Complete Developer Guide

Monchitro project documentation | English PDF edition. The web guide is available in Bangla and English.

1. Project overview

Monchitro is a Bangla-first, bilingual study companion for learners in Bangladesh. The product connects curriculum-shaped digital books with reading, private annotations, guided practice, audio, and age-aware study spaces.

The current working pilot is an original Class 10 Physics chapter. It is demonstration content, not a reproduction or endorsement of an official textbook. Subject editors must verify curriculum alignment and rights before public release.

- Primary users: students in Classes 6-12; guardians, teachers, editors, and administrators.
- Curriculum scope: NCTB Bangla Medium first, with English Version support.
- Product principle: explain and coach without replacing the learner's own thinking.

Demo accounts, activity, progress, and collaboration are sample data unless a connected service is explicitly identified.

2. Technology stack

Web | Next.js App Router 16, React 19, TypeScript | Routes, server rendering, typed UI

Design | Tailwind CSS 4, shadcn/ui, Lucide | Accessible interface primitives and tokens

Reading | react-pageflip / StPageFlip | Real page-turn interactions with selectable HTML

Data and identity | Supabase Auth and Postgres (integration connected; schema rollout is tracked separately) | Accounts, learning records, and policy-protected data

AI | Vercel AI Gateway (connected); server-side AI SDK route planned | Context-aware coaching without exposing credentials

Offline and charts | PWA service worker, IndexedDB, Recharts | Explicitly saved demo chapters and investor assumptions

Future collaboration | Yjs provider and object storage adapter | Enable only after an authenticated provider is configured

3. Frontend project structure

The active repository is a compact App Router demo. The tree below separates current entry points from recommended modules for the next production phases.

- Keep route files thin; place reusable workflows in components and domain data in lib.
- Do not expose server-only keys or privileged database clients to browser bundles.

Repository map

```
  app/
    app/                        # student overview
    onboarding/                 # profile setup demo
    library/                    # searchable sample catalog
    reader/[book]/              # flipbook reader
    practice/[chapter]/         # chapter quiz
    guardian/                   # consent-aware overview demo
    study-room/[room]/          # moderated local demo
    investors/                  # investor section
    documentation/              # developer docs and TOR
  components/monchitro/
    chrome.tsx                  # public and student navigation
    reader.tsx                  # reader tools and local annotations
    reader-flipbook.tsx         # dynamic HTMLFlipBook wrapper
    practice.tsx                # quiz and result state
    student.tsx                 # dashboard, library and guardian views
  lib/
    monchitro-content.ts        # original bilingual sample content
    monchitro-docs.ts           # bilingual documentation data
    supabase/                   # server, browser and session clients (next phase)
  public/
    manifest.webmanifest
    monchitro-sw.js
  supabase/migrations/           # managed schema changes (MCP-applied)
```

4. Flipbook development specification

The reader is dynamically imported with server rendering disabled because StPageFlip depends on browser layout APIs. Page components use React.forwardRef and the PageFlip API-not a CSS page-turn imitation.

- Use landscape two-page mode on desktop and portrait single-page mode on narrow screens.

- Wait for the expected page count before enabling controls; the ref can exist before the HTML pages are initialized.

- Memoize page children so a parent progress update does not replace page DOM during an active turn.

- Keep selectable text; reserve the page margins/corners for drag gestures and provide keyboard/page controls.

[ ] Verify real mouse drag, touch swipe, keyboard arrows, and accessible previous/next buttons.

[ ] Check cover/back-cover hardness and two-page versus portrait layouts at the stated breakpoints.

Dynamic reader and hard/soft pages

```
const Flipbook = dynamic(() => import("./reader-flipbook"), { ssr: false })

const InnerPage = forwardRef<HTMLDivElement, PageProps>(function InnerPage(
  { page }, ref,
) {
  return <div ref={ref} className="reader-page" data-density="soft">
    <article>{page.title}<p>{page.body}</p></article>
  </div>
})

const CoverPage = forwardRef<HTMLDivElement, CoverProps>(function CoverPage(
  { title }, ref,
) {
  return <div ref={ref} className="reader-cover" data-density="hard">
    <h1>{title}</h1>
  </div>
})
```

Responsive settings and ref methods

```
<HTMLFlipBook
  ref={bookRef}
  width={440}
  height={620}
  size="stretch"
  minWidth={280}
  maxWidth={880}
  minHeight={390}
  maxHeight={620}
  showCover
  usePortrait={isMobile}
  mobileScrollSupport
  showPageCorners
  onFlip={(event) => setPageIndex(event.data)}
>
  {pageElements}
</HTMLFlipBook>

bookRef.current?.pageFlip().flipNext()
bookRef.current?.pageFlip().flipPrev()
bookRef.current?.pageFlip().turnToPage(index)
```

5. Textbook content engine

A book is versioned curriculum content: book metadata -> chapter order -> page records -> selectable text nodes and licensed assets. The reader currently consumes typed, bundled sample content; production publishing should validate every revision and preserve an audit trail.

- Store curriculum, class, medium/version, subject, chapter, page number, locale, revision, and rights/source metadata.

- Keep diagrams as SVG or optimized raster assets with text alternatives; never flatten lesson text into an image.

- Attach text-to-speech offsets to authored paragraphs, not to screen coordinates.

- Cache a chapter only after an explicit download action and show its storage/version state.

6. Collaborative annotation architecture

Highlights, private notes, public comments, and shared-room cursors are different data classes. The current reader stores annotations in local component state; a production adapter must persist private records per user and authorize shared content through room membership.

- A Y.Doc can hold a room's ordered comments and shared highlights; private notes belong outside that shared document.
- Use a server-authorized room token, membership checks, rate limits, and moderation before enabling a WebSocket provider.
- Persist periodic snapshots and define retention, replay, and deletion behavior before calling synchronization durable.

> Private note -> authenticated user -> RLS-protected annotation row

> Shared comment -> study room -> approved member -> moderation/report flow

> Yjs provider -> authenticated room token -> document updates -> persistence snapshot

The current study circle is local demo state; it is not live multi-user Yjs synchronization.

7. AI tutor integration

The tutor should scaffold a student's reasoning: clarify the question, surface known quantities, offer one hint at a time, and link back to a reviewed source page. Never silently submit a generated explanation as verified curriculum content.

- Call the Vercel AI Gateway from a server route; keep model identifiers and prompts on the server.
- Pass only the selected text and the minimum chapter context needed; avoid student profile PII.
- Apply request size limits, per-user/IP rate limits, output moderation, and a report-answer path.
- When AI is unavailable, label the sample hint as demo content rather than implying a live model response.

Vercel AI Gateway is connected to this project. A live tutor endpoint must still be implemented and verified before production claims.

8. Assessment engine

- Version question banks and store question prompt, options, correct answer, explanation, marks, topic, difficulty, locale, and publication status.
- Calculate the score on the server for authenticated attempts; do not accept a client-supplied score as authoritative.
- Persist an attempt once with a unique attempt ID, answer map, timing, score, and question version.
- Show a review with each selected answer, correct answer, and an age-appropriate explanation.

The current Class 10 quiz is a local demo and scores only the bundled sample questions.

9. Database design

Use UUID primary keys, explicit foreign keys, timestamps, and narrow indexes for common access paths. Every table containing student or guardian data must enable RLS. The live Supabase project was empty when this guide was prepared; apply and review migrations before using these entities in runtime code.

profiles | id -> auth.users.id | Self read/update; no client role escalation

student_profiles | user_id -> profiles.id | Student self; guardian only via approved link

books / chapters / pages | book -> chapter -> page | Published content readable; writes editor-only

annotations / bookmarks | user_id + page_id | Owner-only by default

quiz_attempts / learning_progress | student_id + quiz/book version | Owner read; validated server writes

guardian_links | guardian_id + student_id | Approved relationship only

RLS ownership pattern

```
alter table public.annotations enable row level security;
```

```
create policy annotations_select_own
on public.annotations for select
using (auth.uid() = user_id);

create policy annotations_insert_own
on public.annotations for insert
with check (auth.uid() = user_id);
```

Never create a self-service admin role policy. Admin access must be granted through trusted server-side administration and audited separately.

10. API documentation

The routes below define the intended contract. They are a specification, not a claim that every endpoint is implemented or deployed.

POST | /api/auth/register | Create an account through Supabase Auth; rate-limited

GET | /api/books | List published books with class/medium filters; public read

GET | /api/books/:bookId | Get a published book and authorized chapters

GET | /api/chapters/:chapterId | Get chapter metadata and page order

GET | /api/pages/:pageId | Get selectable, localized page content

POST | /api/annotations | Create an owner-scoped note/highlight; validate page ownership

GET | /api/annotations | List the signed-in learner's annotations

POST | /api/ai/explain | Server-side, rate-limited tutoring; minimal context

GET | /api/questions/:chapterId | Return active quiz questions without answer keys

POST | /api/quiz/submit | Score an authenticated attempt on the server

GET | /api/student/progress | Return the current student's progress only

POST | /api/subscriptions | Create a pending subscription intent; no fake paid state

Example request

```
POST /api/annotations
Authorization: Bearer <Supabase access token>
Content-Type: application/json

{
  "pageId": "uuid",
  "kind": "highlight",
  "selectedText": "A net force changes motion.",
  "color": "sunshine"
}
```

Status: planned API contract. Return generic client-safe errors; log diagnostics server-side without logging passwords or access tokens.

11. PWA and offline engineering

- Serve a valid web manifest with app name, standalone display, theme color, and real icon assets.

- Cache the app shell and explicitly downloaded chapter data; do not cache private API responses in a public cache.

- Use IndexedDB with a schema version and size-aware deletion; show download progress and offline availability.

- Queue only retry-safe progress writes and reconcile conflicts when the device reconnects.

- On iOS, explain Safari Share -> Add to Home Screen; do not claim an App Store listing.

12. Security and privacy

- Keep Supabase service-role credentials and AI credentials on the server only.

- Enable RLS and scope every private read/write to auth.uid(); enforce guardian links at the parent relationship.

- Collect the minimum age/consent data needed; do not expose student contact details, school identifiers, or precise location.

- Private notes remain private by default; guardians receive only consented summaries.

- Use server-side payment verification and idempotency before marking a subscription paid.

- Add moderation, report, block, mute, retention, export, and deletion procedures before opening peer spaces.

13. Deployment and DevOps

- Use a reviewed Git branch and pull request; deploy previews before production.
- Configure only required environment variables in Vercel; never copy secrets into source or documentation.
- Apply database migrations through the approved Supabase migration workflow and verify RLS before release.
- Monitor error rates, route latency, AI usage, storage, and failed background work.
- Back up data, test restoration, and define incident, rollback, and security-contact procedures.

[ ] Preview smoke test in Bangla and English

[ ] RLS and role-boundary checks

[ ] Mobile reader, PWA, and offline checks

[ ] Content rights and curriculum review

[ ] Rollback and support owner confirmed