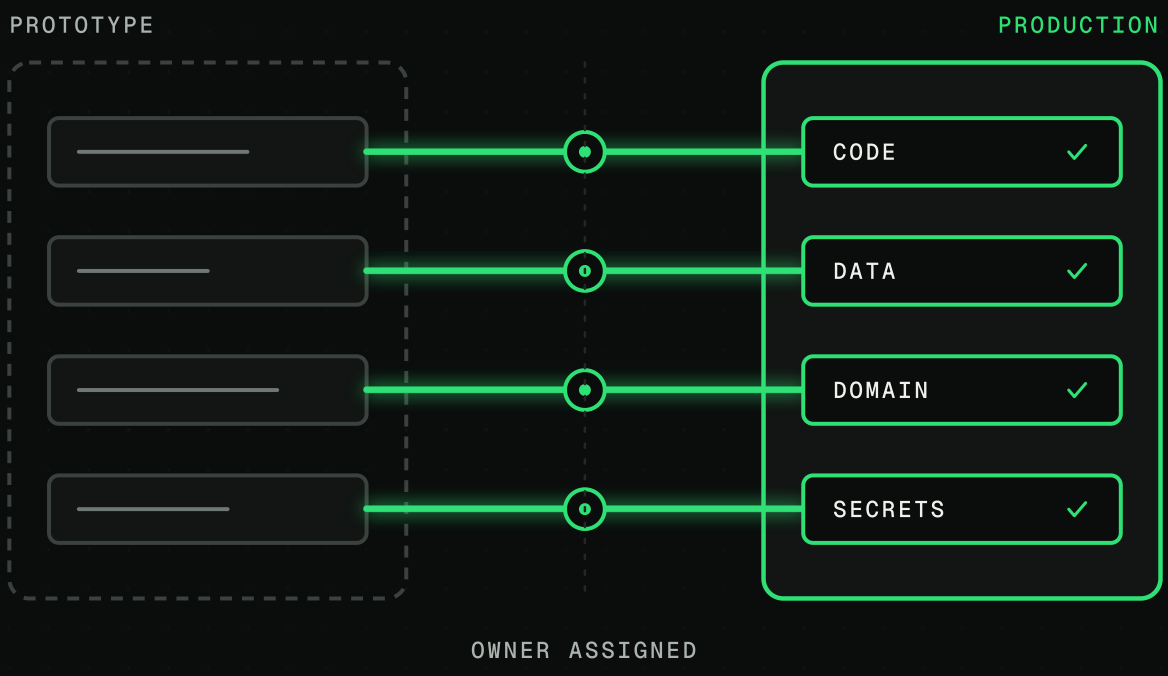


CHECKLIST · FREE

Lovable to Production Handoff Checklist

A 42-check handoff for running a Lovable app as a real product: code, access rules, secrets, domain, backups and change process.

HOW TO USE THIS CHECKLIST

Take a Lovable build to production with the code, backend, domain and secrets under named owners and a tested way to recover.

Forty-two verifiable checks, a secrets and environment inventory, a go-live runbook, and a copy-ready HANDOFF.md for turning an app built in Lovable into a product a named team owns and can keep running.

WORKING METHOD

01 Start by answering one question in writing: does the app run on Lovable's built-in backend or on a Supabase project you connected yourself? Several checks branch on that answer.

02 Work through the four checklists in order. Each check names the evidence that proves it, so paste that evidence, or a link to it, into the handoff record.

03 Fill in the secrets and environment inventory while you do the secrets checks. A row with an empty owner or storage cell is an open item, not a finished one.

04 Run the go-live runbook only after every check is passed or accepted in writing by a named person.

05 Finish by completing HANDOFF.md and committing it to the repository, so the next person does not depend on your chat history.

INSIDE

- ✓ 42 handoff checks
- ✓ Secrets and environment inventory
- ✓ Go-live runbook
- ✓ Ownership handoff template

BEFORE YOU START

Platform screens, plan features, and limits change. Every product statement here was checked against official documentation on the date in the research notes; recheck anything that affects a decision.

Lovable states that its security scans cannot guarantee complete security and do not replace a thorough review. Passing them is a starting point, not a sign-off.

Backup coverage depends on the backend and the plan. Confirm what applies to your project instead of assuming a backup exists.

This checklist does not cover legal or regulatory duties such as privacy notices, data residency, or payment compliance. Add the checks your market requires.

CONTENTS

01	Code and account ownership	8 CHECKS
02	Database and sign-in	12 CHECKS
03	Secrets, domain and integrations	11 CHECKS
04	Secrets and environment inventory	TABLE
05	Operations after handoff	11 CHECKS
06	Go-live runbook	RUNBOOK
07	Ownership handoff template	TEMPLATE

01 / 07 CHECKLIST

Code and account ownership

Prove the business holds the code and every account around it.

GitHub sync

6 CHECKS

- [ ] 01 Confirm the repository belongs to the business: open it on GitHub and record the owning account or organization; expect a company-controlled owner, not a personal one.
- [ ] 02 Prove two-way sync works: make one small edit in Lovable and one commit on the synced branch in GitHub; expect each change to appear on the other side.
- [ ] 03 Record the branch Lovable syncs: check the project's GitHub settings and write the branch name in HANDOFF.md, since Lovable edits and syncs one branch at a time.
- [ ] 04 Build the app from a clean clone: clone the repository on another machine, run the documented install and build commands, and expect success with no unwritten steps.
- [ ] 05 Check repository visibility and history for secrets: confirm the repository is private unless you chose otherwise, and search every commit for secret keys; expect only publishable values.
- [ ] 06 Write down the rule never to delete the synced repository: Lovable documents that deletion stops sync and that reconnecting creates a new repository.

Accounts and access

2 CHECKS

- [ ] 07 Name two owners for every account: confirm two people hold owner or admin access to the Lovable workspace, GitHub organization, and backend; remove anyone who has left.
- [ ] 08 Move logins off personal email: confirm billing, the domain registrar, and the backend account use a company-controlled mailbox with two-factor authentication turned on.

02 / 07 CHECKLIST

Database and sign-in

Test who can read and write each row, and whether real users can sign in on the live domain.

Backend ownership

2 CHECKS

- [ ] 09 Identify which backend the app runs on: check the project settings for Lovable's built-in backend or a connected Supabase project, and record the answer in HANDOFF.md.
- [ ] 10 Decide before launch whether to stay on that backend: Lovable documents no automatic migration in either direction, so record the decision, its owner, and who pays for usage.

Row level security

6 CHECKS

- [ ] 11 Confirm row level security is on for every table: list all tables in exposed schemas with their RLS status and expect no table without it.
- [ ] 12 Read every policy in plain language: write one sentence per policy saying who can do what, and flag any always-true condition for a named person to fix.
- [ ] 13 Test cross-account access with two test users: sign in as the first and request the second user's rows by ID; expect an empty result or an error.
- [ ] 14 Test signed-out access: query each table using only the publishable key and no session; expect no private rows returned and no writes accepted.
- [ ] 15 Test write rules separately from read rules: as a test user, try to change a row's owner column to another user's ID; expect the update to be rejected.
- [ ] 16 Review views and file storage: confirm each view applies the policies of its underlying tables and each storage bucket's access rule matches the files it holds.

Authentication

4 CHECKS

- [ ] 17 Set the production site URL and exact redirect URLs: complete sign-up, sign-in, and password reset on the live domain and expect every link to land on that domain.
- [ ] 18 Confirm how sign-in emails are sent and their limits: register with an address outside the team and expect delivery; on your own Supabase project, configure custom SMTP first.
- [ ] 19 Add the production domain to each sign-in provider: sign in with every enabled provider on the live domain and expect a completed session, not a redirect error.
- [ ] 20 Store roles where users cannot edit them: as a normal test user, try to change your own role or admin flag; expect the request to be refused.

03 / 07

CHECKLIST

Secrets, domain and integrations

Know where every key lives, who controls the domain, and that money and email flow in production mode.

Secrets and keys

4 CHECKS

- [ ] 21 Complete the secrets and environment inventory: list every key the app uses and confirm each row has a storage location, exposure level, owner, and rotation date.

- [ ] 22 Confirm no secret key reaches the browser: search the repository and the built files for secret and service role key values; expect only publishable ones.
- [ ] 23 Replace any key that was exposed: for each key pasted into chat, committed, or sent in a message, issue a new one and revoke the old.
- [ ] 24 Keep a recovery copy of every secret: saved secret values cannot be viewed again in Lovable, so confirm each one exists in the team password manager.

Domain and publishing

4 CHECKS

- [ ] 25 Confirm who controls the domain: sign in to the registrar and DNS host with a company account and record the renewal date and auto-renew status.
- [ ] 26 Connect the custom domain and choose the primary: load the root and www addresses over HTTP and HTTPS; expect all to end on one HTTPS address.
- [ ] 27 Publish the final version on purpose: publish after the last change and compare the live site with the editor, since later edits stay unpublished until you publish again.
- [ ] 28 Read the security findings at publish: open the scan results, resolve or accept each critical finding in writing, and run the deeper scan before launch.

Payments, email and integrations

3 CHECKS

- [ ] 29 Run one real payment with live keys, then refund it: confirm the charge, the receipt, and the database record agree, and that the refund is recorded.
- [ ] 30 Point every webhook at the production address: send a test event from each provider and expect a success response and a verified signature in the logs.
- [ ] 31 Send app email from a domain you control: deliver a test message to two mailbox providers and confirm it passes SPF and DKIM and lands in the inbox.

04 / 07

REFERENCE

Secrets and environment inventory

One row per key or setting. Replace the bracketed cells, add rows for anything missing, and delete rows that do not apply. Never write a secret value in this table.

The three VITE_ names are the ones Lovable documents for projects on its built-in backend; your project may use different names, so copy them from your own .env file. Values with a VITE_ prefix are embedded in the browser bundle at build time, which is why a secret must never use that prefix. Keep this table in the repository and the values in a password manager.

NAME	USED FOR	STORED IN	EXPOSURE	OWNER AND NEXT ROTATION
VITE_SUPABASE_URL	Backend address read by the browser	[.env in repository, host build settings]	Public by design	[NAME], rotation not needed
VITE_SUPABASE_PUBLISHABLE_KEY	Browser access to the backend, limited by RLS	[.env in repository, host build settings]	Public by design	[NAME], rotate if backend keys are reset
VITE_SUPABASE_PROJECT_ID	Identifies the backend project	[.env in repository, host build settings]	Public by design	[NAME], rotation not needed
Backend secret or service role key	Server-side access that bypasses RLS	[Function secrets only]	Secret, never in browser or repository	[NAME], [DATE]
Database password or connection string	Direct database access for exports and tools	[__]	Secret	[NAME], [DATE]
[PAYMENT PROVIDER] secret key	Creating charges and refunds from server code	[__]	Secret, live and test values kept apart	[NAME], [DATE]
[PAYMENT PROVIDER] webhook signing secret	Verifying that webhook events are genuine	[__]	Secret	[NAME], [DATE]
[EMAIL PROVIDER] API key or SMTP password	Sending sign-in and transactional email	[__]	Secret	[NAME], [DATE]
[SIGN-IN PROVIDER] client ID and secret	Social sign-in	[Auth provider settings]	ID public, secret is secret	[NAME], [DATE]
[AI OR API PROVIDER] key	Calls to [SERVICE] from server code	[__]	Secret	[NAME], [DATE]
[OTHER NAME]	[__]	[__]	[Public or secret]	[NAME], [DATE]

05 / 07 CHECKLIST

Operations after handoff

Make recovery, alerts, and future changes somebody's named job.

Backups and recovery

4 CHECKS

- [ ] 32 Confirm which backups exist for your backend and plan: open the backup screen and record how often backups run and how long they are kept in HANDOFF.md.

- [ ] 33 Take your own database export: download a full export, store it outside the platform, and record where it is kept and who can reach it.
- [ ] 34 Back up uploaded files separately: database backups and exports do not include storage files, so copy the buckets on a schedule and record the last run.
- [ ] 35 Rehearse a restore: load the latest export into a separate project, point a local build at it, and record how long recovery took.

Monitoring

3 CHECKS

- [ ] 36 Set an uptime check on the live domain: monitor the home page and one signed-in route, trigger a test failure, and confirm the alert reaches a named person.
- [ ] 37 Find the backend and function logs: cause a deliberate error in one server function and locate the entry in the logs within five minutes.
- [ ] 38 Set a usage or spend alert on every metered service: record the threshold and the person notified for hosting, backend, email, payments, and AI calls.

Change process

4 CHECKS

- [ ] 39 Decide where changes are made after handoff: write in HANDOFF.md whether edits happen in Lovable, in code, or both, and confirm every contributor has read it.
- [ ] 40 Decide how changes are reviewed: Lovable commits to the branch it syncs, so if you require pull requests, move Lovable to a working branch and prove one merge.
- [ ] 41 Review every database change before it is applied: read the migration file, take an export first, and record who approved it and when.
- [ ] 42 Tag the launch version in GitHub: confirm the tag exists and that HANDOFF.md states how to return the live site and the database to that version.

06 / 07

RUNBOOK

Go-live runbook

The order of operations for launch day. Replace bracketed values, assign each step to a named person, and stop at the first step that fails.

Database changes are harder to undo than code changes. A database restore discards everything written after the backup was taken, so decide in advance who may order one and prefer fixing forward when real users have already created data.

01 Three days before: confirm all 42 checks are passed or accepted in writing, the inventory has no empty owner cells, and the workspace is on a plan that allows custom domains.

02 Two days before: announce a change freeze from [DATE, TIME] to [DATE, TIME]. No prompts, commits, or database changes outside this runbook during the window.

03 One day before: take a full database export and a copy of the storage files, store both outside the platform, and write the location in the launch log.

04 One day before: add the DNS records shown in the domain setup screen at your DNS host. Proceed only when the domain shows as verified with a certificate issued.

05 Launch day, first: set the production site URL and redirect URLs in the auth settings, and add the production domain to every sign-in provider.

06 Next: switch payment, email, and other integrations to live keys through the secrets settings, and point each provider's webhook at the production address.

07 Next: publish the final version. Read the security findings shown at publish and stop if any critical finding is open and not accepted in writing.

08 Within [30] minutes of publishing: on the live domain, complete sign-up, sign-in, password reset, the core task, and one real payment followed by a refund. Use a signed-out browser for each.

09 Roll back if any smoke test fails and cannot be fixed within [60] minutes: restore the previous version of the app, switch integrations back, and restore the database only if launch changed its data.

10 For the first [24] hours: [NAME] checks uptime alerts, function logs, failed payments, and email bounces every [2] hours and logs each check with the time.

11 After [7] days without a rollback: tag the launch version in GitHub, lift the freeze, complete HANDOFF.md, and schedule the first restore rehearsal and key rotation dates.

07 / 07 TEMPLATE

Ownership handoff template

A HANDOFF.md to commit at the root of the repository. It records who owns what, where things live, and how to change and recover the app.

HANDOFF.MD · COPY AND ADAPT

```
# Handoff: [APP NAME]

Handoff date: [YYYY-MM-DD]
Handed over by: [NAME, CONTACT]
Received by: [NAME, CONTACT]
Second owner: [NAME, CONTACT]
```

```
## What this app does
[Two sentences: who uses it and the one task it must never fail at.]

## Where things live
| Item | Location | Owner account |
| --- | --- | --- |
| Lovable project | [PROJECT URL] | [WORKSPACE NAME] |
| Code repository | [GITHUB URL] | [GITHUB ORGANIZATION] |
| Synced branch | [BRANCH NAME] | |
| Backend | [Lovable built-in backend OR Supabase project URL] | [ACCOUNT] |
| Live domain | [https://DOMAIN] | Registrar: [NAME], renews [DATE] |
| DNS host | [PROVIDER] | [ACCOUNT] |
| Payments | [PROVIDER] | [ACCOUNT] |
| Email sending | [PROVIDER, SENDING DOMAIN] | [ACCOUNT] |
| Password manager vault | [VAULT NAME] | [ADMIN] |

## Access
| Person | Lovable | GitHub | Backend | Domain | Payments |
| --- | --- | --- | --- | --- | --- |
| [NAME] | [ROLE] | [ROLE] | [ROLE] | [ROLE] | [ROLE] |
| [NAME] | [ROLE] | [ROLE] | [ROLE] | [ROLE] | [ROLE] |

Last access review: [DATE] by [NAME]. Next review: [DATE].

## Secrets
Names, storage locations, and owners are in [PATH TO INVENTORY]. Values are only in [VAULT NAME].
Never paste a secret into chat, a commit, or this file.
Next rotation due: [DATE], owner [NAME].

## How to run it locally
1. Clone [GITHUB URL] and check out [BRANCH NAME].
2. Install: [COMMAND]
3. Create the local environment file from [EXAMPLE FILE]; values are in [VAULT NAME].
4. Start: [COMMAND]
5. Build: [COMMAND]

## How changes are made
- Changes are made in: [Lovable / code / both, with the rule for each].
- Review rule: [WHO REVIEWS, AND WHERE].
- Database changes: export first, then apply [HOW]; approved by [NAME].
- Going live: [WHO PUBLISHES OR DEPLOYS, AND HOW].
- Never delete, recreate, or disconnect the synced repository without [NAME]'s approval.

## Data and access rules
- Tables holding personal or payment data: [LIST].
- Row level security last tested: [DATE] by [NAME], using two test accounts and a signed-out session.
- Open findings accepted at launch: [LIST, EACH WITH THE PERSON WHO ACCEPTED IT].

## Backups and recovery
- Automatic backups: [FREQUENCY, RETENTION, WHERE TO FIND THEM].
- Manual database export: [SCHEDULE], stored at [LOCATION].
- Storage file copy: [SCHEDULE], stored at [LOCATION].
- Last restore rehearsal: [DATE], took [DURATION]. Next: [DATE].
- Who may order a database restore: [NAME].

## Monitoring and incidents
- Uptime alerts go to: [NAME, CHANNEL].
- Logs: [WHERE TO FIND APP, FUNCTION, AND AUTH LOGS].
- Spend alerts: [SERVICE: THRESHOLD, PERSON].
- If the app is down: [FIRST THREE ACTIONS, IN ORDER].
```

```
- Launch version tag: [TAG]. To return to it: [STEPS].

## Known issues and open decisions
- [ISSUE OR DECISION, OWNER, DUE DATE]

## Sign-off
Handed over by: [NAME, DATE]
Accepted by: [NAME, DATE]
```

01 Fill in every bracket or delete the line. A bracket left in place tells the next reader the answer is unknown.

02 Link to the secrets and environment inventory instead of copying it, so there is one list to keep current.

03 Update the file in the same pull request as any change to access, domains, backups, or the change process.

04 If an agency is handing over to a client, both parties sign the last section only after the client has logged in to every account as an owner.

FIELD NOTES

Common mistakes, useful habits, hard limits.

01 **Treating a working preview as a tested access model**
An app can look correct to its builder while any signed-in user can read another user's rows. Only a test with two accounts and a signed-out session shows what the policies allow, so run those tests before real data arrives.

02 **Leaving the accounts in one person's name**
If the repository, the backend, the domain, or the billing sits under a single personal login, the product stops when that person is unavailable. Put each account under a company-controlled owner and add a second owner before launch.

03 **Assuming the platform backup covers everything**
Database backups and exports do not include uploaded files, secrets, or sign-in provider settings. Keep a separate copy of storage files, keep secret values in a password manager, and write the auth settings down.

PRACTICAL TIPS

Create the two test accounts once and keep their credentials in the password manager. You will rerun the access tests after every change to tables or policies.

Do the clean-clone build early. It finds missing environment values and unwritten setup steps while the original builder is still available to ask.

Write the change rule before the first post-launch edit. Two people editing the same branch from Lovable and from code is easier to prevent than to untangle.

Put renewal dates for the domain, paid plans, and payment cards on a shared calendar with two attendees.

Rerun the Database and sign-in section whenever a new table, storage bucket, or sign-in method is added.

WHEN NOT TO USE THIS

A prototype or demo with no real users, no real data, and no payments. Run only the GitHub sync group so the code is safe, and return to the rest before launch.

A plan to rebuild the app on a different stack. Use the inventory and HANDOFF.md to record what exists, and plan the rebuild as its own project.

A static marketing site with no sign-in and no database. The repository, domain, and publishing checks are enough.

SOURCES · CHECKED 2026-11-22

Lovable Documentation, Sync your project with GitHub · Sync is two-way, Lovable edits and syncs one branch at a time (the default branch unless switched), the repository is private by default, deleting the repository stops sync, disconnecting leaves the repository on GitHub, reconnecting creates a new repository, and Git sync cannot link to an existing repository.
https://docs.lovable.dev/integrations/github

Lovable Documentation, Deployment, hosting, and ownership options · Apps, code, and content created with Lovable belong to the user, subject to third-party rights. Git sync and the codebase download include source and migration files but not data; the database and storage files are exported separately.
https://docs.lovable.dev/tips-tricks/deployment-hosting-ownership

Lovable Documentation, Deploying and hosting outside Lovable · Projects on the built-in backend use VITE_SUPABASE_URL, VITE_SUPABASE_PUBLISHABLE_KEY, and VITE_SUPABASE_PROJECT_ID from the .env file; VITE_ variables are embedded at build time; a new production domain must be added to OAuth redirect URLs.
https://docs.lovable.dev/tips-tricks/external-deployment-hosting

Lovable Documentation, Publish · The live site keeps showing the version last published, and later changes stay off it until the project is published again.
https://docs.lovable.dev/features/publish

Lovable Documentation, Custom domain · Custom domains are available on paid plans, the project must be published before the domain serves the site, the required DNS records are shown in the setup window, Lovable issues the SSL certificate automatically, and all other connected domains redirect to the primary domain.
https://docs.lovable.dev/features/custom-domain

Lovable Documentation, Security overview · A quick scan runs automatically at every publish and checks database rules including tables without row level security; a deep scan is started manually; API keys pasted into chat are detected; the tools cannot guarantee complete security and do not replace a thorough security review.
https://docs.lovable.dev/features/security

Lovable Documentation, Secrets · Secrets are encrypted, injected into Edge Functions, and never reach the browser. A saved value cannot be viewed again, only replaced or deleted. VITE_ prefixed values belong in the .env file, not in Secrets, because they are embedded in the client bundle.
https://docs.lovable.dev/features/secrets

Lovable Documentation, Built-in backend · The built-in backend provides a database, storage, authentication, realtime, and functions and builds on Supabase's open-source foundation. Lovable does not move a project from the built-in backend to your own Supabase project in one step.
https://docs.lovable.dev/integrations/cloud

Lovable Documentation, Connect to Supabase · With a connected Supabase project, Supabase bills backend usage, secrets are stored in the Supabase project, authentication settings live in Supabase, each Lovable project connects to one Supabase project at a time, and there is no automatic migration between the built-in backend and your own Supabase project in either direction.
https://docs.lovable.dev/integrations/supabase

Lovable Documentation, Back up, restore, and export your project's database · On the built-in backend Lovable takes one database backup per day and keeps roughly 14 days. A restore cannot be undone and covers the database only. An export contains the full database but not storage files, edge function code, secrets, sign-in provider settings, or API keys, and can be requested once every 24 hours.
https://docs.lovable.dev/features/database-backups

Supabase Docs, Row Level Security · RLS should be enabled on every table in an exposed schema; with RLS on and no policies, no data is accessible with a publishable key; auth.uid() returns null without an authenticated user; views bypass RLS by default; with check governs the resulting row; user-editable metadata is not a good place for authorization data.
https://supabase.com/docs/guides/database/postgres/row-level-security

Supabase Docs, Understanding API keys · Publishable keys are safe to expose in web pages and source code. Secret keys and the legacy service_role key bypass Row Level Security and must never be used in a browser.

https://supabase.com/docs/guides/api/api-keys

Supabase Docs, Redirect URLs · The Site URL is the default redirect when no redirectTo is given and matters for email confirmations and password resets; allowed redirect URLs are set on the URL Configuration page; exact paths are recommended in production.

https://supabase.com/docs/guides/auth/redirect-urls

Supabase Docs, Send emails with custom SMTP · The default email service only delivers to addresses in the project's team, is limited to 2 messages per hour, and is not meant for production use; a custom SMTP server is needed for real users.

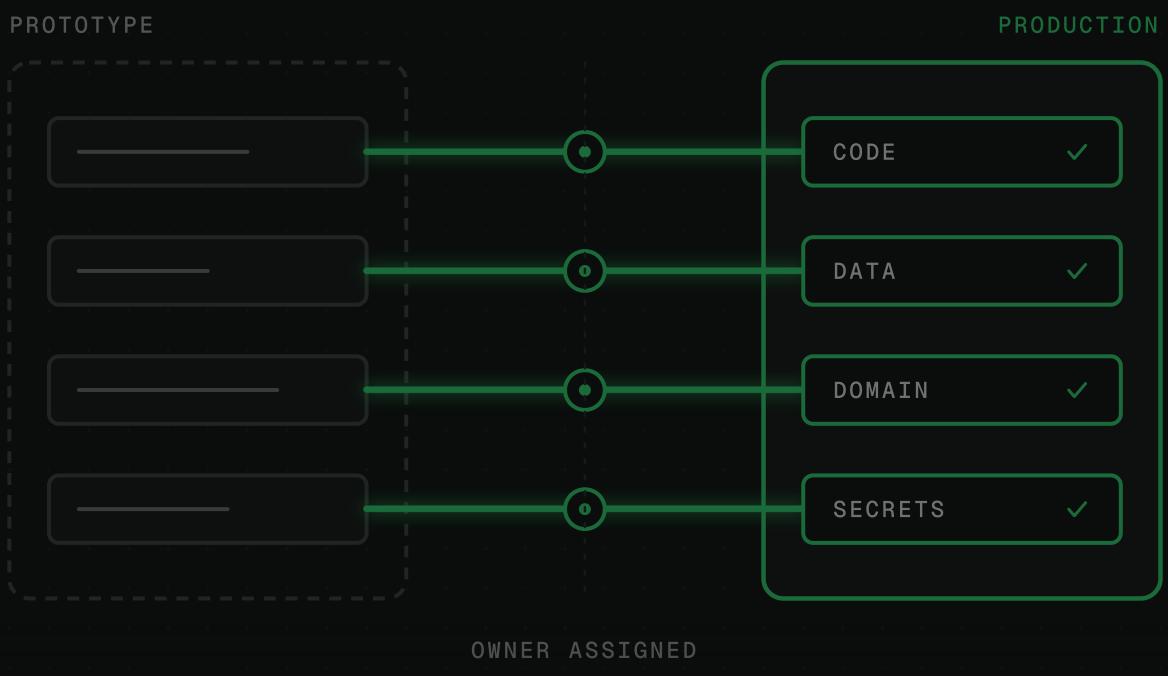
https://supabase.com/docs/guides/auth/auth-smtp

Supabase Docs, Database Backups · Daily backups are taken for Pro, Team, and Enterprise projects with 7, 14, and up to 30 days of access respectively; Free projects are advised to export their own data; database backups do not include objects stored through the Storage API.

https://supabase.com/docs/guides/platform/backups

Supabase Docs, Production Checklist · Free Plan projects with low activity in a 7-day period may be paused, and database backups are not available for download on the Free Plan.

https://supabase.com/docs/guides/deployment/going-into-prod



FAISAL KARKOH / RESOURCES

More working files for building with AI.

Prompt packs, checklists, templates and playbooks for shipping reliable AI products. Free to download and reuse in your own work.

LIBRARY
faisalkarkoh.com/resources

WRITING
faisalkarkoh.com/blog

WORK TOGETHER
faisalkarkoh.com/contact