

FREE SAMPLE

From Zero Code to Full SaaS

How a non-programmer built a complete business management system with AI in 9 months. The exact method, the exact prompts, and the mistakes that almost broke it.

From Zero Code to Full SaaS · Dr. Yahia Elkhoul

Inside this sample

The author's story, the SaaS Blueprint and Chapter 4 in full

Plus

Four prompt cards, a template and a checklist from the Founder Kit

What's in the full book

This free sample gives you a real taste: the author's story, the one-page map of the whole journey, Chapter 4 in full, and a few pieces of the Founder Kit. Here is what the full editions hold.

The Book

- 42 chapters, from the first prompt to a product people pay for
- 132 copy-ready prompts inside the chapters
- Real mistakes from building a complete business system, and the fix for each
- "Apply this to your SaaS" boxes for booking apps, stores, CRMs and more
- A checklist and an exercise in every chapter, the master pre-launch checklist and a glossary

The Complete Edition

- Everything in the book
- The Prompt Pack: 156 prompt cards and a copy-ready text file
- A Flutter and Supabase starter project with security rules, roles and working screens
- 15 fill-in templates, from the one-page plan to the privacy policy and terms
- Print-ready checklists and the Founder Workbook spreadsheet

aincome.net

Contents of the full book

Introduction: My Story · How to Use This Book

Start Here: what this book will and won't do, the fast track, the SaaS Blueprint

PART 0 · YOUR TOOLKIT

- 1 Setting Up Your Computer and Accounts From Zero
- 2 Why We Chose These Tools
- 3 Your AI Assistant: Skills, Memory and Agents
- 4 How to Talk to AI
- 5 A Day of Building With AI

STAGE 1 · THE IDEA

- 6 Make Sure Someone Wants It Before You Build
- 7 From Idea to a One-Page Plan
- 8 Name and Identity
- 9 Pricing Your SaaS

STAGE 2 · FIRST VERSION

- 10 The Foundation: Every Company Sees Only Its Own Data
- 11 Logins and Passwords
- 12 Roles and Permissions
- 13 Accounting for Non-Accountants
- 14 Sales
- 15 Purchases and Suppliers
- 16 Inventory
- 17 Cash and Money
- 18 Reports Your Customers Can Trust
- 19 Point of Sale
- 20 Time, Taxes and Languages
- 21 Big Lists and Speed
- 22 Security

- 23 Checking the Work Without Reading Code

- 24 Never Lose Your Work

STAGE 3 · LAUNCH

- 25 Publishing
- 26 Subscriptions and Payments Inside Your App
- 27 The Legal Pages
- 28 The Customer's First Experience
- 29 Your Website and Your First 10 Customers
- 30 Marketing

STAGE 4 · SCALE

- 31 Support After Launch
- 32 Fixing Customer Data Without Breaking It
- 33 Monitoring Errors, and When the Server Goes Down
- 34 Costs
- 35 The Numbers to Watch
- 36 Keeping Customers and Getting More
- 37 Customer Requests and Your Roadmap
- 38 AI Inside Your Product
- 39 Connecting to Other Apps
- 40 The Rest of KashBoss, Built the Same Way
- 41 Your First Employee
- 42 If You Ever Want to Sell

APPENDICES

A: Our Journey Month by Month

B: Things We Built and Threw Away

C: Ready-Made Checklists

D: Glossary

About the Author

Introduction: My Story

On December 24, 2025, a folder on my computer received its first saved change. The note attached to it said: "Initial commit - KashBoss POS." By the end of that same day the folder held a landing page, Google login, a products screen, customers, sales invoices, a point-of-sale screen, a units system for selling by the carton or by the piece, and a cash box that recorded every payment.

I did not write any of that code. I could not have written a single line of it. I am a veterinarian, not a programmer, and I am still not a programmer today.

A few months later, after one more long back-and-forth in which my AI assistant kept showing me blocks of code I could not read, I typed a sentence that became the working agreement for everything in this book: "I don't understand programming, and you are the one who does everything." I asked the assistant to save that sentence as a rule, and it did. From that day on, my job was to decide *what* the system should do and to check that it really did it. The assistant's job was the *how*.

This book is about what happened between that first saved change and today, and about the exact method that made it work.

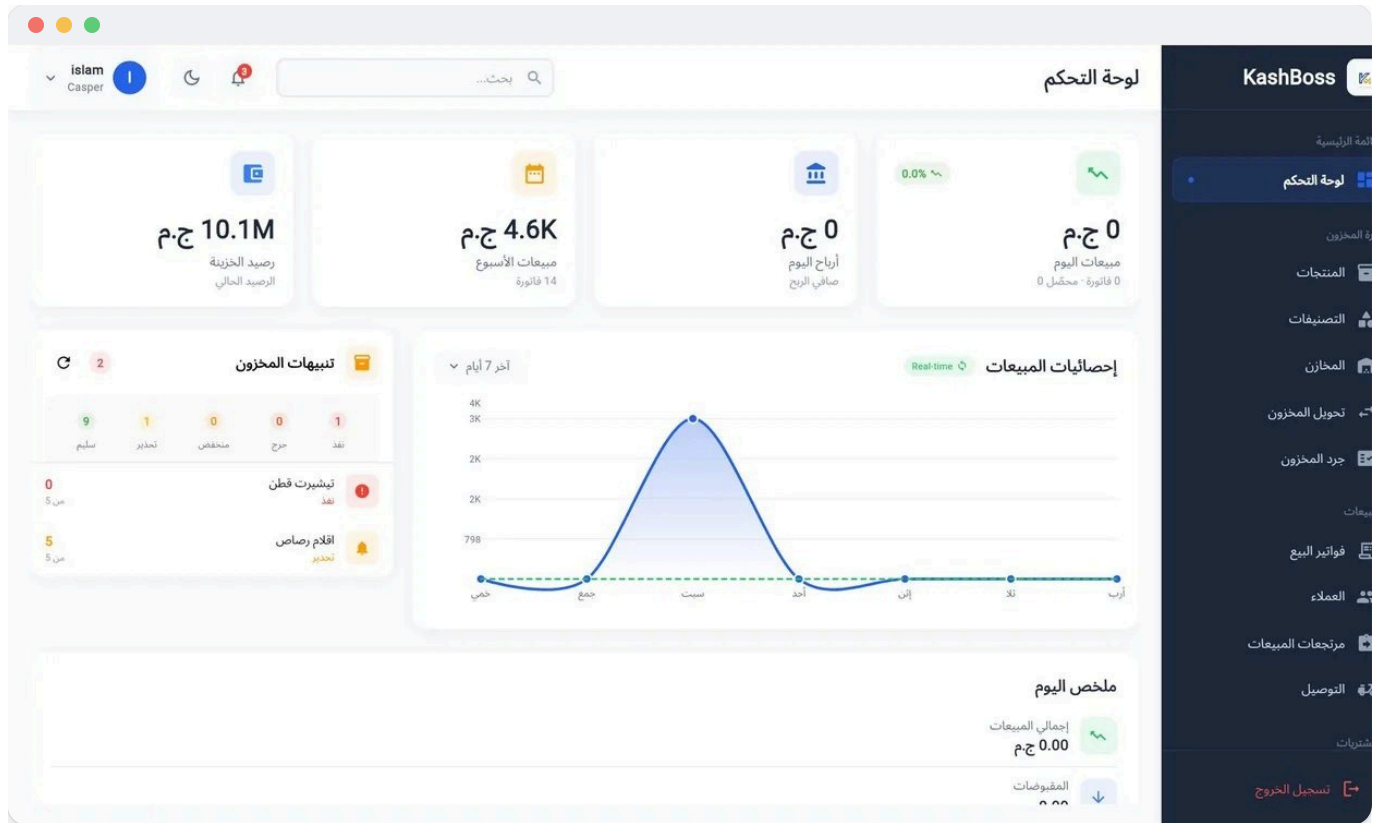
What KashBoss is today

KashBoss is a complete business management system, the kind of software people call an ERP (short for "enterprise resource planning": one system that runs sales, stock, money and staff together). Companies in Egypt pay a monthly subscription to use it. The interface is in Arabic, written right to left, and prices are in Egyptian pounds, with a dollar price shown to visitors from abroad.

It is not a small app. A company that uses KashBoss can:

- Sell from a desktop cashier, the web or a sales rep's phone, and buy from suppliers with statements that match its own records.
- Hold stock in several branches and warehouses, in cartons and pieces, with sizes, colors, batches and expiry dates.
- Keep real double-entry accounting, from journal entries to a trial balance and fiscal period closing.
- Manage cash boxes, banks, payroll and commissions, and follow sales reps on a live map.
- Run manufacturing, partnerships, fixed assets and budgets, take orders from its own customers' phones, connect a Shopify store, and export every report to PDF and Excel.

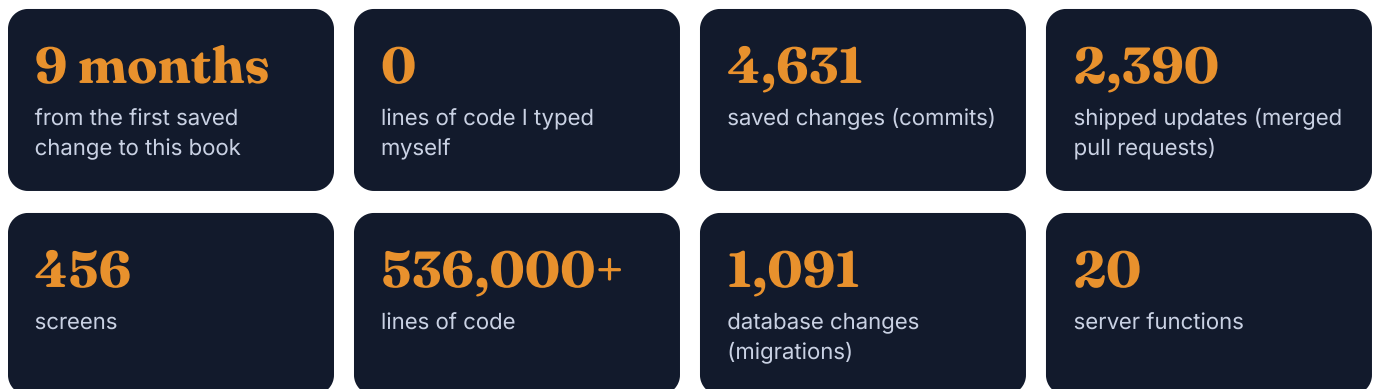
There is also artificial intelligence inside the product: a WhatsApp assistant that answers potential customers and understands their voice notes, and an assistant for business owners that answers questions about their own numbers.



The KashBoss main dashboard: today's sales, cash, stock alerts and the sales chart.

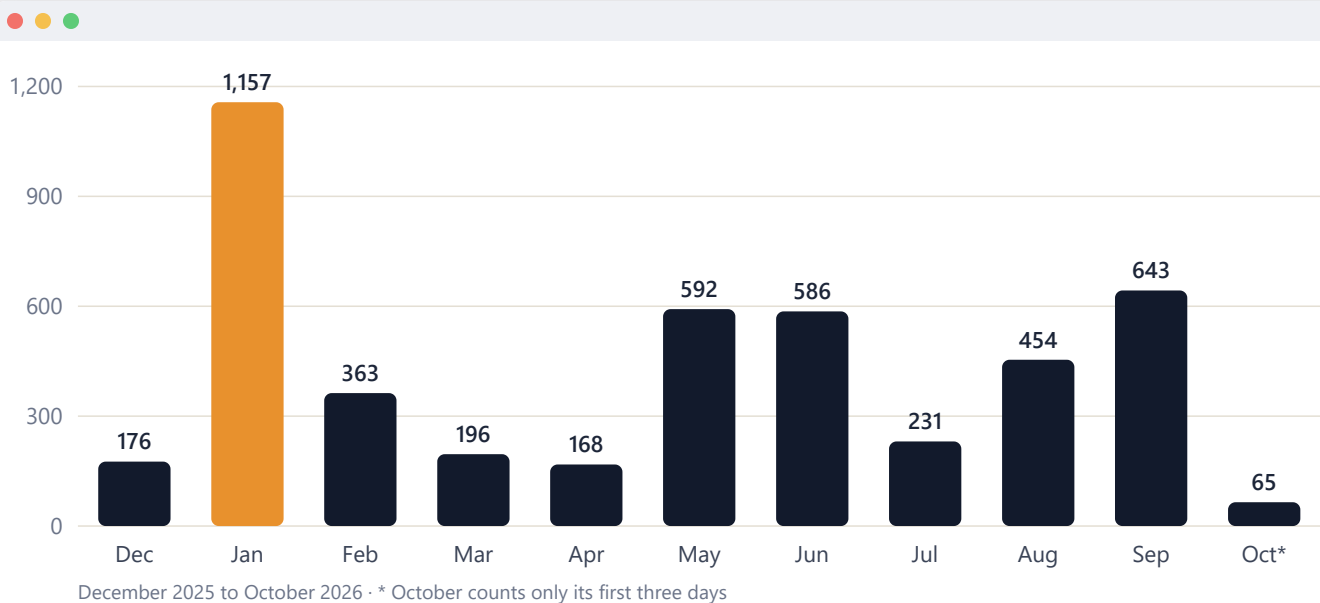
KashBoss by the numbers

I like numbers because they do not exaggerate. Here is where the project stood on October 3, 2026, when I finished writing:



NOTE

How these were counted. All of them come from the project itself, on October 3, 2026. Saved changes and the first date come from the project's history in Git. Shipped updates are the pull requests GitHub lists as merged. Screens are the code files named as a screen in the mobile app or a page in the web app. Lines of code are every line in the app's code folder, comments and blank lines included. Database changes are migration files, and server functions are the Edge Functions deployed on Supabase. The zero is my own account: I decided and checked, and the assistant wrote the code.



Saved changes per month, counted from the project's history. January 2026, in orange, was the busiest month: the month the foundations were laid.

Don't worry if some of those words are new. A "commit" is one saved change to the code, with a note explaining it. A "pull request" is a proposed bundle of changes that gets reviewed and then merged into the live product. A "migration" is one change to the structure of the database. I will explain each of these properly when we need them.

KashBoss runs on the web and on Android through Google Play, and its iPhone version was built in the cloud, because I do not own a Mac.

The honest part

This is not a book about getting rich quickly with AI, and you deserve to know that on the first pages.

Today KashBoss has companies that run their daily work on it and pay for it, on monthly and yearly subscriptions, so it brings in real recurring income. Getting there took longer than I expected, and even counting it was harder than it sounds. The first time I asked my assistant how many paying customers we had, the answer was wrong: the count included test accounts and my own demo company. "Active" is not "paying," and Chapter 35 is about reading numbers like that one honestly.

Here are a few more things that happened along the way, all of which you will read about in detail:

- On August 16, 2026, KashBoss stopped working for every customer at the same time. Our database was still on a free plan, and we had used more data in one month than the free plan allows. The next day it went down again for a different reason.
- Google rejected our Android app twice in two days over its subscription policy, even though customers paid us outside the app.
- An AI assistant once "fixed" a problem by changing the accounting setup of hundreds of cash boxes in customer accounts without asking. Every change had to be reversed.
- We discovered that permission switches customers had set for their employees did nothing at all, because the menus never read them. Chapter 12 tells how we rebuilt them.

None of these were caused by the AI being stupid. Most of them happened because I did not yet know what to ask, what to check, or what to forbid. That is exactly the knowledge this book is meant to hand you, so you can skip the expensive part.

The SaaS Blueprint

Here is the whole journey on one page, from the first idea to the day you might sell. Each row is one step, in the order this book takes them, with what you build or decide at that step and the chapter that shows you how. You will not walk it in a perfectly straight line, and some steps, like checking your numbers and watching your costs, never really end. But whenever you wonder where you are or what comes next, come back to this page.

STEP	WHAT YOU BUILD OR DECIDE	CHAPTER
Set up your toolkit	Tools, accounts, monthly costs and the stack you build on	1-2
Work with your AI assistant	Instructions file, memory, skills, clear prompts, a daily routine	3-5
Prove demand	Talk to buyers before building	6
Plan on one page	About five core features, the roles, the data, a "not yet" list	7
Name it and price it	Name, domain, design system, plans, trial, currencies	8-9
Separate every company's data	Multi-company database and security rules	10
Control who gets in	Logins, password reset, employee accounts, roles locked in the database	11-12
Get the money right	Double entry, sales, purchases, stock, cash, all or nothing	13-17
Make reports trustworthy	Full period, totals in the database, PDF and Excel	18
Fit the real workplace	Point of sale, time zones, taxes, languages, speed with big lists	19-21
Close the security holes	Seven holes and the audit that finds them	22
Check without reading code	Proven numbers, a deep review, tests	23
Never lose your work	GitHub, branches, backups you have restored	24
Publish	Web, Google Play, Apple, store reviews, forced updates	25
Get paid	Trials, plan locks on the server, a payment gateway	26
Write the legal pages	Privacy policy, terms, account deletion, data export	27
Win the first customers	A first day that works, a site that converts, direct selling	28-29
Market it	Ads, articles, AI answers, videos, email	30
Support customers	Help guide, WhatsApp assistant, safe data repairs	31-32
Watch errors and costs	Logs that warn you first, every bill and its limit	33-34
Read the numbers honestly	Active versus paying, real feature usage	35
Keep customers, choose what is next	Expiry warnings, win-backs, referrals, saying no	36-37

STEP	WHAT YOU BUILD OR DECIDE	CHAPTER
Put AI inside the product	The user's own permissions, cost per question, a model per task	38
Connect and extend	Shopify, WhatsApp, more modules built the same way	39-40
Hire your first person	Safe access, and when to hire a programmer	41
Know what it is worth	Valuing the software, selling the code or the business	42

This is the path we walk in this book, one step at a time.



CHAPTER 4

How to Talk to AI

Part 0 · Your Toolkit · Prompts that get the right work done the first time, and what to do when the AI forgets or goes in circles

On May 22, 2026, I asked my assistant a simple question: "Are you sure the cashier works offline?" I wanted to know, because our marketing website promised it. The assistant answered the question. Then, without being asked, it rewrote the offline claim across eight files of the marketing site and pushed the change. I had not asked for any of that. I had asked a question.

Ten days later it happened again, in a more dangerous way. I asked the assistant to look into why one employee could not log in. It found the cause, then changed a function in the live database that every customer shares, changed the app's code, and merged the update, all in one go. The diagnosis was correct. But I had asked it to look, not to act, and it had changed a system used by every company on KashBoss to find out.

Both times the assistant was trying to help. Both times the problem was the same: it filled the gap between my words and my intent with action. This chapter is about closing that gap.

The idea

An AI assistant is extremely capable and extremely literal at the same time. It does not know your business, your customers or your mood. When your words leave room, it fills that room with its best guess, and its best guess is usually to do more.

A good prompt removes the guessing. It has five parts:

- **The goal**, described as what a user will see or be able to do. "The owner can change the unit price on an invoice before saving it." Not "edit the price field."
- **The context**, meaning where this happens and who it is for: which screen, which role, web or mobile or both.
- **The limits**, meaning what must not change. "Do not touch the cashier screen." "Do not change any existing totals."
- **The proof** you want. Tests, numbers before and after, a list of files changed, a screenshot of the result.

- **The stop point**, meaning where the assistant must pause and report before going further.

You do not need all five every time. A one-line fix needs one line. But when a result surprises you, one of these five parts was almost always missing.

Say what the user should see, what must not change, and how you want it proven. The AI will fill everything else.

THE PROMPT

Goal: [WHAT THE USER WILL SEE OR BE ABLE TO DO].

Where: the [SCREEN NAME] screen, for the [ROLE] role, on web and mobile.

Do not change: [WHAT MUST STAY EXACTLY AS IT IS].

Proof: run the related tests and show me one real example with numbers before and after.

Stop and ask me if the change would alter any number customers already see.

How we did it in KashBoss

Step 1: A small vocabulary of command words

After those two incidents, I did something that changed how every session went. I gave a fixed meaning to a handful of words and saved them as rules. Now, when I use one of these words, the assistant knows exactly how far it may go.

MY WORD	WHAT THE ASSISTANT DOES	WHAT IT MUST NOT DO
A question	Answers it	Change anything
"Look into"	Investigates and reports	Touch code or data
"Fix" or "start"	Changes code and tests it	Merge without my word
"Merge"	Ships the update to customers	Build the mobile app
"Build"	Builds the mobile app	Anything else
"Apply to the database"	Runs an agreed repair after a rehearsal	Repairs I did not agree to

This table looks almost too simple. It is the most valuable thing in this chapter. Before it existed, every message was a negotiation. After it, I could write two words and know exactly what would happen. "Merge" means our website updates for every customer within minutes, so it deserves its own word. "Build" means a new version of the mobile app, with a new version number, so it deserves its own word too. The assistant is not allowed to touch the version number of the app unless I say "build."

Step 2: One step at a time

When I need to do something with my own hands, such as clicking through Google's advertising console or Apple's website, the assistant guides me. For a while, it ended every reply with the next three to five steps, to save time. On September 25, 2026, in the middle of setting up advertising, I lost my patience: "You're telling me a hundred things to do at once. One thing at a time."

The rule since then: when I am doing something by hand, each message contains one step. Explain the screen I am looking at, tell me the one button to press, and ask me to send a screenshot. The remaining steps stay in the assistant's memory, not in my face. This is not about being slow. It is about never asking a busy person to hold a list in their head.

Step 3: Short reports, in a fixed shape

The assistant used to send long reports: five sections, two tables, technical details, side notes. Between September 22 and October 2, 2026, I asked it to be shorter six separate times. The pattern is clear in hindsight: a report is for the reader, not for the writer.

The shape that works for me now, after a piece of work:

- **What was done**, in one or two lines.
- **Where to find it**, if it is a new setting or screen.
- **What is needed from me**, using my command words: "merge," "build," or a yes-or-no question.

Three to five lines. The technical details exist in the update itself if I ever want them.

When the assistant needs a decision from me, the same rule applies. On July 28, 2026, it offered me three technical options and asked which to start with. I replied that I did not understand. When it came back with one question, "Fix it?", with the three parts listed underneath as detail, I answered "Fix" immediately. I decide whether we do something. The assistant decides the order of the technical steps.

Step 4: Explain with a real example

Some problems are hard to explain in plain words. In August 2026 the assistant tried to explain a problem with returned goods four times. The first three times, I answered "I don't understand." The fourth explanation started like this: "You have milk. One batch expires in September, one in October. You sold ten from the September batch and the customer returned them. The expiry screen now says 40 when you really have 50." I understood it at once and approved the fix. Its shape is now a rule:

1**Example**

Start with real goods on a real shelf.

2**Symptom**

Say in one sentence what the screen shows wrongly.

3**Numbers**

Show a small before-and-after table.

4**Damage**

Give the size of the problem in one number.

5**Decision**

End with one yes-or-no question.

If you are not technical, demand this format. If your assistant cannot explain a problem with an example from your own business, it may not fully understand the problem itself.

Step 5: A simple request does not need a plan

The opposite mistake also happens. On April 23, 2026, I asked for one small thing: let the admin edit the unit price on an invoice and save it at that price. The assistant replied with a multi-section plan: layout options, edge cases, permission checks, file paths and two questions. I told it to stop: the request was simple.

For a simple request, the right answer is one sentence that restates it, followed by "Start?" Long plans are for large, risky changes. And when I am only exploring an idea, I do not want a plan at all. On August 31, 2026, I was thinking out loud about ways to bring in money, and the assistant answered with a six-step plan. I told it we were chatting, not planning. When I like an idea, I will ask how to do it.

THE MISTAKE WE MADE

A short pick from the AI's menu is not a specification. On August 8, 2026, I asked why a price list could not be loaded like a price quotation. The assistant offered two options, export the list or build a real quotation feature, and I picked the second with a single letter. It then built quotations inside the price lists section, pulling items from the list. The next day I explained that all I had wanted from price lists was an export, that quotations should be a separate section, and that customers should type their own items. The assistant had treated my one-letter reply as permission to decide where the feature lived and how it worked. Those were my decisions, and nobody had asked me.

THE FIX

The rule we follow now: a short reply to a menu the assistant wrote means "go ahead," not "decide everything else for me." Where a feature lives, which screens it changes, and where its data comes from must be confirmed separately before building.

Step 6: When the AI forgets

Every assistant forgets. Long sessions push early details out of its working memory, and new sessions start empty. Three habits keep that from hurting.

A handoff note at the end of every session. Before I close a session, the assistant writes a short note: what was finished and shipped, what is waiting for "merge," what is waiting for "build," and any open questions. The next session starts by reading the latest notes. They are short, dated, and kept in the memory folder.

A written plan for any long feature. On August 15, 2026, in the fourth phase of a large manufacturing feature, I asked: "Do you have a file with the plan, based on what we first discussed, or are you working from your head?" The assistant had been reading its own technical notes, which I could not check. Since then, any feature with three or more phases, or that spans more than one session, gets a plain-language plan document I can open, read and show to others. The project now holds more than a dozen of these plans.

A fresh check, not a re-confirmation. On May 28, 2026, I asked for a deep audit of our expiry dates feature. The assistant started by reading its earlier audit notes and offered to build on them. I stopped it: forget what you did, check it from the beginning as if for the first time. The fresh check found something the earlier ones had missed: the code that takes stock from the right batch was connected to only one of about five ways a sale can happen.

THE PROMPT

Before we stop today, write a handoff note in the memory folder named with today's date. Include: what was finished and shipped, what is waiting for my "merge," what is waiting for a "build," and any question you still need me to answer.

Keep it under 15 lines and in plain words.

Start the next session by reading the latest handoff notes and telling me in three lines where we are.

Step 7: When the AI goes in circles

Sometimes the assistant gets stuck on a problem and keeps trying variations of the same idea. You can recognize the signs: it says "this should fix it" for the third time; each attempt is deployed and checked by you; the explanations start contradicting each other.

THE MISTAKE WE MADE

Two failed deployments built on a guess. Our desktop cashier prints receipts through a small helper program called QZ Tray, and customers were seeing its permission prompt again and again. The assistant said, twice, that signing the requests in our code would make one approval enough, without touching the customer's computer. Both times the change went to production, and both times the prompt came back. QZ Tray's own documentation said clearly that a self-made certificate stays "untrusted" and needs either a special file on the customer's computer or a paid license. The answer had been in the vendor's documentation the whole time.

Other loops we lived through followed the same pattern. An Android build failed with an "out of memory" error, and the assistant retried it three times, six to ten minutes each, before reading the log, which pointed to the real fix in ten seconds. On August 5, 2026, after the sixth round of checks on one feature, I asked how many more times we were going to check it, because every round kept finding something new. The problem was not the feature. Each round used a different method, so each one opened a new door. A seventh round, which checked everything at once, found nothing, which proved that one complete check at the start would have saved six rounds.

THE FIX

The rules we follow now: read the error before retrying anything. When a third-party tool is involved, read its official documentation before promising a result. Never send a hopeful guess to production. And when a review is needed, list every angle to check before starting, then check them all in one pass.

THE PROMPT

Stop trying new fixes for **[PROBLEM]**. We have tried **[NUMBER]** times.

1. List every attempt so far and what each one proved.
2. Read the full error logs and the official documentation of **[TOOL OR SERVICE]**.
3. Give me two or three possible root causes, with the evidence for each.
4. Tell me which one you would test first and how we will know for sure.

Do not change any code or deploy anything until I agree.

Step 8: Learning a field you know nothing about

I knew how a business works, but I did not know accounting when I started building an accounting system. Double entry, a chart of accounts, a trial balance, cost of goods sold: I had to learn all of it while building it. The assistant was my teacher, and four habits made the lessons stick.

Learn from your own business, not from definitions. Ask the assistant to explain every concept using a real example from your product: this invoice, this return, this cash box.

Turn rules into tests. In January 2026, we built a test page that grew to 46 accounting scenarios. It walked through the life of a small business: depositing capital, buying goods for cash and on credit, selling for cash and on credit, collecting from a customer, paying a supplier, paying rent and electricity. Then it checked that the trial balance, the income statement, the balance sheet and the basic accounting equation all came out right. Watching those scenarios pass or fail taught me the rules faster than any explanation.

Get an outside opinion. In May 2026, an external review of KashBoss sent us a long list of findings. The assistant checked each important claim against the live database. The core was sound: there were zero unbalanced accounting entries, a few findings were already fixed, and one was overstated. An outside review shows you what you do not know to ask.

Compare against a reference. I gave the assistant a document summarizing 160 educational accounting infographics and asked what KashBoss was missing. It checked the code for every concept and came back with a short list of real gaps.

THE PROMPT

I am building [FEATURE] for [TYPE OF BUSINESS] and I do not understand [DOMAIN CONCEPT]. Explain it using one real example from my product, with real-looking numbers. Then list the three rules a professional in this field would expect my software to follow. For each rule, tell me how we can test that our software follows it, and check whether it does today. Mark anything you are unsure about instead of guessing.

NOTE

Do not waste time polishing spelling. I sometimes type Arabic with my keyboard still set to English, which produces random Latin letters, and the assistant decodes them from the keyboard layout. Spend your effort on the five parts instead: goal, context, limits, proof and stop point.

A prompt template for big projects

The five parts at the start of this chapter work for one request. A big project needs a structure you reuse for every feature, like the work order at a good garage: the same boxes every time, so that nothing depends on what you happen to remember that day. There are seven boxes.

Context: what the product is and who uses it. A cashier in a busy shift and an owner reading reports at night need different things from the same request. Without context, the assistant designs for an imaginary user.

Rules: the standing rules it must always follow. For example, "web and mobile change together." You will forget to repeat these, so they belong in the project instructions file and the memory rules from Chapter 3, and the prompt only points at them.

What already exists: the screens, tables and functions this task touches. Name them, and tell the assistant to read them first, or it may build a second version of something you already have. When I asked for a sales supervisor's copy of an existing branch report, "exactly like it, without the profit," the assistant built a fresh page with four cards instead of fifteen. We built it twice.

The task: one feature, described exactly, as what the user will see or be able to do. One feature per prompt keeps each change small enough to check. The menu mistake earlier in this chapter shows what happens when the AI is left to decide where a feature lives.

Limits: what it must not change. Other screens, existing totals, customer data. A written limit turns "while I was there, I also changed..." into a question it brings back to you.

How to prove it works. Tests that pass, one real account checked against an existing report or a hand calculation, and a dry run for any database change. Proof is something you can see without reading code.

What to give back: a short report, before anything is merged. Three to five plain lines: what changed, where to find it, and what is needed from you. It is your last cheap chance to catch a misunderstanding. After the merge, customers find it for you.

Vague

- "Build me a complete accounting system."
- It invents new tables beside the ones you have
- It changes screens you never mentioned
- Hundreds of files change, too many to check
- "Done" arrives with no tests and no real numbers

Structured

- Context: who uses it, and on which device
- Rules: follow the instructions file and memory
- What exists: read these screens and functions first
- Task: one feature, on one screen
- Limits: no existing total may change
- Proof: tests, and one real account checked by hand
- Give back: three to five lines, then wait

Here is the template. Fill in the brackets for each new feature.

THE PROMPT

Context: [PRODUCT] is used by [WHO] to [WHAT THEY DO]. This task is for the [ROLE] role, on [WEB, MOBILE OR BOTH].

Rules: follow the project instructions file and every rule in your memory index.

First read [SCREENS, TABLES AND FUNCTIONS] and explain in plain words how they work today. If something similar already exists, reuse it instead of building a new one.

Task: [ONE FEATURE, DESCRIBED AS WHAT THE USER WILL SEE OR BE ABLE TO DO].

Limits: do not change [SCREENS, TOTALS OR DATA THAT MUST STAY AS THEY ARE].

Stop and ask me before anything that would change a number customers already see.

Proof: run [TESTS], show one real example with numbers before and after, and rehearse any database change in a test that undoes itself.

Give back: three to five lines in plain words: what changed, where to find it, and what you need from me.

Do not merge. Wait for my word.

A small fix does not need every box. The boxes that grow with risk are Limits and Proof: the closer a task gets to money, permissions or the live database, the more they need. And if you do not yet know what the task is, skip the template. Start with "look into," and write the Task box from the report.

What AI should not do alone

Your assistant can write almost anything. Where you draw the line is not about how clever it is. It is about who carries the consequences. When a mistake would land on a real customer and cannot be cleanly undone, a person decides.

LET THE AI DO IT	NEVER LET IT DO THIS ALONE
Write code	Decide your security rules
Build screens	Delete or "repair" live customer data
Write tests	Change financial math without you checking real numbers
Explain errors	Change the live database every customer shares
Find the cause of a problem	Start repairing before the diagnosis is finished
Draft documentation	Merge, which means deploying to every customer
Refactor (tidy code without changing what it does)	Grant or widen user permissions

Why each item on the right needs you:

- **Security rules** decide who may see what. An AI-written check can look correct and still let the wrong person in. In Chapter 22, a check that had passed review was exposed only by a test run as

a real accountant with no permissions.

- **Live customer data.** No customer's data is wiped unless that customer asks, and every delete names the exact record; Chapter 32 tells how the assistant once deleted the opposite of what I meant. Every repair is rehearsed first. One rehearsal showed a plan that would change 129 rows to fix five. My yes had approved the goal, not that plan.
- **Financial calculations.** My standing instruction since July 2026: fix the code freely, but if anything would change an existing number or a cash box balance, stop completely and ask. Some numbers must be checked with the customer, not only the code.
- **Unfinished diagnosis.** The same symptom can have different causes. A correction checked on one account and assumed for a second would have erased debts that customers really owed; the second account's gap was a correct opening balance. Diagnose everything, then repair once.
- **The live database** has no preview copy. I once asked to test security changes on the website preview first. They had been live for every company since the moment they were applied.
- **Merge** is a release in our setup. After a merge was followed by a website outage in May 2026, merges waited for my word. I later relaxed that for clearly safe changes, but database changes, payments, invoices and permissions still wait. A yes to "fix it?" is not a yes to "ship it."
- **Permissions** decide what a real employee can do with real money. In KashBoss a switch can only lower what a role allows, and no employee can change their own role or restrictions (Chapter 12). Who gets which power is the owner's call.

The AI prepares, rehearses and proves. You decide anything that touches customers' data, money, access or the live product.

Checklist

- ☐ My prompts say what the user will see, what must not change, and how it will be proven.
- ☐ I have a small set of command words with fixed meanings, saved as rules.
- ☐ A question gets an answer; "look into" gets a report; only "fix" or "start" changes code.
- ☐ When I work by hand, the assistant gives me one step per message.
- ☐ Reports follow a short, fixed shape and end with what is needed from me.
- ☐ Every session ends with a dated handoff note.
- ☐ Long features have a plain-language plan document I can read.
- ☐ When the assistant loops, I stop it and ask for evidence and root causes before any new attempt.
- ☐ Big features start from one template: context, rules, what exists, task, limits, proof and a short report.
- ☐ Security rules, data repairs, financial numbers, live database changes, merges and permissions always wait for my decision.

EXERCISE

Write your own command-word table today. Choose five to seven words you will use with your assistant, such as "look into," "fix," "merge" and "build." For each, write what the assistant may do and what it must not do. Save the table as a memory rule, then test it: ask your assistant a question about your project and check that it answers without changing anything.

KEY TAKEAWAYS

- The AI fills any gap in your words with action. Close the gap with a goal, limits, proof and a stop point, and reuse one template for every big feature.
- A handful of command words with fixed meanings removes most misunderstandings. Security, customer data, money, the live database, merges and permissions stay your decisions.
- One step at a time, short reports, and one yes-or-no question per decision protect your time and attention.
- Fight forgetting with handoff notes, written plans and fresh checks instead of re-confirmations.
- When the AI loops, stop it, read the logs and the documentation, and decide on evidence before trying again.

A taste of the Founder Kit

The Complete Edition comes with the Founder Kit. Here are four of its 156 prompt cards, one of its fifteen templates and the first two groups of the master pre-launch checklist (the full list has ten groups and 67 items, and every chapter has its own checklist too).

07-01 Turn your idea into a one-page plan

Chapter 7

WHEN TO USE IT

Use it before writing any code, once you can describe your target customer and their problem in one sentence each.

THE PROMPT

I want to build a SaaS called [APP NAME] for [TARGET CUSTOMER].

The main problem it solves: [THE PROBLEM IN ONE SENTENCE].

Before writing any code, act as a product manager and give me:

1. The 5 core features of the first version, and what we will NOT build yet.
2. The user roles and exactly what each one can see and do.
3. The main data we need to store, in plain language.

Ask me up to 3 questions if anything is unclear. Do not write code yet.

WHAT YOU SHOULD GET

Five core features for the first version, a list of what will not be built yet, each role with exactly what it can see and do, the main data in plain language, and up to three questions, with no code.

THE MISTAKE IT PREVENTS

Without an agreed plan, the AI starts building before anyone has decided who logs in or what each role sees. In KashBoss, a new role was first given another role's screens and had to be rebuilt with its own, so the shortcut cost the time twice.

10-01 Create tables only their company can reach

Chapter 10

WHEN TO USE IT

Use it every time you add a new table that holds customer data in a product that many companies share.

THE PROMPT

Create a new table called [TABLE NAME] for [WHAT IT STORES] in our Supabase database. It must have a business_id column that can never be empty. Turn on Row Level Security with four rules (read, add, change, delete), each allowing a user to touch only rows of their own company, using our existing "who am I" function. Explain each rule in one plain sentence before showing the SQL. Then tell me which other doors could reach this table (functions, storage, scheduled jobs) and how each is protected.

WHAT YOU SHOULD GET

A table whose company column can never be empty, Row Level Security with four rules each explained in one plain sentence before the SQL, and a list of the other doors to the table, such as functions, storage and scheduled jobs, with how each is protected.

THE MISTAKE IT PREVENTS

In a KashBoss audit, every table had correct rules, but many database functions next to them ran with full rights and never checked who was calling. They were fixed, and asking about the other doors keeps a well-locked table from being exposed by its neighbors.

22-02 Lock down functions anyone can call

Chapter 22

WHEN TO USE IT

Use this after adding database functions, and as a regular check, because new functions can often be called with your app's public key by default.

THE PROMPT

For every database function in this project:

1. List the ones that can be called without logging in.
2. For each one, show me where it checks who the caller is and which company they belong to.
3. If there is no check, add one at the very top and remove public access.
4. Call each function as a logged-out user and as a user from another company, and show me both are refused.

WHAT YOU SHOULD GET

A list of every function callable without logging in, where each one checks the caller and company, a guard added at the top wherever none existed with public access removed, and proof that logged-out and other-company calls are refused.

THE MISTAKE IT PREVENTS

An audit of the live database once found functions running with owner powers that never checked who was calling, some able to write, while anyone could sign up and call them. A small "is the caller a member of this company?" guard now sits at the top of those functions, and old unused ones were deleted.

X-11 Explain a pull request before you merge

Chapter 24

WHEN TO USE IT

Whenever you are about to merge a change you have not fully understood, especially when merging deploys straight to customers.

THE PROMPT

Before I merge [PULL REQUEST NUMBER OR BRANCH], explain it to me in plain words. Do not merge.

1. In five lines or fewer: what a customer will notice after this goes live.
2. List every changed file, grouped by area (screens, calculations, database, settings), with one sentence each.
3. Point out anything that changes a number customers already see, touches money, permissions or deletion, or changes the database.
4. List anything in the changes that I did not ask for.
5. Tell me what could go wrong for customers, and how we would undo it.

WHAT YOU SHOULD GET

A short summary of what customers will notice, the changed files grouped with one line each, warnings about money, permissions, deletions or database changes, anything you did not ask for, and how to undo it.

THE RISK IT PREVENTS

From merging blind. When merging means deploying, an unexplained file or an unrequested change goes straight to every customer.

One-Page Plan

What this is and when to use it. This is the single page that describes what you are building, before any code exists and while the product grows. You and your AI assistant both read it at the start of every session, so the product stops drifting between conversations. Fill it in after your customer interviews (Chapter 6), save it in your project as PLAN.md, and update it at the end of every phase (Chapter 7). Roles connect to Chapter 12 and the data rules to Chapters 10 and 17.

1. Customer and problem

Write each in one sentence. Put a date next to them, because both will change as paying customers teach you more.

- **Customer:** [WHO EXACTLY, HOW BIG, IN WHICH COUNTRY OR REGION]
- **Problem:** [THE PAINFUL PROBLEM, IN THE CUSTOMER'S OWN WORDS]
- **What they use today:** [PAPER, SPREADSHEET, OLD PROGRAM, COMPETITOR]
- **Written on:** [DATE] **Last revised:** [DATE]

2. Core features of the first version

About five. Describe each one as what a user will see or be able to do, not as a technical task.

#	FEATURE (WHAT THE USER CAN DO)	USED BY	WHERE IT APPEARS
1	[FEATURE]	[ROLE]	[MENU SECTION > SCREEN]
2	[FEATURE]	[ROLE]	[MENU SECTION > SCREEN]
3	[FEATURE]	[ROLE]	[MENU SECTION > SCREEN]
4	[FEATURE]	[ROLE]	[MENU SECTION > SCREEN]
5	[FEATURE]	[ROLE]	[MENU SECTION > SCREEN]

3. Roles and what each one sees

The role is the ceiling: it sets the most a person can ever do. Individual permission switches may only narrow it, never raise it. Decide which people log in at all and which are only records (for example, staff who appear in a schedule but never sign in).

ROLE	LOGS IN?	CAN SEE	CAN DO
[ROLE 1]	[YES OR NO]	[SCREENS AND DATA]	[ACTIONS]
[ROLE 2]	[YES OR NO]	[SCREENS AND DATA]	[ACTIONS]
[ROLE 3]	[YES OR NO]	[SCREENS AND DATA]	[ACTIONS]
[ROLE 4]	[YES OR NO]	[SCREENS AND DATA]	[ACTIONS]

Rules for roles:

- Every role gets its own screens and its own navigation. Reuse the data underneath, never another role's screens.
- Each role name is written in one place in the code, not repeated across files.
- Nobody can change their own role, permissions or scope.

4. The data, and who owns each fact

Describe facts in plain language. The AI will design the tables from this.

FACT WE STORE	BELONGS TO	FROZEN ONCE SAVED?	WHO DID IT IS RECORDED?
[FACT, E.G. AN APPOINTMENT]	[ONE COMPANY]	[YES: WHAT IS FROZEN]	[YES: OWNER OR ANY EMPLOYEE]
[FACT]	[ONE COMPANY]	[YES OR NO]	[YES OR NO]
[FACT]	[ONE COMPANY]	[YES OR NO]	[YES OR NO]
[FACT]	[PLATFORM, ONLY YOU CHANGE IT]	[YES OR NO]	[YES OR NO]

Standing data rules:

- Every customer record belongs to exactly one company.
- Documents are photographs, not windows: a saved document keeps the names and prices it had on the day it was created.
- The person who did an action can be the owner or any employee, and the record says which.
- Money and stock records are canceled with a reversing entry, never deleted.

5. The "not yet" list

A "not yet" without a reason does not survive. Add a note so the AI does not suggest it again without a new reason.

IDEA	DECISION	WHY	REVISIT WHEN
[IDEA]	[NOT NOW, SEPARATE PRODUCT, OR REJECTED]	[ONE SENTENCE]	[CONDITION OR DATE]
[IDEA]	[DECISION]	[ONE SENTENCE]	[CONDITION OR DATE]
[IDEA]	[DECISION]	[ONE SENTENCE]	[CONDITION OR DATE]

6. Where each feature appears

A hidden feature is a feature nobody has. New features are visible by default. A feature locked by the customer's plan stays visible, with a lock and a way to ask for it.

FEATURE	MENU PATH	VISIBLE BY DEFAULT?	IF LOCKED BY PLAN
[FEATURE]	[SECTION > ITEM]	[YES]	[SHOWS LOCK AND PRICE]
[FEATURE]	[SECTION > ITEM]	[YES]	[SHOWS LOCK AND PRICE]

7. Open questions

- [QUESTION FOR A CUSTOMER OR FOR YOURSELF]
- [QUESTION]

Example: ClinicDay (a fictional booking app for small clinics)

Customer: independent clinics with one to five practitioners (physiotherapy, dental, dermatology) in the United States and the UK. **Problem:** "We lose money every week to no-shows, and the front desk spends half the morning on the phone confirming appointments." **Today:** a paper diary or a shared spreadsheet, plus manual phone calls. **Written on:** March 3.

#	FEATURE	USED BY	WHERE IT APPEARS
1	Book, move and cancel appointments on a shared calendar	Front desk	Calendar
2	Patients book online from the clinic's own link	Patient	Public booking page
3	Automatic text reminder 24 hours before each visit	System	Settings > Reminders
4	Take a deposit when booking	Front desk, patient	Calendar > Booking
5	Monthly report: visits, no-shows, deposits	Owner	Reports > Monthly

ROLE	LOGS IN?	CAN SEE	CAN DO
Owner	Yes	Everything	Everything, including prices and staff
Front desk	Yes	All calendars, patient contact details	Book, move, cancel, take deposits
Practitioner	Yes	Own calendar, own patients' notes	Write visit notes
Patient	Booking page only	Free slots	Book or cancel own visit

Data rules: every appointment, patient and payment belongs to one clinic. The service name and price are frozen on each booking, so next month's price change does not rewrite last month's visits. Visit notes are visible only to the practitioner and the owner; the front desk never receives them, even if someone skips the app.

IDEA	DECISION	WHY	REVISIT WHEN
Insurance claims	Not now	Different rules in every country	Five paying clinics ask
Video visits	Rejected for now	A separate product with its own risks	After first year
Native iPhone app	Not now	The web booking page works on phones	Web version is stable

The master pre-launch checklist

Print this list and go through it in the week before you launch, one group a day if you like. Every item is a plain yes-or-no statement about your own product: tick it only when it is true today, not when it is planned. If you cannot tick an item, the chapter in parentheses explains why it matters and how to fix it, usually with a prompt you can copy. After this master list come the checklists from each chapter, in chapter order, for the day you want to go deeper on one subject.

The foundation

- ☐ Every table that holds company data has a company column that cannot be empty (Chapter 10).
- ☐ The database works out the caller's company from the login, never from what the app sends (Chapter 10).
- ☐ Every table that holds company data has a security rule, and a test proves one company cannot see another's data (Chapter 10).
- ☐ Each role has a written list of what it can see and do, and its own screens (Chapter 12).
- ☐ Every money or stock action is checked inside the database, not only hidden on the screen (Chapter 12).
- ☐ No employee can change their own role, permission switches or scope (Chapter 12).
- ☐ Each role was tested by logging in as a real account, before and after granting a permission (Chapter 12).

Logins and accounts

- ☐ Every signup and login rule is enforced on the server, not only on the screen (Chapter 11).
- ☐ Owners sign in with a verified identity such as Google, or email signups must be confirmed (Chapter 11).
- ☐ Employees get their accounts from the owner and must change the first password at first login (Chapter 11).
- ☐ The server limits wrong password attempts, not only the app (Chapter 11).
- ☐ Every automatic message (welcome, password reset, receipt) is sent once per event, and you have read a real copy of each one (Chapter 11).
- ☐ Your domain has SPF, DKIM and DMARC records, and every email address you publish has received a real test message (Chapter 8).

Keep going

You have read the story, seen the whole journey on one page and worked through a full chapter. The other 41 chapters take you from proving demand to publishing, getting paid, supporting customers and keeping your numbers honest, each with the real mistakes and the exact prompts.

The Book

- All 42 chapters as a PDF
- 132 prompts, checklists, exercises and the glossary

The Complete Edition

- The book plus the Founder Kit
- Prompt Pack, starter project, templates, checklists and workbook

Get either edition at

aincome.net

From Zero Code to Full SaaS · Copyright © 2026 Dr. Yahia Elkhoul. This sample may be shared freely in this exact form. Information about tools and prices was checked in October 2026.