)
- `meta` jsonb (voice-suggestions, prompts)
- `created_at`, `updated_at`

### habit_logs

- `id` UUID PK
- `user_id` FK
- `habit_id` FK
- `quantity` numeric
- `co2_kg` numeric (computed)
- `note` text
- `photo_s3_key` text NULL
- `logged_at` timestamptz (when action happened)
- `synced` boolean default true (for server-side sync tracking)
- `created_at`, `updated_at`

Indexes: `CREATE INDEX ON habit_logs (user_id, logged_at);` and consider BRIN or partitioning by `logged_at`.

### goals

- `id` UUID PK
- `user_id` FK
- `target_co2_weekly` numeric
- `target_co2_monthly` numeric
- `start_date`, `end_date`
- `created_at`, `updated_at`

### badges

- `id` UUID PK
- `user_id` FK
- `badge_type` text
- `earned_at` timestamptz
- `meta` jsonb

### social_groups (private leaderboards)

- `id`, `name`, `owner_user_id`, `join_code`, `is_private` boolean

### group_members

- `group_id` FK, `user_id` FK, `role`, `joined_at`

### notifications

- `id` UUID, `user_id`, `type`, `payload` jsonb, `sent_at`, `read_at`, `created_at`

### audit / event_log

- `id`, `user_id`, `event_type`, `payload` jsonb, `created_at` — for analytics & debugging.

### chatbot_messages

- `id`, `user_id`, `session_id`, `message_text`, `direction` enum(inbound/outbound), `intent` text, `response_meta` jsonb, `created_at`

### attachments

- `id`, `user_id`, `s3_key`, `type`, `created_at`

---

# 4 — API contract (high-level endpoints)

Use RESTful API (or GraphQL if preferred). I'll outline REST endpoints:

## Auth

- `POST /api/v1/auth/signup` — body: {email, password, name, timezone, locale}
- `POST /api/v1/auth/login` — returns access token (JWT) + refresh token
- `POST /api/v1/auth/refresh` — exchange refresh token
- `POST /api/v1/auth/logout` — revoke refresh token
- `POST /api/v1/auth/magic-link` (optional)

## Users & Profile

- `GET /api/v1/users/me`
- `PATCH /api/v1/users/me` — update profile, locale, baseline_profile
- `POST /api/v1/users/me/avatar` — upload photo -> returns S3 upload URL

## Habits

- `GET /api/v1/habits` — user habits + pre-set suggestions
- `POST /api/v1/habits` — create custom habit
- `GET /api/v1/habits/:id`
- `PATCH /api/v1/habits/:id`
- `DELETE /api/v1/habits/:id`

## Habit logs (important)

- `POST /api/v1/habit-logs` — single or batch log (for sync). Body supports array of logs (for offline sync).
- `GET /api/v1/habit-logs?from=&to=&category=&limit=&offset=`
- `GET /api/v1/habit-logs/:id`
- `PATCH /api/v1/habit-logs/:id` — edit entry
- `DELETE /api/v1/habit-logs/:id`

## Sync

- `POST /api/v1/sync` — client pushes local DB changes; server returns server-side changes and conflict resolution hints.

## Goals & Badges

- `GET /api/v1/goals`
- `POST /api/v1/goals`
- `GET /api/v1/badges`
- `GET /api/v1/leaderboards?group_id=&period=week|month`

## Chatbot

- `POST /api/v1/chatbot/message` — {message, session_id} → returns {reply, actions[]} where actions might be `create_habit_log` with structured params
- `GET /api/v1/chatbot/history?session_id=`

## Social & Sharing

- `POST /api/v1/share/card` — server generates a shareable image (returns S3 link)
- `POST /api/v1/groups` — create private leaderboards
- `POST /api/v1/groups/:id/join` — join by code

## Admin/Analytics (internal)

- `GET /api/v1/admin/metrics` — aggregated metrics
- `POST /api/v1/admin/seed-default-habits`

**Security:** All protected endpoints require Bearer JWT access tokens; refresh tokens stored hashed in DB.

---

# 5 — Non-functional & infra recommendations

- **Auth:** JWT access tokens (short lived, e.g., 15–60 min) + refresh tokens. Store refresh token hash server-side to allow revocation.
- **Offline-first:** Local SQLite (react-native-sqlite-storage or WatermelonDB) for habit_logs + background sync endpoint `/sync`. Use optimistic conflict resolution: server compares `client_updated_at` and `server_updated_at`.
- **Photo uploads:** Use pre-signed S3 upload URLs generated by backend (`POST /attachments/presign`) and store `s3_key` in DB.
- **Rate limiting & security:** API rate limiting (IP and user), helmet, CORS, input validation (Joi / zod).
- **Caching:** Redis for leaderboard caching, computed weekly reports, badge statuses.

- **Workers:** BullMQ (Redis) for background tasks: badge recompute, push notification scheduling, image generation for share cards, analytics ETL.
- **Monitoring & logging:** Sentry for errors, structured logs to centralized system (ELK / Datadog). GDPR/Privacy: minimize PII storage, retention policy.
- **Backups:** Daily DB backups; retention policy 30–90 days; snapshots for S3.
- **Testing:** Unit tests (Jest), integration tests (supertest), E2E for RN with Detox or Appium.

---

# 6 — Backend folder structure (Express / Node.js)

Use TypeScript for safety.

```
/backend
├─ /src
│  ├─ /api
│  │  ├─ /v1
│  │  │  ├─ auth.controller.ts
│  │  │  ├─ users.controller.ts
│  │  │  ├─ habits.controller.ts
│  │  │  └─ chatbot.controller.ts
│  ├─ /services
│  │  ├─ auth.service.ts
│  │  ├─ habit.service.ts
│  │  ├─ chatbot.service.ts
│  ├─ /workers
│  │  ├─ badge.worker.ts
│  │  └─ shareCard.worker.ts
│  ├─ /db
│  │  ├─ index.ts (knex/TypeORM/Prisma client)
│  │  └─ migrations/
│  ├─ /models (if using ORM)
│  ├─ /middleware
│  ├─ /utils
│  ├─ /config
│  ├─ app.ts
│  └─ server.ts
├─ /scripts
├─ /migrations
├─ /docker
├─ Dockerfile
├─ docker-compose.yml
├─ package.json
└─ tsconfig.json
```

**Notes:** I recommend using Prisma for Postgres schema migrations and typesafety.

---

# 7 — Frontend folder structure (React Native — TypeScript / Expo or bare RN)

```
/mobile
├─ /src
│   ├─ /components        // small reusable components (Button, Card, Avatar)
│   ├─ /screens
│   │   ├─ OnboardingScreen/
│   │   ├─ HomeScreen/
│   │   ├─ HabitScreen/
│   │   ├─ ChatbotScreen/
│   │   ├─ GoalsScreen/
│   │   └─ ProfileScreen/
│   ├─ /services          // API wrappers, auth client
│   ├─ /hooks             // custom hooks (useAuth, useSync, useHabits)
│   ├─ /store             // redux or Zustand store
│   ├─ /navigation        // react-navigation stacks
│   ├─ /lib               // utilities (date handling, co2 calc)
│   ├─ /assets
│   ├─ /i18n
│   ├─ /styles
│   └─ App.tsx
├─ app.json (or app.config.js)
├─ package.json
└─ tsconfig.json
```

**Offline storage:** Implement a SyncManager that tracks local changes (queue) and posts `POST /api/v1/sync` periodically / on network.

---

# 8 — Example API payloads & sample validations

**POST /api/v1/habit-logs** (batch)

```
{
  "logs": [
    {
      "client_id": "local-uuid-123",   // client's temp id for dedupe
      "habit_id": "uuid-abc",
      "quantity": 2.5,
      "logged_at": "2025-10-05T12:00:00Z",
      "note": "Took bus",
      "photo_s3_key": "uploads/abc.jpg",
      "client_updated_at": "2025-10-05T12:05:00Z"
    }
  ]
}
```

Response:

```
{
  "synced": [
    {
      "client_id": "local-uuid-123",
      "server_id": "uuid-987",
      "conflict": false
    }
  ],
  "server_changes": { "habit_logs": [] }
}
```

---

# 9 — GitHub issues & project management (epics + tasks)

I'll create three milestones: **MVP (v0.1)**, **Polish (v0.2)**, **Launch (v1.0)**. Below are Epics and concrete issues ready to create in GitHub. Each issue should have labels: `backend/frontend`, `priority`, `component`, `estimate` (points or hours).

## Epic: MVP — Core backend (assigned to you)

- **Issue:** `BE: Setup project skeleton & CI` — TypeScript + ESLint + Prettier + GH Actions. [backend, high, 8h]
- **Issue:** `BE: Auth (signup/login + Clerk)` [backend, high, 16h]
- **Issue:** `BE: Users endpoints + avatar presign` — profile GET/PATCH, presigned upload URL. [backend, medium, 8h]
- **Issue:** `BE: Habits CRUD + default suggestions seeding` — migrations, seed script. [backend, high, 16h]
- **Issue:** `BE: Habit-logs endpoints (batch + query)` — ensure atomic inserts and validation. [backend, high, 20h]
- **Issue:** `BE: Sync endpoint (/sync) & conflict strategy` — implement server diff & merge. [backend, high, 20h]
- **Issue:** `BE: Attachments (S3 presign) + photo storage policy` [backend, medium, 6h]
- **Issue:** `BE: Chatbot endpoint (basic) — route to LLM/mock` — return structured actions. [backend, medium, 12h]
- **Issue:** `BE: Setup Redis + BullMQ + worker skeleton` — for queue operations. [backend, medium, 8h]

## Epic: MVP — Frontend components (assign each to one of 4 frontend members)

**Frontend Member A — Onboarding & Profile**

- `FE: Onboarding quiz screens & API integration` [frontend, high, 16h]
- `FE: Profile & baseline profile UI` [frontend, medium, 8h]
- `FE: Motivation hub placeholder + feed` [frontend, medium, 12h]

**Frontend Member B — Daily Habit Tracking**

- `FE: Habit list, quick-add, default suggestions` [frontend, high, 20h]
- `FE: Quick-add voice-to-text integration (native module)` [frontend, medium, 12h]
- `FE: Photo attach & local storage` [frontend, medium, 8h]
- `FE: Local DB + offline queue` [frontend, high, 20h]

**Frontend Member C — Chatbot**

- `FE: Chat UI & message flow` [frontend, high, 16h]
- `FE: Chatbot integration with /api/v1/chatbot/message` [frontend, medium, 12h]
- `FE: Action handling (create habit log from chat)` [frontend, medium, 8h]

**Frontend Member D — Social, Goals & Sharing**

- `FE: Goals UI + progress visualization` [frontend, high, 16h]
- `FE: Badges & streaks view` [frontend, medium, 12h]
- `FE: Shareable card generation flow (request server image)` [frontend, medium, 12h]
- `FE: Private group / leaderboard UI` [frontend, medium, 12h]

# Epic: Ops / infra / polish

- `Infra: Staging & production deploy pipelines` [ops, high, 12h]
- `Infra: DB backups & restore playbook` [ops, medium, 8h]
- `Polish: Add analytics events for key flows` [backend+frontend, medium, 10h]
- `Polish: Accessibility audit + fixes (screen reader, alt text)` [frontend, medium, 8h]
- `QA: E2E test habit flow` [qa, high, 12h]
- `Security: Pen test checklist & implement rate limiting` [security, medium, 8h]

---

# 10 — Developer handover checklist (what you should push to repo & CI)

- README with architecture overview + local dev instructions.
- `.env.example` (no secrets).
- DB migration scripts + seed for default habits.
- Postman / OpenAPI spec for APIs or a Swagger file.
- Scripts to create S3 buckets + IAM roles (or docs).
- GH Projects board with issues linked to epics & assignees.
- PR template with checklist (tests, lint, security).

---

# 11 — Helpful implementation tips & quick patterns

- **$CO_2$ calculation logic:** keep canonical formula server-side and store `co2_per_unit` on habit. Also compute `co2_kg` at ingestion so historical changes don't retroactively change past entries.
- **Optimistic UI for logging:** show entry in UI before sync, mark pending; reconcile on server response.
- **Chatbot actions:** return `action` objects from chatbot endpoint that contain `type` (e.g., `create_habit_log`) and canonical fields to avoid parsing on client.
- **Badge recalculation:** run worker on `habit_logs` insert events; update `badges` table and push to user.
- **Rate limit for chatbot:** throttle per-user LLM calls to control costs.
- **Data retention & export:** create `GET /api/v1/users/me/export` to allow export of personal data in JSON/CSV for privacy compliance.