Von null zur ersten App

Eine Methode, kein Rezept.

Cover-Bild: ruhiges Wien-Editorial-Stillleben
(cover-eingangstuer.png — noch nicht generiert;
führe die feedfoundry-Bildpipeline aus und baue erneut)

Was drinsteht

1. **Teil 0, Orientierung.** Was Agentic Building wirklich ist, die zwei Wege, und warum Verifikation der entscheidende Skill ist.
2. **Teil 1, Dein Werkzeugkasten.** Die Tool-Tabelle und die Reihenfolge-Regel: womit du anfängst und was bewusst später kommt.
3. **Der Methode-Loop, angewandt.** Eine echte kleine App, einmal von vorne bis hinten gebaut.
4. **Wie es weitergeht.** Der Cliffhanger, die Community und der nächste Schritt.

Teil 0: Orientierung

Was das hier wirklich ist (und was nicht)

Du wirst überall „Vibe Coding„ hören. Gemeint ist meistens: Du beschreibst der KI in normaler Sprache, was du willst, und sie baut es. Das funktioniert, und es ist erstaunlich weit. Aber es gibt einen Unterschied, den kaum jemand offen sagt: zwischen einer *Demo* und einem *Produkt*.

Eine Demo läuft genau einmal, vor dir, auf deinem Bildschirm. Ein Produkt läuft auch dann, wenn jemand anderer es benutzt, etwas Falsches eintippt, oder du nachts schläfst. Der Sprung von Demo zu Produkt ist die ganze un-sexy Arbeit, die in den meisten Gratis-Tutorials fehlt. Wir fangen hier mit der Demo an. Das ist richtig so. Aber ich will, dass du von Anfang an weißt, dass es danach noch weitergeht.

Die zwei Wege

Es gibt im Grunde zwei Arten, wie du mit KI etwas baust, und du solltest wissen, welcher Weg gerade dein Weg ist.

Der erste Weg ist der **App-Builder**. Du beschreibst oder klickst, die KI baut dir eine fertige Web-App im Browser. Kein Terminal, keine Installation, kein Verständnis von Dateien nötig. Das ist der Einstieg, und es ist der Weg, den ich dir in diesem Handout zeige.

Der zweite Weg ist die **Agentic-CLI**. Da arbeitet ein Agent in deinem eigenen Projekt-Ordner, im Terminal, mit Zugriff auf jede Datei. Das ist mächtiger, mehr Kontrolle, mehr Tiefe. Aber es setzt voraus, dass dich ein schwarzes Terminal-Fenster nicht abschreckt. Deshalb kommt dieser Weg bewusst später, nicht jetzt.

Verifikation ist der eigentliche Skill

Hier ist der Teil, den fast niemand lehrt, und der den Unterschied macht. Die KI schreibt dir den Code. Sie wird dir auch sehr selbstbewusst sagen, dass alles funktioniert. Das heißt nur leider nicht, dass es stimmt.

Der Skill, den du dir aneignen musst, ist nicht Programmieren. Es ist **Verifikation**. Zwei Sätze, die ich mir gemerkt habe: Gib dem Agenten einen Check, den er selbst laufen lassen kann. Und: Verlange Evidenz, vertraue nicht blind. Du bleibst der Mensch in der Schleife. Simon Willison nennt das treffend „vibe *engineering*, not vibe coding„. Die Verantwortung bleibt bei dir, auch wenn die KI tippt.

Das klingt jetzt vielleicht groß. Im echten Loop weiter unten ist es ein einziger konkreter Schritt. Mehr nicht, fürs Erste.

Das Grundprinzip

Das Wichtigste an diesem ganzen Handout ist ein einziger Satz: **Du kannst die KI bei allem fragen.** Wenn du nicht weiterweißt, ist das keine Sackgasse, sondern die nächste Frage. „Was bedeutet diese Fehlermeldung?„ „Erklär mir das, als hätte ich noch nie programmiert.“ „Wie prüfe ich, ob das wirklich tut, was es soll?„

Genau deshalb lehre ich eine Methode und kein Rezept. Ein Rezept funktioniert nur für ein Gericht. Die Methode wendest du auf deinen eigenen Use-Case an, und die KI macht den individuellen Rest. Was bei dir herauskommt, weiß ich nicht. Du baust schließlich, was *du* willst.

Teil 1: Dein Werkzeugkasten

Hier ist die Übersicht über die Werkzeuge, die immer wieder vorkommen. Lies sie einmal durch, aber merk dir vor allem die Reihenfolge-Regel direkt darunter. Die ist wichtiger als jedes einzelne Tool.

Werkzeug	Was / für wen	Erster Start	Einstieg
Lovable	App-Builder, schönste UIs · non-coder	Browser → beschreib deine App	**ab Teil 0/1**
Bolt	App-Builder, transparenter Code, Deploy · non-coder/ Techie	bolt.new → Prompt	**ab Teil 0/1**
Claude Code	Agentic-CLI, bestes Code-Reasoning · Techie	Terminal → `claude`	Teil 2+
OpenAI Codex	Agentic-CLI, in ChatGPT gebündelt · Techie	CLI/IDE → ChatGPT-Login	Teil 2+
Cursor	AI-Editor (Brücke CLI ↔ non-coder)	App installieren	Teil 2+
Ollama	lokale Modelle, CLI, Dev-Endpoint	`ollama run …`	Teil 2+ (optional)
LM Studio	lokale Modelle, GUI für non-coder	klicken, Modell laden	Teil 2+ (optional)

Werkzeug	Was / für wen	Erster Start	Einstieg
Docker	MCP-Server / lokale Stacks / Deploy	Desktop, eine `run`-Zeile	Teil 2+ (optional)

> **Reihenfolge-Regel:** Cloud-API und App-Builder zuerst, alles andere danach. Claude Code, Codex, Cursor, die lokalen Modelle (Ollama, LM Studio) und Docker sind in der Tabelle bewusst als „kommt in Teil 2+„ markiert.

Warum diese Reihenfolge? Weil ich gesehen habe, woran Anfänger abspringen, und es ist fast immer dasselbe: das Wort „Terminal„. Wenn dein erster Schritt ein schwarzes Fenster und ein Installations-Befehl ist, hörst du auf, bevor du je etwas gebaut hast. Mit dem App-Builder hast du in einer halben Stunde etwas Sichtbares. Das schnelle erste Erfolgserlebnis trägt dich durch den Rest.

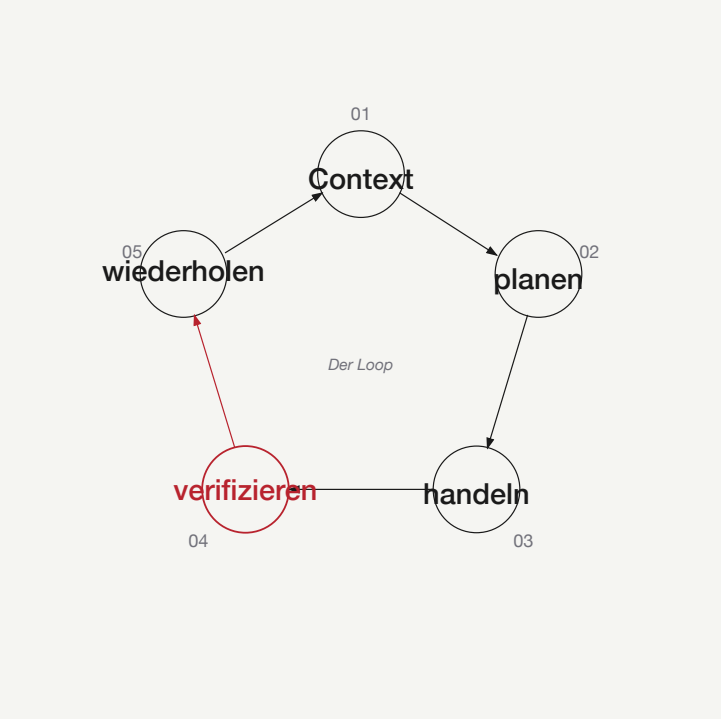
Die CLI, die lokalen Modelle und Docker sind kein Rückschritt, sie sind ein **Capability-Upgrade**. Du holst sie dir, wenn du sie brauchst: für mehr Privacy und DSGVO, für Kostenkontrolle, für Offline-Betrieb. Aber das ist die zweite App, nicht die erste.

Eine ehrliche Fußnote zu den Preisen: Die Tools haben Gratis-Stufen, und die Bezahl-Stufen liegen grob im Bereich von 20 bis 25 Dollar pro Monat. Genau festnageln will ich das nicht, weil sich die Preise praktisch monatlich ändern. Prüf vor jeder Nutzung kurz selbst nach, was gerade gilt.

Der Methode-Loop, angewandt

Jetzt bauen wir wirklich. Ein einziges Mal, von vorne bis hinten, an einem echten Beispiel. Werkzeug: **Lovable**. Was wir bauen: eine **Landing-Page mit einem Lead-Formular** (Name und E-Mail eintragen). Das ist klein genug, um es in einer Sitzung zu schaffen, und echt genug, dass du es danach benutzen könntest.

Der Loop hat fünf Schritte: Context sammeln, planen, handeln, verifizieren, wiederholen. Bei unserer Seite heißt das ganz konkret: planen ist deine Mini-Spec, handeln ist das Bauen-Lassen, verifizieren ist der eine Verify-Schritt. Den Loop merkst du dir einmal, und du wendest ihn auf alles an, was du je baust.



1. Context sammeln

Bevor du der KI irgendetwas sagst, klär für dich selbst drei Dinge. Für wen ist die Seite? Was soll der Besucher tun? Was passiert mit dem, was er einträgt? Das muss nichts

Aufwendiges sein, zwei, drei Sätze auf einem Notizblock reichen. Der Grund: Je klarer du den Context hast, desto klarer kann die KI bauen. Eine vage Ansage führt zu einem vagen Ergebnis, immer.

2. Planen: deine Mini-Spec

Jetzt sagst du Lovable, was du willst, aber in geordnet. Ich nenne das eine Mini-Spec: kein Roman, sondern eine kurze, klare Ansage. So ungefähr würde ich es schreiben:

> *Bau mir eine schlichte Landing-Page für einen Newsletter über KI-Tools.*
> *Oben eine Überschrift und ein Satz, der erklärt, worum es geht.*
> *Darunter ein Formular mit zwei Feldern: Name und E-Mail.*
> *Ein Button „Eintragen„.*
> *Wenn jemand absendet, soll eine kurze Dankesnachricht erscheinen.*
> *Schlichtes, ruhiges Design, viel Weißraum.*

Mehr braucht es für den ersten Wurf nicht. Du sagst, *was* du willst, nicht *wie* es technisch geht. Das *Wie* ist der Job der KI.

3. Handeln: bauen lassen

Du fügst die Mini-Spec in Lovable ein und lässt bauen. Nach kurzer Zeit hast du eine echte Seite vor dir, mit deinem Text und deinem Formular. Gefällt dir etwas nicht, sagst du es einfach im nächsten Satz: „Mach die Überschrift größer.„ „Das Formular soll mittig stehen.“ Du redest mit der KI weiter, als wäre sie jemand, der neben dir baut. Das ist der ganze Trick, kein Code-Wissen nötig.

4. Verifizieren: ein Verify-Schritt

Und hier kommt der Teil, den die meisten überspringen, und den du nicht überspringst. Die Seite *sieht* fertig aus. Aber tut das Formular auch wirklich, was es soll? Genau das prüfen wir, und zwar so, dass die KI den Check selbst macht und dir Evidenz liefert, statt dass du blind vertraust.

Sag ihr so etwas:

> *Prüf bitte, ob das Formular wirklich absendet und die Eingaben ankommen.*
> *Trag testweise einen Namen und eine E-Mail ein, schick es ab,*
> *und zeig mir, wo die Daten landen und ob sie vollständig angekommen sind.*

Wenn die KI dir zeigen kann, dass eine Testeingabe wirklich durchläuft und ankommt, hast du Evidenz. Wenn nicht, weißt du genau, wo es hakt, und du fragst nach. Das ist Verifikation in ihrer einfachsten Form: ein Check, den die KI selbst laufen lässt, und ein Beweis statt eines Versprechens.

5. Wiederholen

Mehr ist der Loop nicht. Du gehst zurück zu Schritt eins, nimmst die nächste Kleinigkeit, und drehst die Runde nochmal. Überschrift schärfen, ein Feld ergänzen, eine Bestätigungs-Mail. Jede Runde ist klein, jede Runde wird verifiziert. So wächst aus der Demo nach und nach etwas Belastbares.

Und genau hier fängt das an, was dieses Handout nicht mehr abdeckt. Denn sobald dein Formular läuft, kommen die un-sexy Fragen: Läuft es auch *sicher*? Wo genau landen die E-Mail-Adressen, die dir fremde Menschen anvertrauen? Und was sagt eigentlich die DSGVO dazu? Diese Fragen beantworte ich hier nicht zu Ende. Aber ich will, dass du sie riechst, weil sie der Unterschied zwischen „läuft bei mir„ und „läuft verlässlich“ sind.

Wie es weitergeht

> ***Es läuft. Aber läuft es verlässlich?***

Das ist die Frage, an der sich alles entscheidet. Deine erste App läuft, und das ist ein echter Moment, den solltest du genießen. Aber zwischen „es läuft auf meinem Bildschirm„ und „ich kann das anderen Menschen zumuten“ liegt die ganze Arbeit, die kaum jemand zeigt.

Genau diese Tiefe ist der nächste Schritt: der €29-Guide **„Von null zur verlässlichen App„**. Da geht es um Verifikation über den einen Check hinaus, um Security und Secrets (die Falle, in die massenhaft schnell gebaute Apps tappen), um Kostenkontrolle, bevor die Rechnung überrascht, um den Weg vom lokalen Bauen zum echten Deploy, und um eine DSGVO-Checkliste, technisch gerahmt. *(Kein Rechtsrat. Für rechtlich Verbindliches sprich mit Anwalt oder Steuerberater.)* Dazu alle zehn Use-Cases, nicht nur der eine von oben, jeder mit eigener Mini-Spec und eigenem Fertig-Check. Buy once, in Ruhe durcharbeiten.

Davor und daneben: Du musst das nicht allein machen. Im **Discord** `agenticbuilders.at` sitzen Leute, die gerade dasselbe bauen. Stell deine Fragen, zeig deinen Build, hol dir Peer-Help. Ich bin auch dort. Komm vorbei, bevor du dich an einer Fehlermeldung festbeißt.

Deine zwei nächsten Schritte

1. Discord `agenticbuilders.at` — Fragen stellen, deinen Build zeigen, Peer-Help holen.

2. €29-Guide „Von null zur verlässlichen App„ — die ganze Tiefe: Verifikation, Security, Kosten, Deploy, DSGVO-Checkliste, alle zehn Use-Cases.

Wenn du weiterlernen willst

Du brauchst keine 50 Lesezeichen, um anzufangen. Hier sind fünf Quellen, die mir wirklich etwas gebracht haben, in der Reihenfolge, in der ich sie dir geben würde. Alle sind gratis, die meisten auf Englisch.

- **Anthropic Academy, „Claude Code 101"**: ein gratis Hands-on-Kurs, der dich vom Nichts zur ersten echten Session führt. Der sanfteste Einstieg, den ich kenne.
- **Andrej Karpathy, „Software Is Changing (Again)„**: der eine Talk, der erklärt, *warum* sich gerade alles ändert. Wenn du nur eine Sache schaust, dann diese.
- **Simon Willison, „Not all AI-assisted programming is vibe coding„**: die ehrliche Begriffsklärung. Warum Verifikation kein Detail ist, sondern der ganze Punkt.
- **Anthropic, „Best Practices for Claude Code„**: wenn du tiefer willst, ist das die kanonische Anleitung, sauber und ohne Hype.
- **Discord `agenticbuilders.at`**: kein Artikel, sondern Menschen, die gerade dasselbe bauen wie du. Frag, bevor du dich festbeißt.

Die volle, gepflegte Sammlung, geordnet vom Einstieg bis tief, steht auf `agenticbuilders.at/ressourcen`. Sie ist die lebende Quelle, wir aktualisieren sie, wenn sich die Tool-Landschaft dreht. Diese fünf hier sind bewusst die Eingangstür, nicht die ganze Bibliothek.

Schlussbild: Wien-Stillleben, Abendlicht
(closing-werkbank.png — noch nicht generiert)