

WEBINAR · 25 SEPTEMBER 2026 · 1:00 PM CEST

Product Development in 2026

From Concept to Delivery using AI

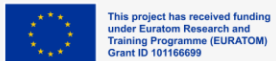
Figma Make → Claude → production

Romain Cauhet · Marketing Engineer | Oussama Radouche · R&D Engineer

AQUILAB by Coexya



Risk assessment *Tools* for severe side *Effects* after *breasT* Radiotherapy:
radiation safety through biological extended models and *digital twinS*



This project has received funding
under Euratom Research and
Training Programme (EURATOM)
Grant ID 101166699

EU Grant Agreement n. 101166699

Your speakers

Two jobs



PART 1 · DESIGN & USER FLOW

Romain Cauhet

Marketing Engineer, AQUILAB by Coexya
Product design, branding, go-to-market

Part 1: from brand guidelines to a clickable product in Figma Make.

PART 2 · CODE & DELIVERY

Oussama Radouche

R&D Engineer, AQUILAB by Coexya
Application development, architecture, deployment

Part 2: from generated code to a product running for real.

We build medical software. Nothing we show today is specific to that field: the method is what matters.



Agenda

Three threads: what Figma Make is, what changes in the workflow, what changes in the way we work

- | | | |
|---|---|------------------|
| 1 | Introduction — what AI changes in the product pipeline | Both · 5 min |
| 2 | Part 1 — Figma Make: understand the tool, then our 7-step method | Romain · 15 min |
| 3 | Bridge — from prototype to code | Both · 3 min |
| 4 | Part 2 — from code to production, with Claude | Oussama · 15 min |
| 5 | Key takeaways | Both · 2 min |
| 6 | Your questions | 5 min |



The pipeline we all know...

Linear, sequential, full of handovers



TYPICAL DURATION

6 to 12 weeks

Before anyone outside the team can click on anything.

COST

6 handovers

Each handover re-explains the intent and loses part of it.

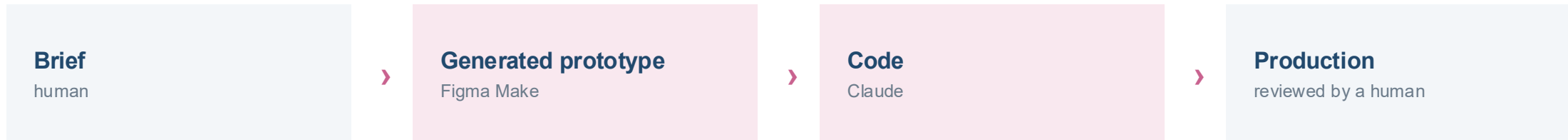
RISK

1 feedback loop

User feedback arrives after the most expensive phase.

The pipeline we actually used

Fewer steps, earlier proof

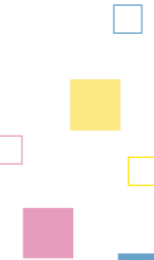


Two things changed, and only two.

The prototype is now the specification: there is no point writing twenty pages to describe screens that already exist and can already be clicked through.

The prototype's code is now the starting point: it is no longer thrown away after the demo, it is cleaned up and kept.

Everything else (reviews, testing, security, deployment) is exactly as demanding as before.



The bottleneck has moved

This shift alone justifies the new method

BEFORE

Producing was expensive

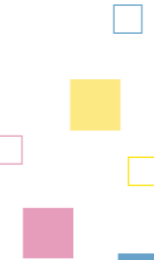
- Drawing every state of every screen
- Writing the plumbing code
- Redoing everything after each review

NOW

Deciding and verifying are expensive

- Choosing what is actually worth building
- Knowing which generated output deserves trust
- Proving it is safe to ship

AI did not remove the work. It moved it.



What none of this replaces

A necessary framing before getting into the method

Product decisions

What to build, what to cut, what to refuse. A model will perfectly build a product that should never have been built.

Domain knowledge

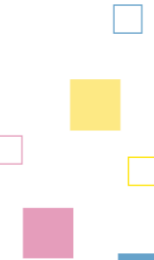
In our field, a wrong default value is not a software bug but a clinical risk. The model cannot know that.

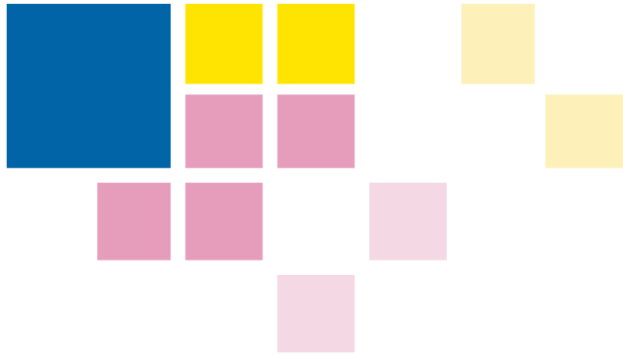
Security and compliance

Access rights, personal data, traceability. This is where generated code fails silently.

Accountability

When an incident happens, a person is named on the report. Never the model.



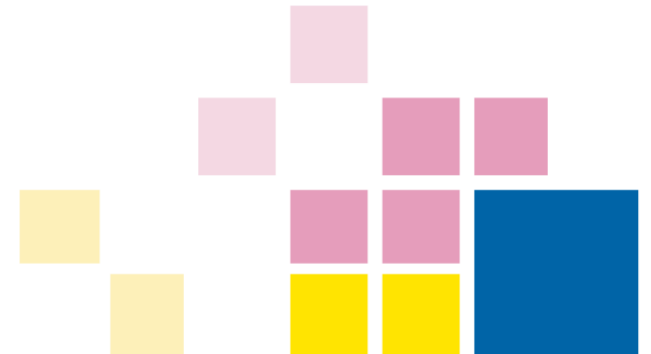


PART 1

Figma Make: understand the tool, then build with it

Romain Cauhet

Marketing Engineer · AQUILAB by Coexya



Part 1: the plan

First the tool, then the method

1 · UNDERSTAND

Figma Make, in six questions

- What is it?
- How do you use it?
- What can you build with it?
- Who is it for?
- What do you need to get started?
- Why start there?

2 · BUILD

Our method, in seven steps

- 1 Write the brief
- 2 Import the design system
- 3 Write the first prompt
- 4 Attach what already exists
- 5 Iterate
- 6 Validate the flow
- 7 Put it in front of users

Then the watch-outs, and a recap of what you actually have in hand.

Figma Make: what it is and how it works

A prompt goes in, a working app comes out

A prompt-to-app tool

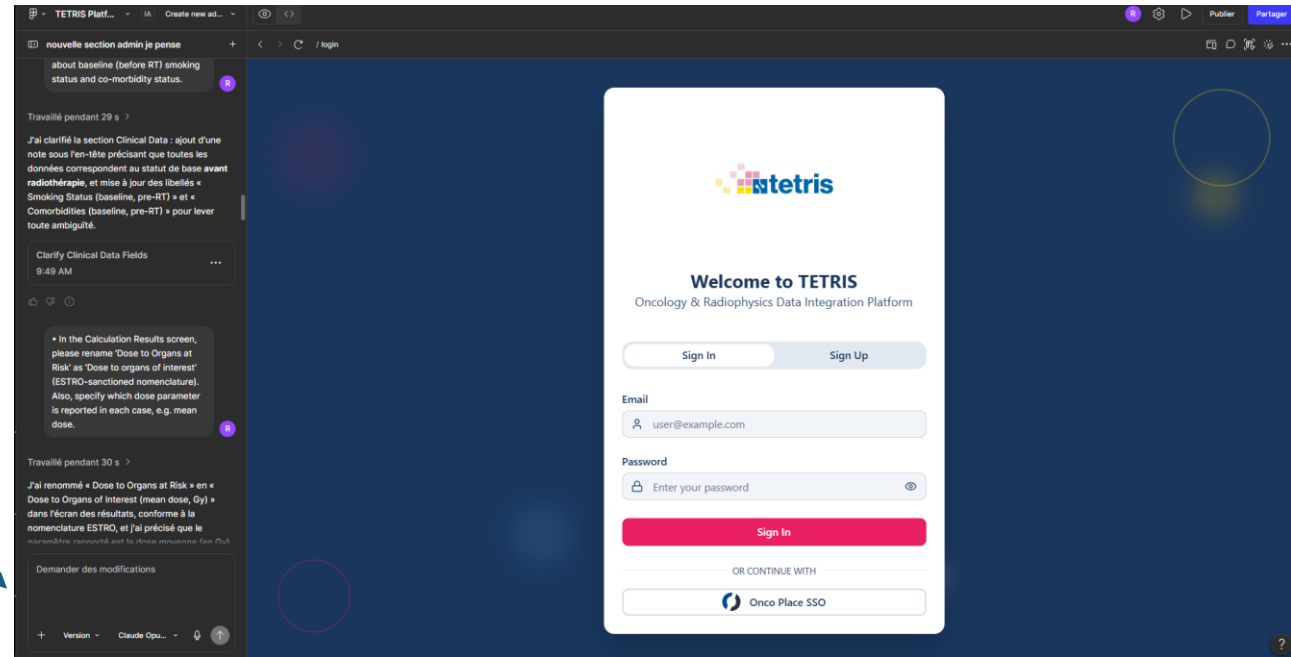
You describe the product in a conversation; the tool builds an interactive prototype or a web app that actually works.

Two work areas

The conversation on one side, the live preview on the other, with editable code underneath.

Available to everyone

The tool is out of beta. Publishing Make files is still in beta: keep that in mind before committing to a client URL.



How you use it: describe, generate, adjust

A short loop, repeated as many times as needed

Describe

In the chat, in plain language: what you want, who it is for, in what style.



Generate

Within seconds, the tool produces a prototype you can already click through, shown in the preview.



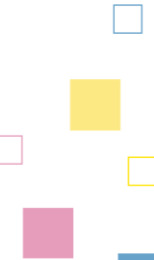
Adjust

One request at a time: make a header bigger, change a colour, add a screen.

🔄 Then back to “Describe”, until you get the result you want.

To get started: from your Figma dashboard, create a new file of type Figma Make.

It is never right the first time: it is a dialogue, not a one-shot order.



What you can build with it

The most common uses — and what stays available behind the screen

Campaign pages and showcase websites

Task management apps

Booking and appointment tools

Data dashboards

Learning tools and quizzes

Internal team tools

THE DETAIL THAT MATTERS

The code stays accessible

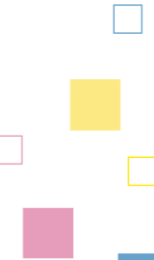
The generated code (HTML, CSS, React) can be viewed and exported.

That is exactly what makes the second half of this session possible.

The limit to keep in mind

These are working starting points, not finished products: the rest of the session is precisely about the gap between the two.

Based in part on S. Wilawun, “Figma Make: The Beginner’s Guide to Building Apps Without Coding”, Medium, June 2026.



Who the tool is for

The core promise: no technical skills are needed to produce a first version

Product and marketing people

Mock up a website, a booking tool or a campaign page without filing a development request.

Project owners

Show a committee or an investor what the app would look like, before any budget is committed.

Creative people

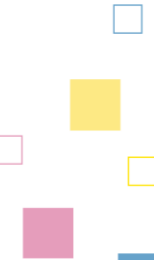
Turn a visual intent into screens you can interact with, without writing code.

Technical teams

Get a concrete basis for discussion instead of a spec written in the dark.

The nuance that matters in a product team

The main effect is not to remove development, but to move the moment the first version exists — and to widen the circle of people able to produce it.



What you need to get started

Three prerequisites, only one of which is a real blocker

1

A Figma account

Free, usable straight from the browser; a desktop app is also available.

2

The right licence

Anyone can try the tool, but creating a Figma Make file from scratch requires a Full seat on a paid plan.

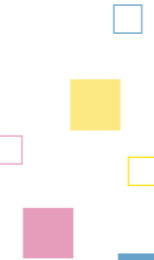
3

An idea, even a rough one

The initial level of detail hardly matters: it builds up as you go.

AI credits: the thing to check before running a workshop

Every prompt uses credits. Full seats currently have unlimited credits; other plans work with a daily quota that resets every day.



Why start here, and not in a code tool

Both approaches lead to the same app; they do not involve the same people

Your brand already lives here

The palette, the typography, the components are already in Figma. You don't describe them to the model, you hand them over.

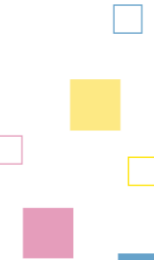
Non-technical people stay involved

A marketing manager, a clinician or a product owner can change the flow themselves, which is not possible in a development environment.

Validation comes before investment

The expensive work (back-end, authentication, infrastructure) only starts once the flow is validated.

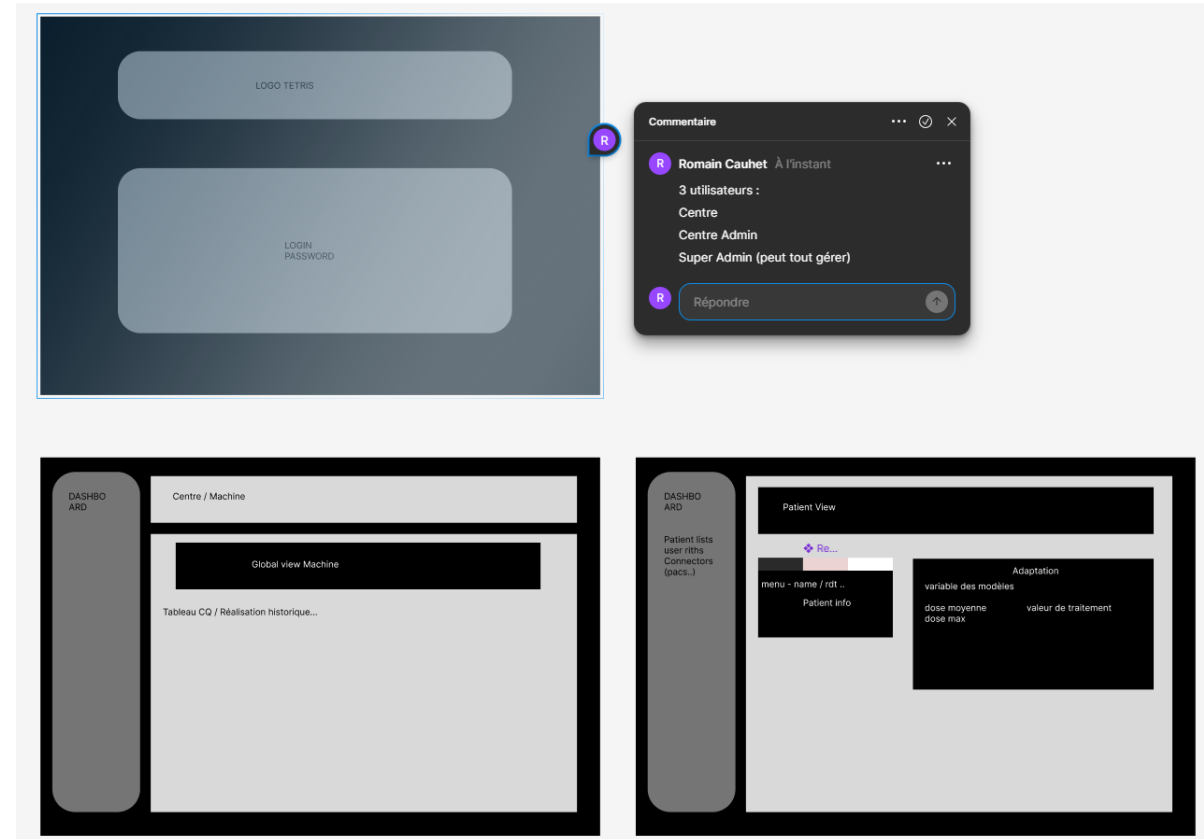
One honest caveat: a team of three back-end developers and no designer is better off starting in the code tool. Tools adapt to teams, not the other way round.



Step 1: write the brief, by hand

The brief is the only deliverable we never generate

- 1 **Who it is for**
Named roles, not “the users”.
- 2 **The problem, in one sentence**
If it takes three, the scope is wrong.
- 3 **The screens you think you need**
Name them. Four is often enough.
- 4 **One hard constraint**
Data, regulation, hardware, deadline.
- 5 **What “done” means**
The test you run before calling the step finished.





Step 2: import your design system

Before the first prompt, not after a first disappointing result

Import an existing Figma library and it inherits:

- the colour palette and how to use it
- the typography
- the styling of your components

Skip this step and the result is decent but generic, like every other generated app. It is the most obvious giveaway.

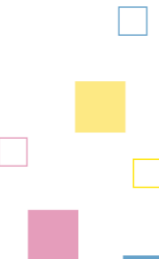
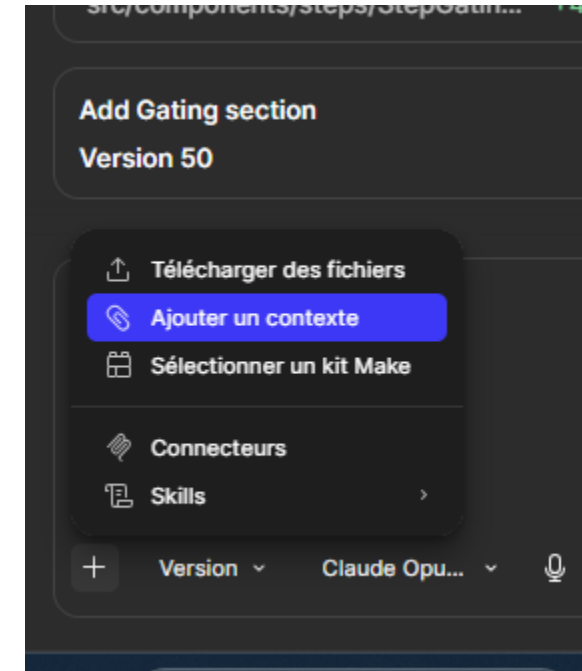
Download files

Add context

Select a Design kit

Connectors

Skills





Step 3: write the first prompt

Six blocks, in this order — the reference slide of Part 1

ROLE

Who the app is for (in their own words).

PRODUCT

What it does, in one sentence.

USERS

The roles, and what each one is allowed to see.

SCREENS

Name them and list them: this is not the model's call.

CONSTRAINTS

What must never happen, stated precisely.

TOPE

The visual register, aligned with the imported library.

Six blocks, about 150 words. Any shorter and the result becomes a gamble.





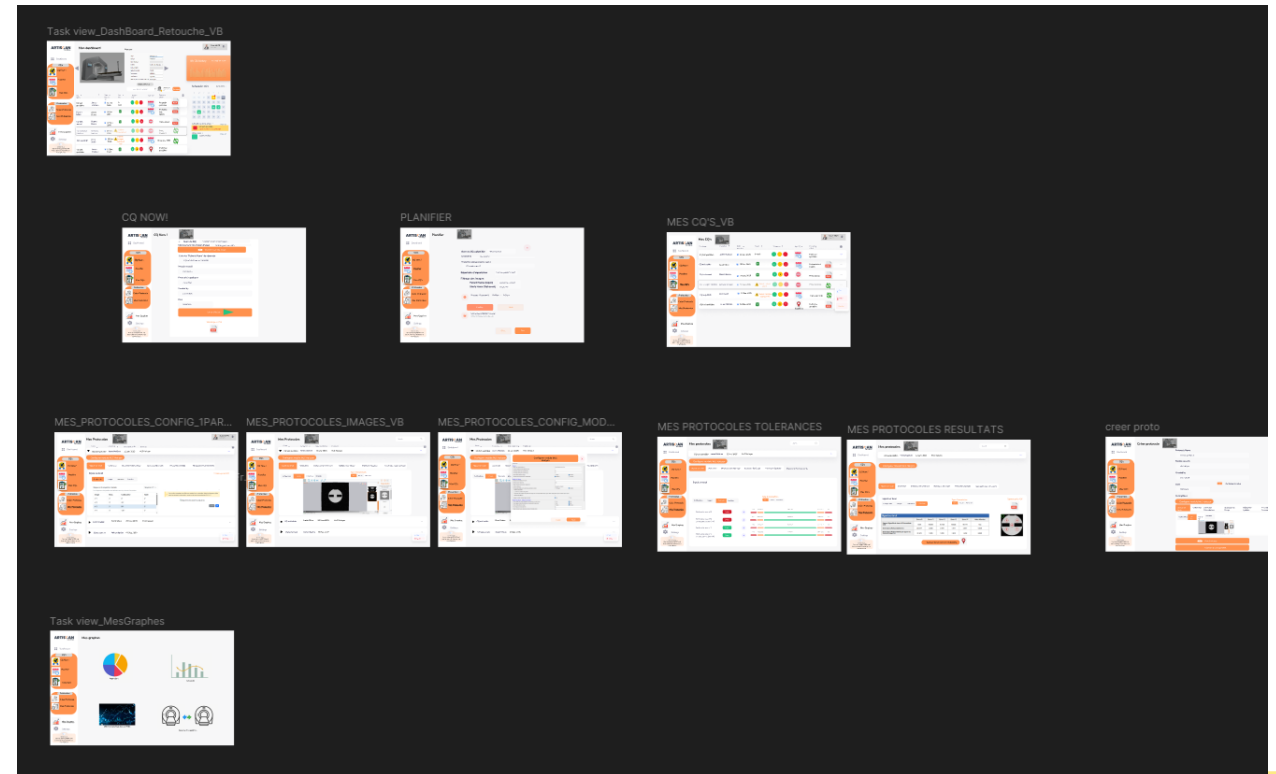
Step 4: attach what exists instead of describing it

No need to describe a component that already exists

You can attach to a prompt:

- existing designs and frames
- your components
- content from the Figma Community

A screenshot of the real screen always beats three paragraphs of description.





Step 5: iterate — chat for structure, click for detail

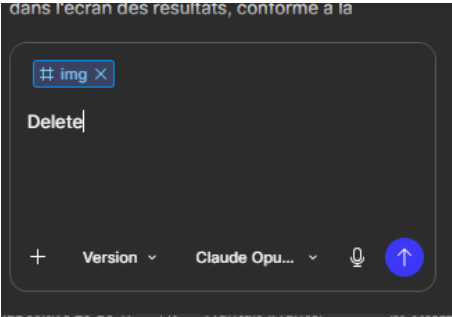
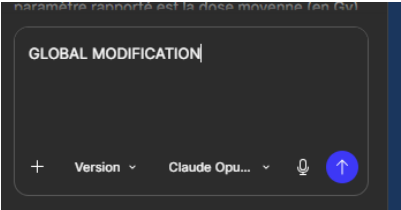
The classic mistake: doing everything through the chat

Use the chat to

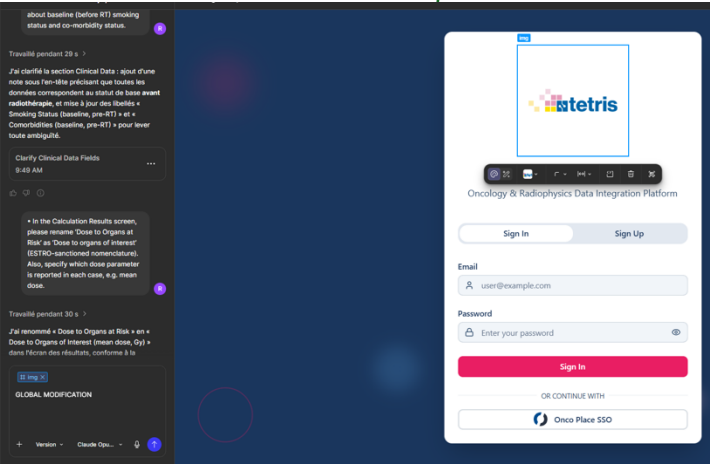
Add a screen, change the flow, rework an entire section.

Click the element to

Adjust margins, colours, text properties — or annotate it to ask the agent for one precise change.



Regenerating the whole app to move one button costs you half a day.





Step 6: validate the flow, not the pixels

The deliverable expected at the end of Part 1

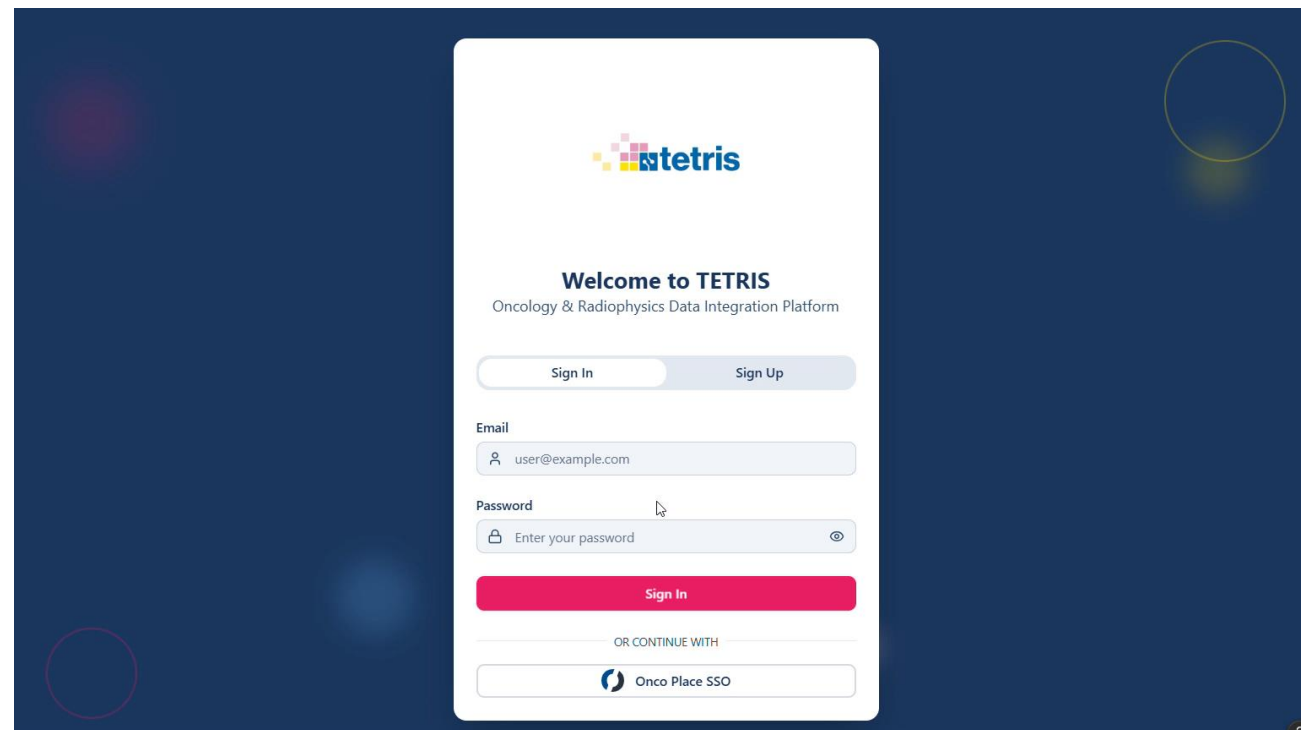
Click through every screen in front of someone who will actually use the tool.

Fake data, no database, no login: none of that matters yet.

What you are looking for:

the forgotten screen, the step with two clicks too many, the field that doesn't exist in real life.

Spotting a missing screen now costs ten minutes; discovering it once the back-end is wired costs a week.





Step 6: the validation checklist

Five criteria, all required: otherwise Part 2 pays the price.

✓	The happy path	From start to finish, without a single dead end.
✓	Empty states	What the very first user sees, with no data at all.
✓	Error states	What happens when it fails, designed upfront rather than improvised.
✓	Responsive behaviour	The screen sizes your users actually have.
✓	Naming	One term per concept, everywhere. This is the criterion that protects Part 2.



Step 7: put it in front of users

A test link beats a presentation

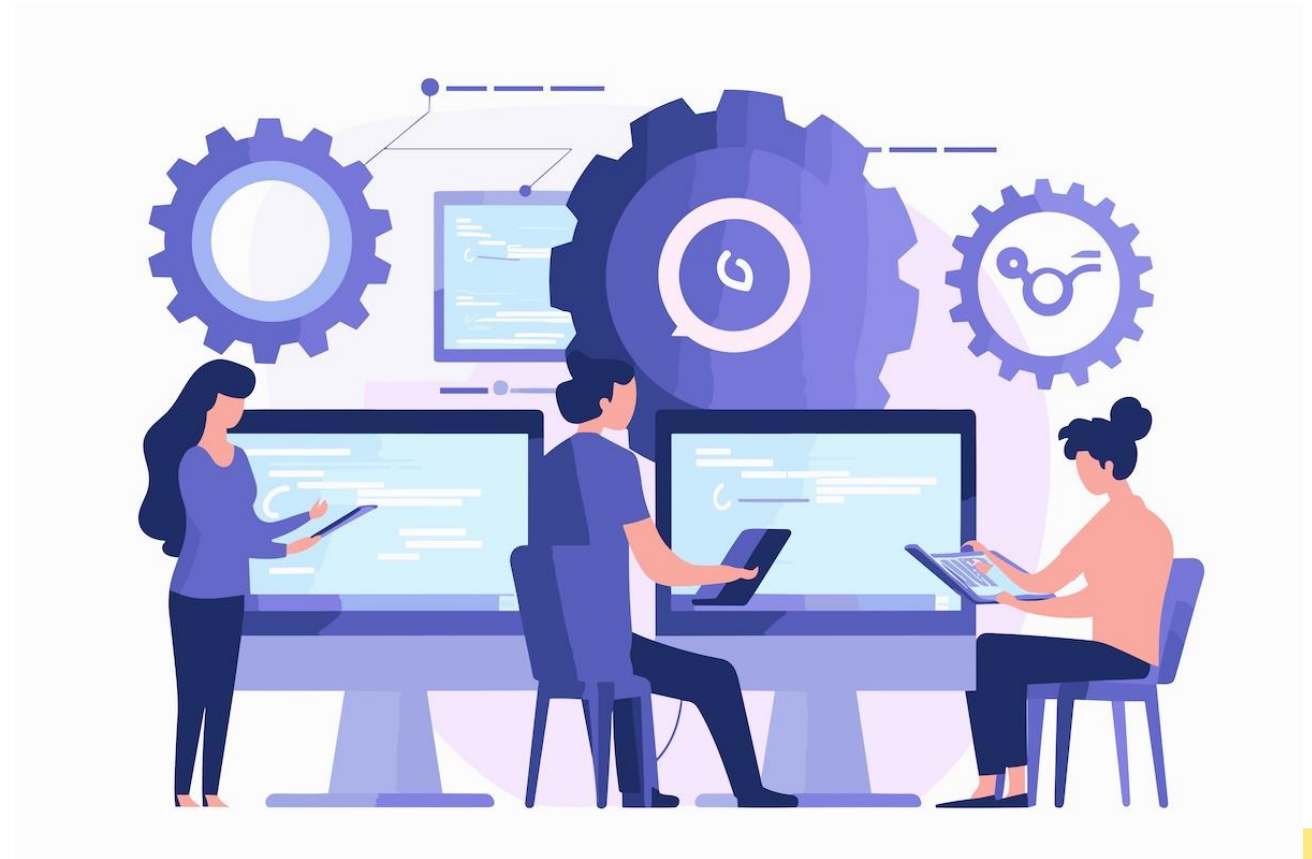
The file can be shared privately or published; publishing is still in beta and limited to Full seats.

A limitation to keep in mind

Decide where feedback will be collected before you send the link, otherwise it scatters and part of it gets lost.

The screenshot shows the 'Publier' (Publish) dialog box in Figma. It has a dark background and a close button (X) in the top right corner. The dialog contains the following fields and options:

- Titre** (Title): Review Projet V2
- URL**: example.figma.site ⓘ
- État** (Status): Non publié
- Problèmes** (Problems): Aucun problème
- Mise en ligne sur la Communauté** (Publish to Community): A toggle switch that is currently turned off.
- En publiant, vous acceptez les Conditions de la version bêta de Figma et la Politique d'utilisation acceptable.** (When publishing, you accept the Figma beta version Conditions and the Acceptable Use Policy.)
- PUBLISH**: A large blue button with the word 'PUBLISH' in yellow text.



Three mistakes to avoid

Seen on our own projects

Credits burn on vague prompts

Every vague prompt triggers a full regeneration. Writing 150 precise words costs less than five rough attempts.

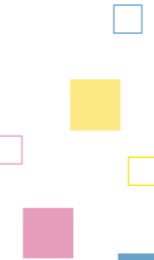
Third-party content slipping in

Make may pull in fonts, code packages or images from the internet. It is up to you to check the rights before publishing.

Mistaking the prototype for a codebase

It is a working prototype, not reviewed, not tested, not secured. Treating it as production-ready is exactly the mistake Part 2 is designed to prevent.

None of these three points is a reason not to use the tool; they are reasons to know exactly what you are holding.



The shortcut: a database without leaving Figma

Useful, but not enough

BENEFIT

What it is great for

Make connects to a Supabase back-end: the prototype can store and read real data.

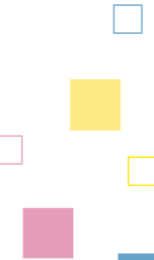
Internal proof of concept, a demo that survives a page refresh, testing a data model before committing to it.

LIMIT

Where it stops

Access rights, traceability, environments, testing, backups, monitoring, and everything a regulated product requires.

A prototype with a database is still a prototype.



Part 1 recap: what you have in hand

Four outcomes at the end of the design phase

A validated flow

By the people who will use it, not assumed.

An on-brand interface

Built from your real library.

A real codebase

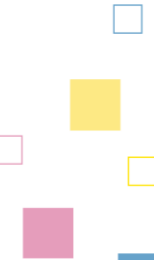
React, working, imperfect, and yours.

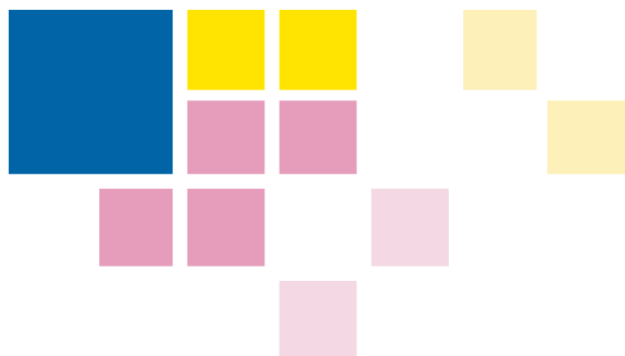
Decisions already made

The trade-offs were settled while changing your mind still cost nothing.

The third point deserves special attention.

Most teams throw the prototype away and start from scratch. That reflex made sense when prototypes were drawings; it makes far less sense now that they are code.





BRIDGE

From prototype to code

Romain Cauhet & Oussama Radouche

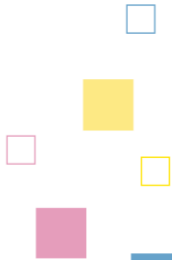
What you export, and why you don't throw it away



Prototype ≠ product

The gap between the two, point by point

	PROTOTYPE	PRODUCT
Data	Fake, in the code	Real, in a backed-up database
Users	One, implicitly	Roles, permissions, edge cases
Identity	None	Authentication you can defend
Confidence	An occasional manual check	Tests run on every change
Failure	Reload the page	Monitoring, alerts, someone on call



What is actually in the export

Look before you judge

What you will recognise

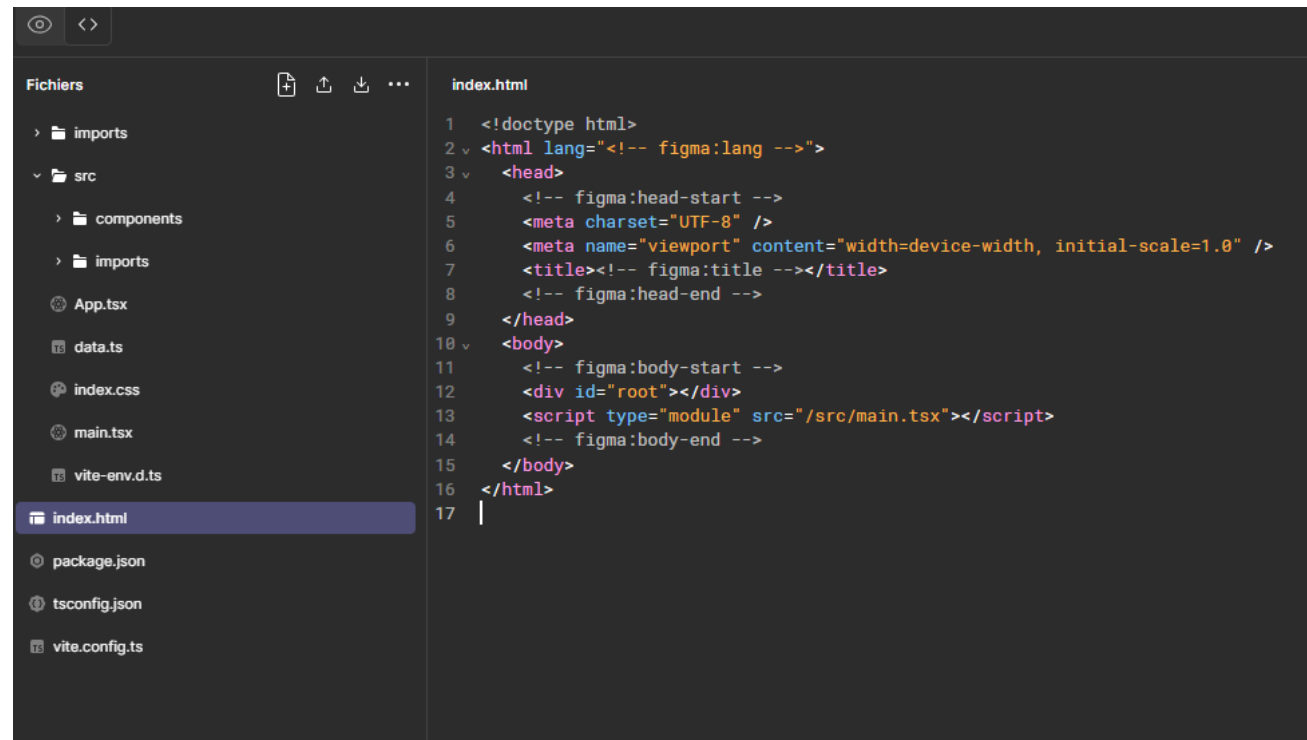
One component per screen, styling, routing, and data hard-coded right where it is used.

What you will keep

The interface layer and the structure of the flow, i.e. most of the visible work.

What you will replace

Everything pretending to be a back-end, and every value typed directly into a component.



The screenshot shows a code editor with a dark theme. On the left, a file explorer pane titled 'Fichiers' displays a directory structure: 'imports', 'src' (expanded), 'components', 'imports', 'App.tsx', 'data.ts', 'index.css', 'main.tsx', 'vite-env.d.ts', 'index.html' (selected), 'package.json', 'tsconfig.json', and 'vite.config.ts'. The main editor area shows the content of 'index.html' with line numbers 1 through 17. The code is a basic HTML template with Figma comments for head and body start/end markers.

```
1 <!doctype html>
2 <html lang="<!-- figma:lang -->">
3 <head>
4   <!-- figma:head-start -->
5   <meta charset="UTF-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7   <title><!-- figma:title --></title>
8   <!-- figma:head-end -->
9 </head>
10 <body>
11   <!-- figma:body-start -->
12   <div id="root"></div>
13   <script type="module" src="/src/main.tsx"></script>
14   <!-- figma:body-end -->
15 </body>
16 </html>
17
```

Two ways out of Figma Make

Only one of them is a workflow

OPTION A

Download the code

You get files, once.

No history, no branches, no way to see what changed when the designer regenerates a screen the following week.

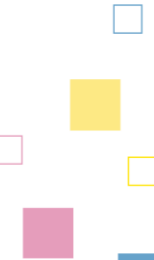
OPTION B — THE ONE TO PICK

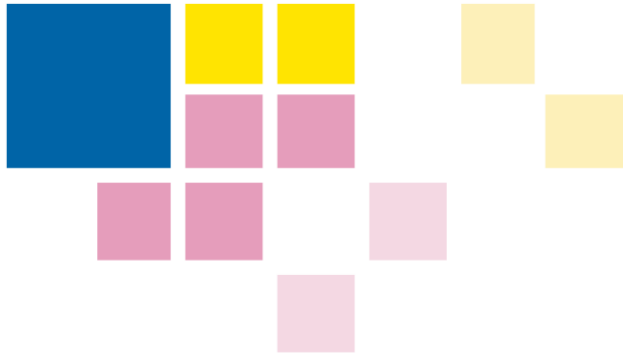
Export to GitHub + work with Claude

Figma creates a private repository and writes commits and branches into it.

You start versioned. Every later change becomes a readable diff (a clear comparison between two versions of a text or of code) that can be reviewed and reverted.

Simple rule: if the handoff is a zip file, the handoff is already broken.



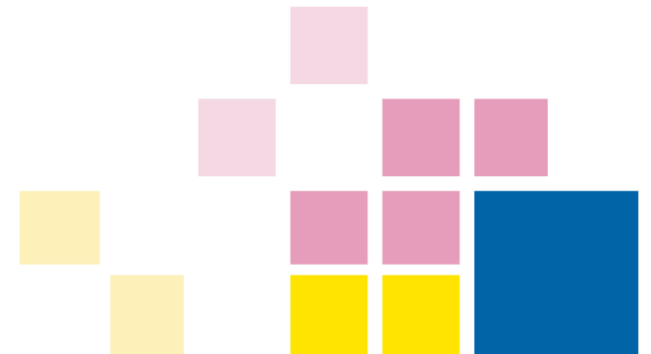


PART 2

From generated code to a live product

Oussama Radouche

R&D Engineer · AQUILAB by Coexya



Setup: one repository, one agent, one source of truth

Five minutes that save three weeks of confusion

1

The repository

The exported repository, private, with the main branch protected.

2

The agent, inside the repository

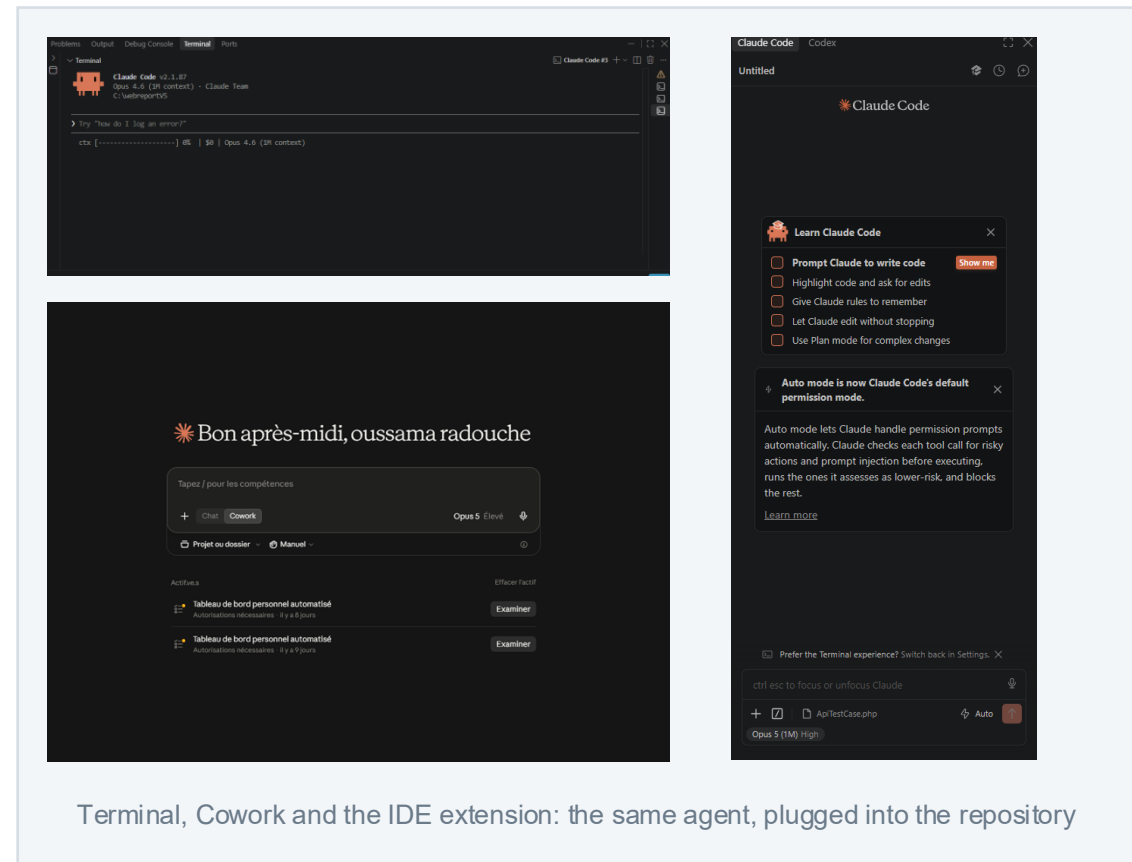
Claude works on the real project, not on pasted snippets.

3

The boundary

Design lives in Figma. Code lives in Git. Nobody edits production by hand.

Copying code into a chat window and pasting it back is not a workflow: it adds steps and wipes out the history.



Step 1: audit before you generate

The first prompt should produce zero lines of code

"Read this repository. Explain the architecture back to me. List what is mocked, what is duplicated, and what will not survive production. Do not write any code yet."

Why it works

You know straight away whether the agent has understood the project, while the cost of a mistake is still zero.

And you get a task list written in the vocabulary of your own codebase, which becomes the plan for the next hour.

```
> Read this repository. Explain the architecture to me.
```

Architecture

```
api/      routes and controllers
core/     business rules
legacy/   old module, still called
```

Points of attention

```
two versions of the same service
hard-coded test data
```

9 tasks proposed, ranked by risk

Sample audit output

Step 2: write the context file, once and for all

The difference between a useful agent and a fast one

The tech stack and versions

The folder structure, and why it is that way

Naming and coding conventions

Build, test and lint commands

What must never be touched

CLAUDE.md

Stack

Node 20, TypeScript, Postgres 15

Structure

api/ routes core/ business rules

ui/ components tests/ unit

Conventions

camelCase, one component per file

no logic in views

Commands

npm run build | test | lint

Never touch

migrations/, .env, legacy/

Sample context file

Written once, read at the start of every session, for the whole life of the project. It is the most important file in the repository.

Step 3: the clean-up pass

Turning a prototype into a foundation you can build on

Pull the data out

Every hard-coded value leaves the component and becomes data the app fetches. That is what makes the back-end pluggable.

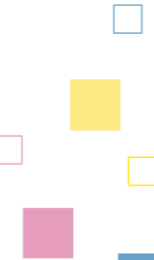
Factor out the repetition

Make generates each screen independently. Three near-identical components become one.

Declare the types

Name your data structures now: every later step depends on them.

One goal per session. Three goals in one prompt produce a diff nobody can review.



Step 4: design the data model through dialogue

The step where human expertise matters most

STEP 1

You describe

The entities and business rules, in plain language. No schema, no jargon. Just how the domain actually works.

STEP 2

It proposes

A schema, with its relationships, constraints, and the questions it could not settle on its own.

STEP 3

You challenge it

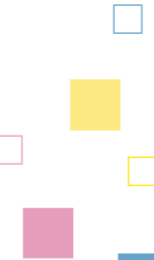
This is where domain knowledge pays off. Most proposals are 80% right and badly wrong on one relationship.

STEP 4

You decide

And the decision is written down, in the repository, with its reason.

A generated schema is a draft. A reviewed schema is a decision.



Step 5: the interface contract first

Types, then API, then interface, in that order

Types

The shape of the data, agreed and frozen.



API

The endpoints that serve exactly those shapes.



Interface

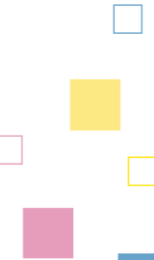
The screens, rewired to consume them.

Why the order matters

Generated in parallel, the interface and the back-end drift apart within two prompts: the same field named three ways, dates in two formats, a list that is sometimes empty and sometimes null.

A frozen contract gives the agent a reference to stick to, and saves hours of debugging.

This old engineering discipline has not become optional: it has become load-bearing.



Step 6: authentication and access rights

Where generated code fails silently

THE TRAP

What happens

The login screen works. The session works. It all looks finished.

Then a user with the wrong role requests a record directly and gets it, because the check was in the interface and never on the server.

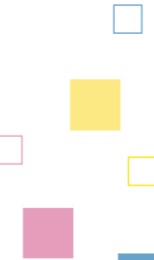
THE HABIT

What we do

Write down who may see what, record by record, before generating anything.

Then test every rule explicitly, including — especially — the ones that seem obvious.

An access-rights bug causes no visible failure: that is exactly what makes it dangerous.



The working rhythm that makes this sustainable

Speed comes from reviewability, not from bigger prompts

One intention per branch

A branch that does two things is a branch nobody reviews properly.

Readable commits

Written for the person who will be tracking down a bug eight months from now.

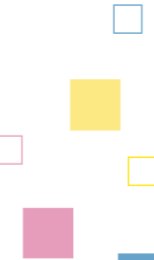
Tests as the checkpoint

Not for coverage, but so that “it works” is a verified fact rather than an opinion.

If you can't review it, it's too big

Revert, split, redo. Every time, no exceptions.

The agent produces far more code than you can reasonably accept. Managing that gap is most of the job today.



What a human still reads line by line

Four non-negotiable areas, whatever the deadline

Security

Authentication, sessions, secrets, anything touching an external boundary.

Data access

Every query that returns something belonging to someone else.

Money

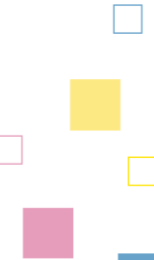
Pricing, billing, quotas. Silent errors here cost you twice.

Personal data

What you store, why, for how long, and who can read it.

Everything else can be reviewed at the level of the diff.

These four are read as if you had written them yourself: as far as your users, your auditor and your regulator are concerned, you did.

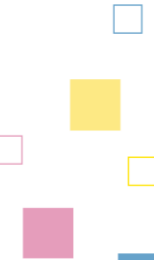


Step 7: environments, CI pipeline, deployment

Principles that don't depend on your tools

- 1 Three environments, minimum**
Development, staging, production. Staging is not a luxury: it is where you find out what the agent didn't tell you.
- 2 Secrets out of the repository**
Asked carelessly, the agent will hard-code a key to make a test pass. Check for this specifically.
- 3 Automated checks before release**
Build, tests and lint on every change, systematically.
- 4 A rollback you have actually tested**
An untested rollback is a hope, not a plan.

Your stack will differ from ours; the order of these four points will not.



After go-live: the phase nobody demos

Going live is a milestone, not the finish line

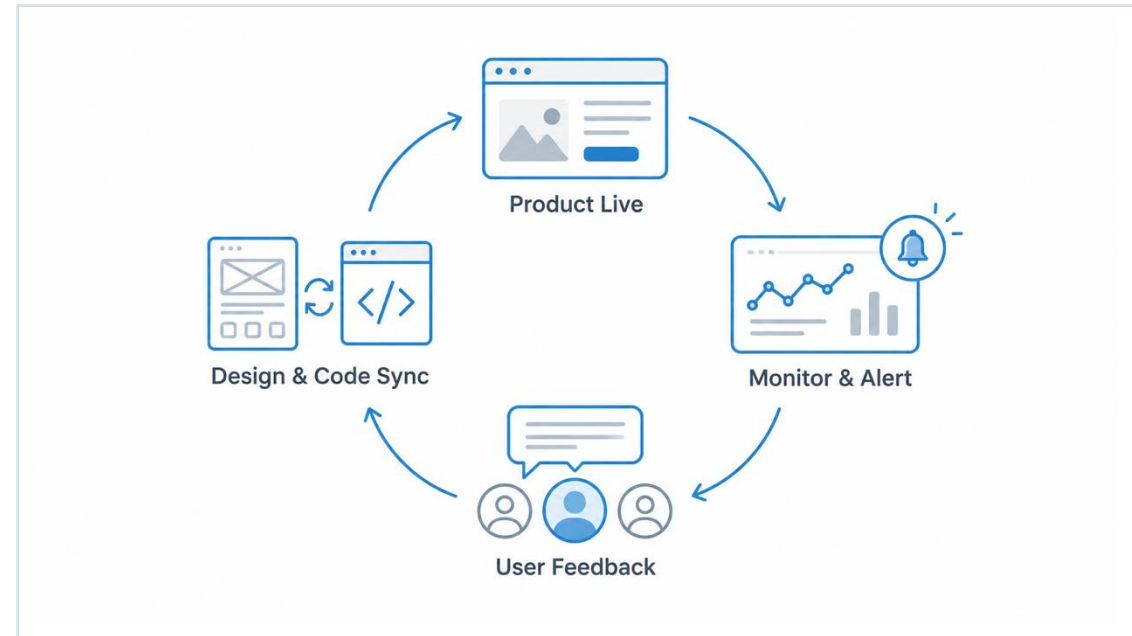
As soon as it is live, three questions come up:

Do you know it is broken before a user tells you?

Where does user feedback land, and who reads it?

How do Figma and the code stay aligned six months from now, when the next screen is designed and the code has moved on?

We don't have a good answer to the third one yet, and neither does anyone we've asked.



The post-launch loop

No role removed: the boundaries moved

What actually changed in the way we work

The designer goes further

It is not about designers writing code: the handover happens later, on something that works rather than on a description.

The engineer steps in earlier

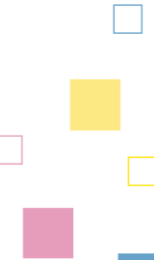
Involved from the exploration phase, when architecture mistakes cost nothing to fix.

The specification is alive

The specification is the clickable product itself: it cannot go out of date.

The role that grew the most isn't listed above: the reviewer.

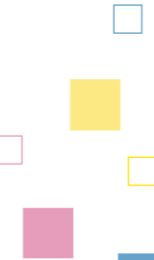
Someone has to decide what is reliable enough to go live. That responsibility has become both harder and more important at the same time.



Seven rules from our experience

What to take away from this session

- 1 Write the brief yourself: it is the one deliverable that is never generated.
- 2 Import the design system before the first prompt, not after a first disappointing result.
- 3 Validate the flow, not the pixels.
- 4 Export to Git, never to a zip.
- 5 Audit the existing code before generating a single line.
- 6 One intention per session, one intention per branch.
- 7 Read security, data access, money and personal data line by line.



Take-home messages

Move faster – Stay in control

Start with the problem, not with AI

A clear brief, real users and explicit constraints are what make AI efficient.

AI accelerates the development workflow

Prototype faster, iterate earlier, industrialize more efficiently - without replacing engineering discipline.

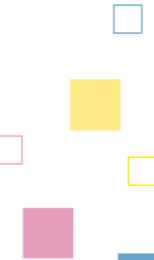
Keep domain expertise in the loop

In scientific and healthcare products, validation, security and compliance remain human responsibilities.

The real value lies in combining AI with human expertise

Design, software engineering, architecture and domain knowledge remain at the core of the process.

AI helps us move faster. Human expertise makes sure we move in the right direction.



Resources and contacts

Primary sources only



SOURCES

Documentation

Figma Make — Figma Help Center

GitHub export — Figma developer documentation

Claude — Anthropic documentation

Contact

Romain Cauhet — Marketing Engineer

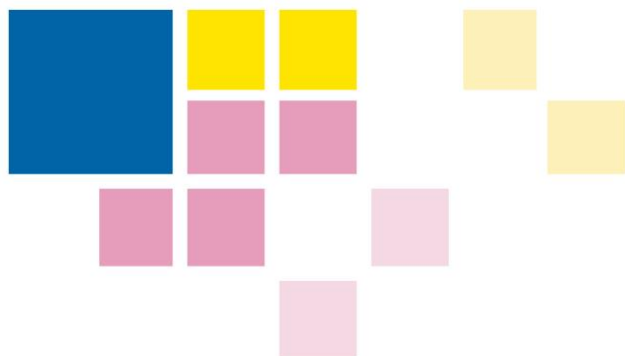
romain.cauhet@coexya.eu

Oussama Radouche — R&D Engineer

oussama.radouche@coexya.eu

AQUILAB by Coexya





QUESTIONS

Romain Cauhet · Oussama Radouche

AQUILAB by Coexya · 25 September 2026



THANK YOU

TETRIS - Risk assessment *Tools* for severe side *Effects* after *breasT* Radiotherapy:
radiation safety through biological extended models and *digital twinS*

EU Grant Agreement n. 101166699



Follow us on social media

