TAKE-HOME ASSIGNMENT — Internship Edition

Time budget: ≈ 4–5 h of focused work
Submission: GitHub repo + live/demo link within 72 h of receiving this brief

Pick ONE of the two mini-challenges below (Web or Mobile).

Both re-introduce a very small e-mail / password auth flow and use a completely FREE "auto-summary" API—no credit card or API key needed.

A. Full-Stack Mini-Challenge: "Link Saver + Auto-Summary"

Must-have user stories

Sign-up / Login

- Basic e-mail + password form
- Hash the password (bcrypt, Argon 2, etc.) and store locally (SQLite / Mongo / Supabase / IndexedDB)
- Issue a JWT or session cookie on login
- *Hint: External authentication libraries like Firebase Authentication or Supabase Auth can also be used to simplify implementation*

Save a bookmark

- Paste any URL → fetch and store title & favicon (hint: document.title, /favicon.ico, OpenGraph tags, etc.)

Generate summary (no paid key)

- Call Jina AI open endpoint (see Appendix A)
- Save the returned text alongside the bookmark

List & view

- Responsive list/grid of saved links with title, favicon and summary text
- Delete bookmark button

Nice-to-have (optional, not graded harshly)

- Tag filter
- Drag-drop re-order

- Dark mode

Tech guard-rails (pick your comfort)

- Front-end: React, Next.js, SvelteKit…
- Back-end: Node/Express, Next API routes, FastAPI…
- DB: SQLite, MongoDB, Supabase or even localStorage for demo

B. Mobile Mini-Challenge: "Three-Screen Workout Tracker"

Must-have user stories

- Auth screen – e-mail/password saved securely on-device (SecureStore/MMKV/KSec)
- *Hint: External authentication libraries like Firebase Authentication can also be used to simplify implementation*
- Home – list 3 hard-coded workouts
- Detail – show exercises; "Start" runs a simple timer that auto-advances
- History – log completed workouts locally and render a weekly calendar or list

Nice-to-have (optional)

- Dark mode
- Voice TTS cues
- Export to Google Fit / Apple Health

Tech guard-rails

- React Native (Expo) --OR-- Flutter
- Local persistence: SQLite, Hive, MMKV, etc.
- Remote sync is optional

SUBMISSION CHECKLIST

☐ Public (or private-invite) GitHub repo

☐ Live link (Vercel / Netlify / Render / Expo Snack / Flutter Web / TestFlight / APK)

☐ Clear README: tech stack, setup, "what I'd do next," and how long you spent

☐ 2–5 screenshots or a short GIF

☐ At least a couple of unit / component / widget tests

☐ Small, logical commit history (no single "big bang" commit)

EVALUATION RUBRIC

- Core functionality met: 40%

- Code clarity & structure: 30%
- UI / UX polish: 20%
- Git & README hygiene: 10%

We value clean, readable code and thoughtful UX over sheer quantity of features.

Note on UI/UX: No Figma designs are provided intentionally. We're evaluating your UI/UX intuition and decision-making. Focus on creating a simple, intuitive, and visually appealing interface that demonstrates your understanding of good design principles. You can use any UI component libraries or design systems of your choice.

If you get stuck

Explain the road-block in a short comment or e-mail and push what you have—we're as interested in your reasoning as in the finished product.

For any questions or assistance, please contact: **gautam@omvad.com**

Good luck, and have fun building! □

Appendix A – Quick-Start Pointers for the FREE Jina AI "r.jina.ai" Summarisation API

Purpose

Public endpoint that returns an extractive summary of any reachable URL. No sign-up, no key, no cost.

Call shape

```
GET https://r.jina.ai/http://<URL-ENCODED_TARGET_PAGE>
```

Example:

```
const target = encodeURIComponent(
  'https://en.wikipedia.org/wiki/Artificial_intelligence'
);
const res = await fetch(`https://r.jina.ai/http://${target}`);
const summary = await res.text();   // plain text, not JSON
```

Things for you to decide (intentionally open)

- When/how to URL-encode
- Where to trim the summary (it can be long)
- How to persist & display multi-line text

CORS & limits

- CORS enabled → works directly from browsers
- Informal limit ≈ 60 calls/hour per IP—handle errors gracefully

Fallback idea (if the API is ever down)

```
try {
  return await getSummary(realUrl);
} catch {
  return 'Summary temporarily unavailable.';
}
```

Happy coding!