A LAYMAN'S FIELD GUIDE

From Idea to Live App

How a total beginner can use VibeKit, an AI coding agent, GitHub and Vercel to turn a napkin idea into a real, working website — step by step, with no coding background required.

VibeKit → Claude Code / OpenCode → GitHub → Vercel
Next.js · Neon · Prisma · Better Auth · Resend

Written for a curious beginner • Worked example: "Chores Champ"

CONTENTS

Table of Contents

CHAPTER 1

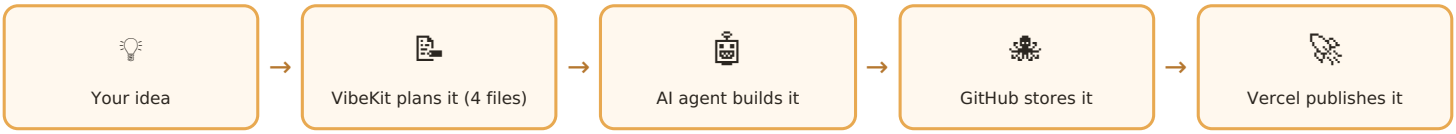
Welcome — The Big Picture

You have an idea. Maybe it's an app that reminds your church group about Bible study, or a little online shop, or a tool that helps families share chores. You are not a software developer — and that is completely fine. This book is written so that a total beginner, who has never opened a "terminal" or typed a single line of code, can go from a napkin idea to a real, working website that other people can visit.

ANALOGY

Think of building an app like building a house. You don't personally lay every brick — you hire a builder. But you still need a **blueprint** (what rooms, what colors, what the kitchen looks like) before the builder can start. In our world: **you** are the homeowner with the idea, **VibeKit** helps you write the blueprint, and an **AI coding agent** (Claude Code or OpenCode) is the builder who reads the blueprint and actually builds the house, room by room.

Here is the entire journey in one picture. Don't worry about understanding every word yet — every stop on this map gets its own chapter.



Why this actually works for a non-developer in 2026

A few years ago, "build an app" meant years of learning to code. Today, AI coding agents can write almost all of the code themselves — but they are only as good as the instructions they're given. A vague instruction like "build me a nice app" produces a messy, broken result. A *detailed blueprint* produces something close to what a professional team would ship. VibeKit's whole job is to turn your fuzzy idea into that detailed, unambiguous blueprint — by interviewing you like a good project manager would.

THE ONE MINDSET SHIFT TO MAKE

You are not "learning to code." You are learning to be a very clear **director** who describes exactly what they want, checks the work, and asks for changes — the same way a film director doesn't operate the camera but still makes every important decision.

What you'll be able to do by the end of this book

- Understand what each tool (VibeKit, Claude Code / OpenCode, GitHub, Vercel) actually does, in plain English
- Set up a computer that's ready to build with AI, even if it's never had a coding tool installed
- Turn your idea into a proper project blueprint using VibeKit's planning prompt
- Hand that blueprint to an AI agent and watch it build your app phase by phase
- Collect the "secret keys" (environment variables) your app needs, safely
- Save your project on GitHub and publish it live on the internet with Vercel
- Know what to do when something breaks — because something always breaks a little, and that's normal

CHAPTER 2

Meet the Cast of Tools

Before we touch anything, let's meet every character in this story. Think of it like introducing the crew before a movie shoot.

🧭 VibeKit

The **planner**. A free framework (built by JB / Desishub) that interviews you about your idea and produces a proper written blueprint — four files that describe exactly what to build, how it should look, and in what order.

🤖 Claude Code / OpenCode

The **builder**. These are "AI coding agents" — programs that live in your terminal, read the blueprint, and actually write, test, and fix the code for your app, one phase at a time.

🐙 GitHub

The **filing cabinet and history book**. A website that stores your project's code safely, keeps a full history of every change, and lets you (or Vercel) grab the latest version any time.

🚀 Vercel

The **stage**. A hosting service that takes the code from GitHub and turns it into a real website anyone in the world can open in their browser, at a real address like `choreschamp.vercel.app`.

The supporting cast (services your app quietly uses)

Tool	What it's for, in plain words
Neon	The **filing room** where all your app's data lives — users, chores, orders, whatever your app stores. It's a database, hosted for you.
Better Auth	The **bouncer at the door** — handles sign-up, login, and "who is allowed to see what."
Resend	The **mail room** — sends emails like "welcome" or "your password was reset."
Upstash Redis	A super-fast **sticky-notes drawer** that remembers recently-asked-for information so your app doesn't have to ask the filing room the same question over and over — this is what makes your app feel fast.
Stripe / DGateway	The **cash register** — handles card payments (Stripe) or Mobile Money like MTN/Airtel (DGateway).
Cloudflare / UploadThing	The **storage locker** — where uploaded photos and files are kept.

ANALOGY

Imagine opening a small restaurant. VibeKit is the consultant who helps you write the full business plan and menu. Your AI agent is the construction crew that builds the kitchen exactly to that plan. GitHub is the recipe book where every version of every recipe is kept, so nothing is ever lost. Vercel is what finally opens your restaurant's doors to the public. Neon is your fridge and pantry. Better Auth is the host who checks reservations at the door.

YOU DON'T NEED TO MASTER ALL OF THIS

You will never personally write a database query or configure a mail server. Your AI agent does that. Your job is to know roughly *what each piece does* so instructions like "add Stripe for payments" or "set up Resend for emails" make sense to you — and so you know where to click when a service asks you to create an account.

Claude Code vs. OpenCode — Which "Builder" to Hire

Both are AI coding agents that work the same basic way: you type instructions in plain English into a terminal window, and the agent reads your files, writes code, runs commands, and reports back. VibeKit works with either one — the blueprint files are agent-agnostic.

	Claude Code	OpenCode
Made by	Anthropic	An open-source community project
Best for a beginner because	Tightest integration with Claude models, very polished, official support and docs	Free/open, works with several different AI models, good if you want flexibility
How you install it	One npm command (Chapter 3)	One npm/curl command (Chapter 3)
How VibeKit talks to it	Auto-installs a Claude Code "skill" file that teaches it the VibeKit rules	Auto-installs a rules file it reads on startup

OUR RECOMMENDATION FOR THIS BOOK

If you're brand new, start with **Claude Code** — the same company (Anthropic) that makes the Claude AI you may already be planning with, and it has the most polished beginner experience. Everything in this book will work the same way with OpenCode; wherever we show a Claude Code command, the OpenCode equivalent is nearly identical.

What "an AI agent works in your terminal" actually means

If you've never seen a terminal, imagine a blank black or white window where you can only type words — no clicking buttons, no icons. It looks intimidating, but you will mostly type two kinds of things: (1) short setup commands you copy-paste exactly as given, and (2) plain-English sentences describing what you want, like you're texting an extremely capable assistant.

TRY PICTURING THIS

You type: `"Add a button that lets a parent mark a chore as done"` — and a minute later, that button exists in your app, styled correctly, connected to your database. That is genuinely what this workflow feels like most of the time.

CHAPTER 3

Setting Up Your Workshop

Before any building happens, you need a few tools installed on your computer. Do this once — you'll reuse the same setup for every future app you build.

The five things you need

1. **A code editor** — Visual Studio Code (VS Code), free from `code.visualstudio.com`. This is where project files live and where your terminal window will open.
2. **Node.js** — the engine that runs JavaScript tools on your computer. Download the "LTS" version from `nodejs.org`.
3. **Git** — the tool that talks to GitHub. Download from `git-scm.com`.
4. **A GitHub account** — free, sign up at `github.com`.
5. **Your AI coding agent** — Claude Code or OpenCode (installed via a terminal command below).

> **ANALOGY**
> If building an app is opening a restaurant, this chapter is buying the oven (Node.js), stocking the pantry with basic tools (Git), getting your business registered (GitHub account), and hiring your head chef (the AI agent). You do this once, and then every new "restaurant" (app) you open afterward reuses the same kitchen.

Opening a terminal for the first time

Open VS Code, then go to the top menu: **Terminal → New Terminal**. A panel appears at the bottom of the screen with a blinking cursor. That's it — that's the terminal. Everything below, you type there and press Enter.

> **DON'T PANIC IF IT LOOKS SCARY**
> You will almost never type a command from memory. You'll copy-paste commands from this book, from VibeKit's site, or ones your AI agent tells you to run. Reading terminal output gets comfortable within a day or two.

Installing your AI agent

Once Node.js is installed, open a terminal and run one of these:

```
# Claude Code
npm install -g @anthropic-ai/claude-code

# OpenCode
npm install -g opencode-ai
```

Then, inside any project folder, typing `claude` (or `opencode`) and pressing Enter starts a conversation with your AI builder right there in the terminal.

Two optional upgrades worth doing on day one

VibeKit recommends two add-ons that make your AI agent noticeably better at design and components. Both are "install once, benefit on every project":

terminal

```
mkdir -p ~/.claude/skills && git clone https://github.com/nextlevelbuilder/ui-ux-pro-max-skill ~/.claude/skills/ui-ux-pro-max
```

This teaches your agent senior-designer habits automatically, in every project.

BEGINNER TIP

If any of these installs give you an error, that's a completely normal first-week experience. Copy the red error text and either search it, or simply paste it to Claude (claude.ai in your browser) and ask "I got this error while installing X, what do I do?" — treat the AI as your patient tech-support friend throughout this whole process.

The Accounts You'll Need — and Why

Most of these are free to start and only ask for a card once you outgrow the free tier. Create them now so Chapters 8–12 move quickly.

Service	Sign up at	Why
GitHub	github.com	Stores your code, required for Vercel deployment
Vercel	vercel.com	Publishes your app to a live URL — sign in with your GitHub account
Neon	neon.tech	Free Postgres database for your app's data
Resend	resend.com	Sends emails (only if your app emails people)
Upstash	upstash.com	Fast cache — VibeKit sets this up for every project by default
Anthropic Console	console.anthropic.com	Only if your app itself uses AI features, and to use Claude Code with your own API key

ORDER OF OPERATIONS

You don't need to create every account today. Create **GitHub** and **Vercel** now. The rest, you'll create the moment VibeKit's blueprint tells your project needs them — usually while your AI agent is working through "Phase 1: Foundation" and asks you for a database connection string.

CHAPTER 4

Meet Our Example App: Chores Champ

To make every following chapter concrete, we'll follow one imaginary app from idea to live website: **Chores Champ**.

THE IDEA, IN ONE SENTENCE

"A simple family app where parents assign chores to kids, kids mark them done, and everyone earns points they can trade for small rewards."

Why this is a good "first app"

- It's easy to picture — everyone understands chores and rewards
- It needs the core building blocks every app needs: **login** (parents vs. kids), **data** (chores, people, points), and a couple of **screens** (dashboard, chore list, rewards)
- It doesn't need payments or file uploads to start, so it's simple — but we'll mention where you'd add them later

Sketching it out before we open any tool

You never need code to plan an app — just answer these in a notebook or in your Notes app, the same questions VibeKit will ask you:

Who uses it?

Parents (assign chores, approve completion) and Kids (see their chores, mark them done, see their points).

What does it store?

A **Family**, made of **People**. Each **Chore** belongs to a Family, is assigned to a Person, and has a point value and a done/not-done status.

What screens exist?

Landing page, login, a Family dashboard, a Chores list, a Rewards page.

What does it need beyond the basics?

Login (Better Auth), a database (Neon), and later — email nudges ("You have 2 chores today!") via Resend.

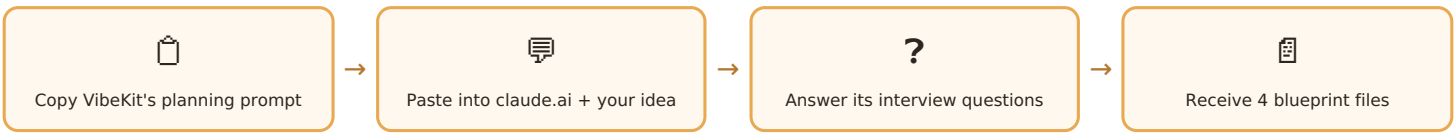
YOUR TURN

Before reading on, jot down these same four answers for *your* real idea. You'll use exactly this when we write the planning prompt in Chapter 5.

CHAPTER 5

Step 1 — Writing the Planning Prompt

VibeKit's whole magic trick lives at `vibekit.desishub.com/docs/quickstart`: a long, carefully written prompt that turns Claude (the chat website, claude.ai — not the coding agent yet) into a professional project planner. You never have to write this prompt yourself — you copy it.



Doing it for Chores Champ

1. Go to `vibekit.desishub.com/docs/quickstart` and click **Copy** on the big prompt block.
2. Open **claude.ai** in your browser and start a new chat.
3. Paste the whole copied prompt. At the very bottom of it there's a line that says `[REPLACE THIS LINE WITH YOUR APP DESCRIPTION]` — replace it with your idea, in your own words.

what you'd paste for Chores Champ

"I want to build Chores Champ — a family app where parents create and assign chores to their kids, kids mark chores as done from their phone, and everyone earns points redeemable for small rewards the parent sets up. Two roles: Parent and Kid. Needs simple login, no payments for now."

Claude reads the whole VibeKit framework in the background and starts acting as your planning assistant.

ANALOGY

This is exactly like walking into an architect's office and saying "I want a small family house with 3 bedrooms." The architect doesn't hand you a blueprint on the spot — they ask you clarifying questions first: "How many bathrooms? Do you want a garage? What style?" That's the interview coming up in Chapter 6.

COMMON BEGINNER MISTAKE

Don't skip describing *who* uses the app and *what data* it needs, even roughly. "Build me something like Instagram" gives Claude almost nothing to work with. "A place where users post photos of their pets and others can comment" gives it real bones to build from.

What "Enough Detail" Looks Like

VibeKit's planning assistant decides how many questions to ask you based on how detailed your first message was. Understanding this saves you time.

Your brief is...	What happens
Very detailed already (users, features, data, design feel, monetization all mentioned)	No interview — it jumps straight to confirming what it understood
Missing a few things	Asks only those 1–4 missing questions, one at a time
Vague ("build me a SaaS")	Runs a full 7–10 question interview covering users, features, data, money, files, email, and design

ANSWER HONESTLY, NOT "CORRECTLY"

There's no wrong answer in this interview. If you don't know whether you need Stripe or DGateway yet, say so — Claude will explain the difference (Stripe = international cards, DGateway = MTN/Airtel Mobile Money for East Africa) and help you decide.

A worked example of the interview, for Chores Champ

Claude asks: "Should kids be able to see other kids' chores and points, like a leaderboard, or only their own?"
You answer: "Yes, a friendly family leaderboard — that's actually the fun part."

Claude asks: "Do you want dark mode support?"
You answer: "Not needed for v1."

Each answer directly shapes the blueprint files generated in the next step. This is the single most important part of the whole process — the ten minutes you spend answering carefully here save you hours of "fixing" later.

CHAPTER 6

The Interview & the Picture Reference

There's one interview step that surprises beginners: VibeKit will refuse to design your app until you show it a **picture**.

> **ANALOGY — WHY WORDS AREN'T ENOUGH**
>
> Tell ten different painters "paint me something premium and clean" and you'll get ten completely different paintings. But show all ten painters the same photograph, and they'll paint something recognizably similar. Words like "clean" or "modern" are too vague for an AI to design from — a real picture removes all the guesswork.

How to do this step

1. Claude will suggest 2–3 search terms for Dribbble (a website full of app design examples), like `"task app modern minimal"` or `"productivity app ui"`.
2. Go to `dribbble.com/search`, paste one of those terms, and browse until a design's colors, fonts, and "feel" match what you picture for your app.
3. Open that design full-size, **right-click → Copy Image**, and paste it straight into your Claude chat (you can do this on both desktop and most mobile browsers).

For Chores Champ, you might search `"family app ui"` or `"kids reward app"` and find something warm, playful, with big rounded buttons and bright accent colors — perfect, since the app is for kids and parents.

> **WHAT CLAUDE DOES WITH YOUR PICTURE**
>
> It "reads" the image and describes back to you, in plain English, what it saw: the background color, whether text is bold or light, whether buttons are pill-shaped or square, how much empty space there is. You get a chance to correct anything before it locks in the design.

Confirming before anything is generated

Right before generating your blueprint files, Claude writes a full summary — app name, users, features, data, integrations, and the design details from your picture — and asks: "*Does this match your intent?*"

> **THIS IS YOUR LAST EASY CHECKPOINT**
>
> Read the summary properly. It is far cheaper to say "actually, change X" here than to ask your AI builder to rip out and redo a feature after code already exists. Take the two minutes.

A Sample "What I Understood" Summary

Here's roughly what Claude would produce for Chores Champ, right before generating the blueprint files — this is exactly the shape of summary you should expect and check for your own idea.

Claude's confirmation message

App: Chores Champ — a family chore and rewards tracker
Primary user: Parents (assign/approve chores) and Kids (complete chores, earn points)
Core features: Chore creation & assignment, mark-as-done flow, points system, family leaderboard, rewards catalog
Data model: Family → has many People; Person → has many Chores; Chore → belongs to a Person and a Family
Integrations: Auth (Better Auth, email + password), Email (Resend, for weekly summaries), Payments (None for v1), File uploads (None for v1), Dark mode (No)
Visual design: Warm cream background, bold rounded sans-serif font, playful pill buttons, soft shadow cards, bright orange accent — "cheerful family app" energy
Out of scope (v1): Payments, multiple families sharing chores, push notifications

You'd reply `"Yes, generate the files"` and move straight to Chapter 7.

CHAPTER 7

The Four Blueprint Files

Once you confirm, Claude generates exactly four files. Download each one (or copy the one big terminal command it gives you, which creates all four at once) into a new, empty folder on your computer — this folder is your project.

1. `project-description.md`

The single source of truth for *what* the app is: users, roles, every feature listed out, the full data model, every page, and which integrations are in or out of scope. Think of it as the restaurant's full menu and business plan in one document.

2. `project-phases.md`

The construction schedule. Breaks the build into phases — Foundation (login, database, layout), Core Features (the actual chores/points functionality), Payments (skipped for Chores Champ v1), File Uploads (skipped), Email, and Polish & Deploy. Your AI builder works through this one phase at a time and pauses for your OK between phases.

3. `design-style-guide.md`

Everything extracted from your Dribbble picture, turned into exact rules: hex color codes, font names and weights, button shapes, card corner-roundness, spacing. Every screen your AI builder makes will follow this file, so the whole app looks consistent — like one designer made it, not twelve.

4. `prompt.md`

The actual message you'll paste into your AI coding agent (Claude Code or OpenCode) to kick off building. It tells the agent to read the other three files in order and start on Phase 1.

ANALOGY

If `project-description.md` is the blueprint and `design-style-guide.md` is the interior design mood-board, then `project-phases.md` is the construction manager's schedule pinned to the site office wall, and `prompt.md` is the one-page memo you hand the construction crew on day one that says "start here, and here's where all your reference documents are."

KEEP THESE FILES FOREVER

Don't delete these four files after building starts. They're your source of truth for the entire life of the project — reopen them any time you or your AI agent need to remember "wait, what were we building again?"

Putting the Files in the Right Place

Create one empty folder for your project, then place the four files directly inside it — not in a sub-folder.

```
# from your desktop, or wherever you keep projects
mkdir chores-champ
cd chores-champ
# now save/move your 4 downloaded .md files into this folder
ls
# you should see:
# project-description.md  project-phases.md  design-style-guide.md  prompt.md
```

`mkdir` means "make directory" (make a new folder). `cd` means "change directory" (step inside that folder). `ls` means "list" the files sitting in the folder you're currently in — a handy sanity check.

OPEN THE FOLDER IN VS CODE

In VS Code: **File → Open Folder…** → select `chores-champ`. Now your editor and your terminal are both pointed at the same project, which is exactly where the next chapter picks up.

CHAPTER 8

Installing the Framework Files

Your four blueprint files describe *your* app. There's also a shared "coding constitution" that VibeKit uses on every project — the exact tech stack rules, coding standards, and a library of ready-made components. One command installs all of it into your project folder.

```
npx vibekit-framework init
```

Run this from inside your `chores-champ` folder. It automatically detects whether you have Claude Code or OpenCode installed and sets up the right files for it.

What gets installed

- `master_prompt.md` — the tech stack rules your agent must follow
- `jb-components.md` — a catalog of ready-made pieces (login screens, data tables, file uploads) your agent should reuse instead of building from scratch
- `pre-deploy-review.md` — a security/quality checklist prompt (Chapter 13)
- A rules file specific to your agent, so it auto-loads all of this every time you start it

Why this matters

Without this step, your AI builder is a talented contractor with no local building codes to follow — it might use an outdated database pattern, forget security basics, or reinvent a login screen instead of using a well-tested one. This command hands it the local rulebook.

GOOD TO KNOW

This command is **safe to run again** later — it never overwrites files you've since edited yourself, it only fills in what's missing.

CHAPTER 9

Handing the Blueprint to Your AI Builder

This is the moment the app actually starts existing as real code.

```
# inside your chores-champ folder
claude
# — or —
opencode
```

This opens a chat right inside your terminal. Open the `prompt.md` file VibeKit generated, copy its entire contents, and paste it as your first message to the agent.

> **ANALOGY**
> This is the ribbon-cutting moment where you hand the construction crew the keys, the blueprint, and the schedule, and say "Phase 1, go." From here, your job shifts from planner to **site supervisor** — you check in after each phase rather than laying every brick yourself.

What happens next

1. The agent reads all five reference files (the coding rules, the component catalog, and your three project files).
2. It starts **Phase 1 — Foundation**: setting up the project structure, applying your design colors and fonts, connecting the database, and building login.
3. Partway through, it will pause and ask you for things only you can provide — like your Neon database connection string, explained in the next chapter.
4. When Phase 1 is fully done, it **stops and asks you to confirm** before moving to Phase 2. This is intentional — check the app actually runs and looks right before more gets built on top of it.

> **CHECKING THE APP AS IT'S BUILT**
> Your agent will tell you to run `pnpm dev` (start a local preview) and open `http://localhost:3000` in your browser. This shows you the app running privately on your own computer — nobody else can see it yet. This is completely normal and how every phase gets checked before moving on.

> **IF A PHASE PRODUCES SOMETHING WRONG**
> Don't start over. Just describe what's wrong, in plain English: "the button color should be orange, not blue" or "clicking 'mark done' doesn't update the points." Treat it exactly like giving feedback to a human developer — specific and calm feedback gets fixed fast.

A Realistic Phase-by-Phase Timeline

For an app the size of Chores Champ, here's roughly what to expect. Real timing varies with how many changes you request.

Phase	What gets built	Typical first pass
1 — Foundation	Project setup, design applied, database connected, login working	30–90 minutes
2 — Core Features	Chores, points, leaderboard, rewards — real data, real screens	1–3 hours
3 — Payments	Skipped for v1 (no monetization yet)	—
4 — File Uploads	Skipped for v1	—
5 — Email	Weekly summary emails via Resend	30–60 minutes
6 — Polish & Deploy	Testing, safety review, going live	1–2 hours

THIS ISN'T A RACE

Stopping to actually click through the app after each phase — on your phone too, not just your laptop — is what separates a polished result from a rushed one. Slower and checked beats fast and broken.

CHAPTER 10

Environment Variables — Your App's Secret Keys

At some point your AI agent will say something like "I need your `DATABASE_URL`" and pause. This chapter demystifies exactly what that means.

> **ANALOGY**
> Environment variables are like the labelled keys on a big keyring hanging by your restaurant's back door: one key opens the walk-in fridge (database), one opens the mailroom (email), one opens the till (payments). Your app needs the right keys to open the right doors — and just like real keys, you never leave them lying around in public (like on GitHub) where a stranger could pick them up.

Where these keys live

They go in a file called `.env.local`, sitting in your project folder. VibeKit already creates this file for you with a labelled slot for every service your app needs — you just have to fill in the blanks by signing up for each service and copying a value in.

`.env.local — filled in for Chores Champ`

```
DATABASE_URL="postgresql://user:pass@ep-xxx.neon.tech/neondb"
BETTER_AUTH_SECRET="a-long-random-string"
BETTER_AUTH_URL="http://localhost:3000"
RESEND_API_KEY="re_xxxxxxxx"
UPSTASH_REDIS_REST_URL="https://xxx.upstash.io"
UPSTASH_REDIS_REST_TOKEN="AYx..."
```

How to fetch each key, in short

Key	Get it from	Where exactly
DATABASE_URL	neon.tech	New Project → Connection Details → copy the "Pooled connection" string
BETTER_AUTH_SECRET	your own terminal	Run `openssl rand -base64 32` and paste the output
RESEND_API_KEY	resend.com	API Keys → Create API Key → copy the value starting `re_`
UPSTASH_REDIS_REST_URL / TOKEN	upstash.com	Redis → Create Database → scroll to "REST API"
GOOGLE_CLIENT_ID / SECRET	console.cloud.google.com	Only if you add "Sign in with Google" — Credentials → OAuth client ID
STRIPE_SECRET_KEY / DGATEWAY_API_KEY	dashboard.stripe.com or dgateway.desispay.com	Only when you add payments — Developers → API keys

> **THE THREE GOLDEN RULES OF SECRET KEYS**
> 1) Never paste a key into a public chat or a public GitHub page.
> 2) Use **test** keys while building, **live** keys only once you're really accepting real payments.
> 3) Always restart your app (`pnpm dev`) after editing `.env.local` — it only reads the file when it starts up.

Why This File Must Never Reach GitHub

Your project has a special file called `.gitignore` that tells GitHub "never save these files, ever." VibeKit already lists `.env.local` in there for you — but it's worth understanding why this matters so much.

WHAT HAPPENS IF A KEY LEAKS

If your `DATABASE_URL` or a payment key ends up on public GitHub, anyone in the world who finds it can read your users' data or run up a bill on your account within minutes — bots scan GitHub for leaked keys constantly. This is the single most common beginner mistake, so it gets its own warning box.

Two files, two purposes

.env.example

A safe, empty template *with* the labels but *without* real values. This one is fine to save to GitHub — it just shows future-you or a teammate which keys the project needs.

.env.local

Has your actual real secret values filled in. This one must **never** be saved to GitHub. VibeKit's setup already blocks it — just don't manually override that.

IF YOU EVER SET UP HOSTING YOURSELF

On Vercel, you re-enter each key separately into **Project Settings → Environment Variables** in the dashboard — this is covered in Chapter 12. Vercel never reads your local `.env.local` file; it has its own private copy.

CHAPTER 11

GitHub for Absolute Beginners

GitHub does two jobs for you: it's a safe backup of your code, and it's the bridge Vercel uses to publish your app. You need to understand four words: **repository**, **commit**, **push**, **clone**.

Repository ("repo")

One project's folder, as stored on GitHub. Chores Champ will live in a repo simply called `chores-champ`.

Commit

A saved snapshot of your code at one moment, with a short note describing what changed — like a save-point in a video game.

Push

Sending your saved snapshots up from your computer to GitHub's website.

Clone

Downloading a full copy of a repo from GitHub onto a computer — useful if you switch laptops.

Connecting your project to GitHub for the first time

1. On github.com, click the **+** icon → **New repository** → name it `chores-champ` → keep it **Private** for now → Create.
2. GitHub shows you a short list of commands. In your project's terminal, run:

```
git init
git add .
git commit -m "First version of Chores Champ"
git branch -M main
git remote add origin https://github.com/YOUR-USERNAME/chores-champ.git
git push -u origin main
```

In plain English: "start tracking this folder" (`init`), "select everything to save" (`add .`), "save a snapshot with this note" (`commit`), "name the main timeline 'main'" (`branch`), "remember where GitHub's copy lives" (`remote add`), and finally "send it up" (`push`).

YOUR AI AGENT CAN DO THIS FOR YOU

You can literally type into Claude Code: "*Initialize git and push this project to a new GitHub repo called chores-champ*" — most agents can run these commands for you once you've logged into GitHub on your computer once (`gh auth login`).

Saving Progress As You Go

You'll repeat a short version of the push steps every time your AI agent finishes meaningful work — think of it as hitting "save" regularly rather than only once.

```
git add .
git commit -m "Add points and leaderboard screen"
git push
```

ANALOGY

If GitHub is your recipe book, every commit is a dated page you've added: "Tuesday — added the leaderboard recipe." Years from now you can flip back and see exactly what your app looked like at any point, and undo a bad change by going back to an earlier page.

A note on Private vs. Public repos

Private means only you (and people you invite) can see the code. Public means anyone can see it. For a personal project or an early-stage business idea, **keep it Private** — you can always make it public later once you're comfortable sharing.

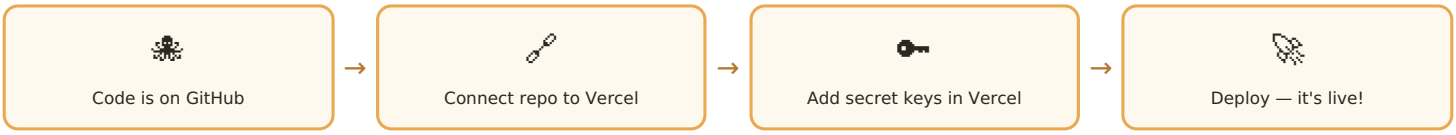
DOUBLE-CHECK BEFORE YOUR VERY FIRST PUSH

Open your project folder and confirm a file named `.gitignore` exists and lists `.env.local` and `node_modules`. VibeKit sets this up correctly by default — this is just your one habit of verifying it before that first push.

CHAPTER 12

Going Live with Vercel

This is the chapter where Chores Champ stops being "something on my laptop" and becomes a real website anyone can open.



Step-by-step

1. Go to **vercel.com** and sign in with your GitHub account (this makes GitHub and Vercel trust each other automatically).
2. Click **Add New → Project**, and pick the `chores-champ` repository from the list.
3. Vercel automatically recognizes it's a Next.js project (VibeKit builds everything on Next.js) — you usually don't need to change any build settings.
4. Before clicking Deploy, open **Environment Variables** and paste in every value from your `.env.local` file, one by one — the same secret keys from Chapter 10, just typed into Vercel's dashboard instead of a local file.
5. Update `BETTER_AUTH_URL` and `NEXT_PUBLIC_APP_URL` to your real Vercel address (like `https://chores-champ.vercel.app`) instead of `localhost:3000`.
6. Click **Deploy**. Vercel builds your app on its own servers — this takes one to three minutes.
7. When it finishes, you get a real, working URL. Open it on your phone and show someone — this is the moment it becomes real.

ANALOGY

GitHub is the recipe book; Vercel is the actual restaurant kitchen that reads the latest recipe and cooks the meal for every visitor who walks in — automatically, every single time you push a new commit to GitHub.

EVERY FUTURE PUSH UPDATES THE LIVE SITE AUTOMATICALLY

Once connected, you never manually "re-deploy." Every time you `git push` new changes, Vercel notices and rebuilds the live site within a couple of minutes. This is called **continuous deployment**.

A Real Domain Name (Optional)

A free `.vercel.app` address works perfectly well and is fine to launch with. If you'd like something like `choreschamp.com` instead:

1. Buy the domain from any registrar (Namecheap, GoDaddy, or Cloudflare Registrar).
2. In your Vercel project, go to **Settings → Domains → Add**, and type your domain.
3. Vercel gives you DNS records to add at your registrar (or, if you use Cloudflare for DNS, add them there). This connects your domain to your live app.
4. Within a few minutes to a few hours, your custom domain goes live with a free padlock (SSL) automatically included.

> **NO RUSH**
> A custom domain is a nice-to-have, not a requirement. Ship with the free `.vercel.app` address first, get real feedback, and buy a domain once you know the idea has legs.

CHAPTER 13

The Pre-Deploy Safety Check

Before your very first real deploy — and again before any major new feature goes live — run VibeKit's pre-deploy review. It's the difference between "looks done" and "actually production-ready."

> **ANALOGY**
> This is the building inspector who walks through the finished house before you're allowed to move in — checking the wiring, the locks on the doors, and that nobody left a fake "5-star reviews!" sign up that nobody actually wrote.

How to run it

Open your AI coding agent in your project folder and paste the contents of `pre-deploy-review.md` (installed back in Chapter 8). It performs a full audit and writes its findings to a new file, `pre-deploy-review-report.md`.

What it checks, in plain terms

Category	Beginner translation
Performance	Is the app fast? Are images and heavy features loaded smartly instead of all at once?
Database & caching	Is data fetched efficiently, and is the sticky-notes cache (Redis) actually being used where it should be?
Security	Are passwords, forms, and login flows protected against common attacks? Are any secret keys accidentally exposed?
Honest copy	Did the AI accidentally invent fake stats or fake testimonials ("Trusted by 10,000 families!") that were never actually true? These get flagged and removed.

> **FIX EVERY "CRITICAL" ITEM BEFORE YOU DEPLOY**
> "High" priority items can wait a week; "Medium" items can wait for your next update. But address every single Critical finding first — these are the ones that can genuinely hurt your users or your app's security.

CHAPTER 14

When Things Go Wrong

Something will break at some point — for every developer, beginner or expert. Here's how to stay calm and fix it.

Symptom	Likely cause	What to do
"redirect_uri_mismatch" on login	The URL you registered with Google/GitHub doesn't exactly match your app's real URL	Go back to the OAuth app settings and make sure the callback URL matches exactly, including `http` vs `https`
App worked yesterday, breaks today after editing .env.local	Dev server wasn't restarted	Stop it (Ctrl+C in the terminal) and run `pnpm dev` again
"Invalid API key" error	Using a test key where a live key is expected, or vice versa, or a typo	Recheck which mode (test/live) the dashboard is in, and re-copy the key carefully
Vercel deploy fails	An environment variable is missing on Vercel that exists locally	Compare your local `.env.local` against Vercel's Environment Variables list — add whatever's missing
Your AI agent seems "confused" mid-project	Context has drifted from the original blueprint	Remind it: "please re-read project-description.md and project-phases.md before continuing"

THE UNIVERSAL FIRST MOVE: PASTE THE ERROR

Copy the exact error text (the red text in your terminal or browser) and paste it straight to your AI agent with: "*I got this error, what does it mean and how do I fix it?*" This alone solves the majority of beginner problems.

ANALOGY — ERRORS ARE NOT FAILURE

A doctor doesn't panic at an unusual symptom — they read it as information and investigate. Treat every red error message the same way: it's not proof something is broken forever, it's a clue pointing at exactly what to fix next.

Where to get unstuck if the AI can't solve it

- VibeKit's `troubleshooting.md` reference file (installed in Chapter 8)
- VibeKit's WhatsApp community and GitHub Discussions, linked from vibekit.desishub.com
- Searching the exact error text on the web — someone else has almost certainly hit it before

Good Habits That Prevent Most Problems

- **Commit and push often.** Small, frequent commits mean you can always go back to "the last version that worked."
- **Test on your phone, not just your laptop.** Most real users will visit on mobile first.
- **Never skip the confirmation pause between phases.** It exists specifically to catch problems early, while they're cheap to fix.
- **Keep test and live payment keys in separate, clearly-labelled places.** Mixing them up is the most common costly mistake beginners make.
- **Re-run the pre-deploy review after every major feature**, not just once at the very start.

A FIVE-MINUTE WEEKLY HABIT

Once Chores Champ is live, spend five minutes each week opening it fresh — as if you were a stranger — and clicking through the core flow (sign up → assign a chore → mark it done). This catches the small things that are easy to miss when you're deep in building.

CHAPTER 15

Glossary — Every Word Explained Simply

Term	Plain-English meaning
Terminal	A text-only window where you type commands instead of clicking buttons
Repository (repo)	A project's folder as stored on GitHub
Commit	A saved snapshot of your code with a note about what changed
Push / Pull	Sending your saved snapshots up to GitHub / bringing the latest ones down
Deploy	Publishing your app so it's live on the internet
Database	Where your app's information (users, chores, orders) is permanently stored
API / API route	A specific "door" in your app's backend that the front-end asks for information through
Environment variable	A secret setting (like a password or key) your app reads when it starts, kept out of the code itself
Auth / Authentication	The system that handles logging in and knowing who's who
Cache	A fast, temporary "sticky note" copy of data kept nearby so the app doesn't have to re-fetch it every time
Framework	A pre-built set of rules and tools (like VibeKit, or Next.js) that saves you from building everything from zero
AI coding agent	A program (Claude Code, OpenCode) that reads instructions and writes/edits real code for you
Phase	One chunk of the build plan — e.g. "Foundation," "Core Features" — completed and checked before moving on
OAuth	The technical name for "Sign in with Google/GitHub" style login
Webhook	A message a service (like Stripe) automatically sends your app when something happens, e.g. "a payment succeeded"
DNS / Domain	The system that turns a human-readable address (choreschamp.com) into the technical address of your live app
Localhost	Your own computer, when your app is only running privately for you to test

More Terms You'll Meet

Term	Plain-English meaning
Next.js	The underlying toolkit your app's code is written in — think of it as the type of construction material the whole house is built from
Prisma	The translator between your app's code and the database — lets your AI agent talk to Neon without writing raw database language
Route / Page	One screen of your app, tied to a web address like `/dashboard`
Component	One reusable visual building block — a button, a card, a form — that gets reused across many screens
Migration	A recorded change to your database's structure — e.g. "add a points column to the Person table"
Seed data	Realistic sample data (fake chores and families) your AI agent creates so you have something to click through while testing
SSL / padlock	The little padlock icon in a browser showing the connection to a site is encrypted and safe — Vercel adds this automatically
Responsive design	An app that resizes and rearranges itself to look right on phones, tablets, and laptops alike

CHAPTER 16

Your One-Page Cheat Sheet

Pin this page. It's the whole journey, condensed.

The seven-step flow, every time you start a new app

1. **Plan** Copy VibeKit's prompt from vibekit.desishub.com/docs/quickstart into claude.ai, describe your idea, answer the interview, paste one Dribbble image.
2. **Confirm** Read the "What I understood" summary carefully, then say "Yes, generate the files."
3. **Save** Download the 4 files into a new empty project folder.
4. **Install** Run `npx vibekit-framework init` in that folder.
5. **Build** Run `claude` (or `opencode`), paste `prompt.md`, work through phases one at a time, filling in secret keys as asked.
6. **Ship** Push to GitHub, connect the repo on Vercel, add your environment variables there, click Deploy.
7. **Review** Run the pre-deploy-review prompt, fix every Critical item, then share your live link.

Commands you'll type most often

```
git add .
git commit -m "describe what changed"
git push

pnpm dev          # preview your app locally
claude            # start your AI builder
npx vibekit-framework init
```

The three habits that prevent 90% of problems

1. Confirm carefully before generating files, and between every build phase.
2. Never let `.env.local` touch GitHub — keep secret keys secret.
3. Commit and push often, so you can always step back to a version that worked.

> **YOU ARE READY**
>
> Everything in this book is one page-flip away whenever you need it. Open `vibekit.desishub.com/docs/quickstart`, and start turning your next idea into something real.