Toolkit: 100 godzin pracy z AI w pigułce

Cheatsheet do artykułu "*Sto godzin, 400 dolarów i 5 lekcji. Anatomia jednego narzędzia AI*". Wzory notatek, polecenia do skopiowania, drzewo decyzyjne i plan startowy dla nowego projektu — wszystko gotowe do wykorzystania.

> **Dla kogo:** osoby, które same eksperymentują z AI — niezależnie od poziomu technicznego. Polecenia w czarnych ramkach są dla osób, które wolą pracować z linią poleceń (terminalem). Wzory notatek i drzewo decyzyjne — dla każdego.

1. TL;DR pięciu lekcji

1. **Nie ulepszaj tego, co działa, zanim nie odtworzysz tego, co działa.** Najpierw zbuduj kopię oryginału, zmierz że działa tak samo, dopiero potem optymalizuj.
2. **Testuj na pięciu produktach, nie na dwustu.** Nigdy nie uruchamiaj przetwarzania >5 pozycji bez wcześniejszej akceptacji próbki. Wybieraj po typach, nie losowo.
3. **Zapisuj surowe odpowiedzi z AI, naprawiaj na bazie tego co masz.** Pełne odpowiedzi w bazie = możesz naprawiać błędy bez ponownego płacenia za AI.
4. **Drugi model AI jako recenzent i adwokat diabła.** Przed pisaniem kodu — drugi model zadaje 5 najtrudniejszych pytań. 12 minut rozmowy = 5 godzin oszczędzone.
5. **Notatki z projektu jako mapa.** Notatka końca sesji + wnioski z błędów per projekt. 3 minuty pisania = 30 minut przy powrocie do pracy.

---

2. Wzory do skopiowania

2.1 Notatka końca sesji (markdown)

Zapisz w pliku, np. w katalogu `.ai/handoffs/` w projekcie. Nazwa pliku: data + krótki tytuł.

```
# Notatka YYYY-MM-DD HH:MM — [krótki tytuł]

**Stan:** aktywny / stabilny / zawieszony

## Co zostało zrobione
- [punkt 1, najlepiej z odniesieniem do pliku]
- [punkt 2]

## Co zostało do zrobienia
- [konkretne zadanie, najlepiej z plikem i numerem linii]

## Przeszkody / decyzje czekające
- [jeśli są, inaczej "brak"]

## Pierwsza akcja w nowej sesji
[1-2 zdania: co dokładnie zrobić jako pierwsze]

## Czego nie ruszać
- [pliki/dane, których NIE wolno modyfikować]
```

2.2 Wnioski z błędów (Lessons Learned) — jeden wiersz

Sekcja w pliku konfiguracyjnym projektu (np. `CLAUDE.md` dla asystenta Claude).

```
## Lessons Learned

- [YYYY-MM-DD] KONTEKST: Co poszło nie tak → Co robić zamiast tego.
- [YYYY-MM-DD] DOMENA: Konkretny przypadek → Konkretna reguła.
```

Zasady:

- Tylko konkretne, praktyczne lekcje (nie ogólniki w stylu "trzeba uważać")
- Maksimum 1-2 wpisy na sesję — kondensacja, nie elaborat
- Nie powtarzaj lekcji, które już są — sprawdź przed dopisaniem
- Przed rozpoczęciem nowej pracy ZAWSZE przeczytaj tę sekcję

2.3 Polecenie "adwokata diabła" do drugiego modelu AI

Wklej do drugiego AI (lokalnego Qwen, ChatGPT, Gemini — czego masz pod ręką) zanim zaczniesz pisać kod.

```
Przedstawiam Ci [plan / projekt / decyzję].

Twoja rola: doświadczony inżynier, który NIE chce, żeby to weszło
na produkcję bez krytycznej analizy. Zadaj mi pięć najtrudniejszych
pytań, które ujawnią luki, niedopowiedzenia lub nieprzemyślane
scenariusze w tym podejściu.

Bądź konkretny, nie ogólnikowy. Pytania mają być takie, że jeśli
nie umiem na nie sensownie odpowiedzieć, projekt się wywali w
jakimś momencie.

Po moich odpowiedziach zadaj pytania pogłębiające, jeśli któraś
odpowiedź wydaje się słaba lub niepełna.

[PLAN/PROJEKT]:
[wklej tutaj swój plan]
```

2.4 Dodanie kolumny zapisującej surowe odpowiedzi (SQL dla bazy danych)

> **Po ludzku:** ten kawałek SQL dodaje do tabeli w bazie danych nową "kolumnę" — taki słupek, w którym dla każdego rekordu zapisujesz pełną odpowiedź AI. Wykonujesz raz, na początku projektu.

```
-- Każda tabela rozmawiająca z AI dostaje taką kolumnę
ALTER TABLE [twoja_tabela]
  ADD COLUMN api_calls JSONB DEFAULT '[]'::jsonb;

-- Dodatkowy indeks (przyspiesza wyszukiwanie po zawartości)
CREATE INDEX idx_[twoja_tabela]_api_calls
  ON [twoja_tabela] USING GIN (api_calls);

-- Każdy zapis w kolumnie ma strukturę:
-- {
--   "step": "nazwa_etapu",
--   "ts": "2026-05-04T14:23:11Z",
--   "request": { ... co wysłałeś do AI ... },
--   "response": { ... co AI odpowiedziała ... },
--   "elapsed_ms": 8421
-- }
```

2.5 Funkcja zapisująca odpowiedzi do bazy (Python)

Mała funkcja w Pythonie, która "owija" wywołanie AI i automatycznie loguje wynik do bazy. Używasz raz, w jednym miejscu — potem każde wywołanie AI przez tę funkcję jest zapisywane.

```
import time, json, requests
from supabase import create_client

def call_api_with_logging(record_id: str, step: str,
                          url: str, payload: dict) -> dict:
    """Wywołaj AI i zapisz pełną odpowiedź do bazy."""
    t0 = time.time()
    response = requests.post(url, json=payload)
    elapsed_ms = int((time.time() - t0) * 1000)

    log_entry = {
        "step": step,
        "ts": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()),
        "request": payload,
        "response": response.json(),
        "elapsed_ms": elapsed_ms
    }

    supabase.rpc("append_api_call", {
        "rec_id": record_id,
        "log_entry": log_entry
    }).execute()

    return response.json()
```

---

3. Polecenia gotowe do uruchomienia

> Polecenia poniżej wpisuje się w terminal (Mac: aplikacja Terminal lub iTerm; Windows: PowerShell lub WSL). Każdy blok można skopiować w całości i wkleić.

3.1 Instalacja drugiej AI lokalnie (jednorazowo)

```
# Instalacja Ollama — aplikacja, która pozwala uruchamiać modele AI
# lokalnie na komputerze, bez internetu i opłat
curl -fsSL https://ollama.com/install.sh | sh

# Pobranie modelu Qwen (jednorazowo, ~22 GB)
ollama pull qwen3.6:35b-a3b

# Test — model powinien odpowiedzieć po polsku
ollama run qwen3.6:35b-a3b "Powiedz cześć po polsku."
```

3.2 Wywołanie "adwokata diabła" lokalnie

```
# Po instalacji Ollama działa lokalny serwis na porcie 11434
curl http://localhost:11434/api/generate -d '{
  "model": "qwen3.6:35b-a3b",
  "prompt": "Przedstawiam Ci plan implementacji. Twoja rola: senior engineer, który NIE chce, żeby to wes
  "stream": false
}' | jq -r '.response'

# Mała sztuczka: dopisek "/no_think" wyłącza tryb głośnego myślenia
# (inaczej Qwen generuje długi wewnętrzny monolog przed odpowiedzią)
```

3.3 Wybór pięciu produktów do testu (po typach, nie losowo)

```
-- SQL: kategoryzuj po typie i bierz po jednym z każdej kategorii
WITH categorized AS (
  SELECT id, opis_produktu,
    CASE
      WHEN opis_produktu ~* '\m[A-Z]{2,}-?[0-9]+' THEN 'konkretny_model'
      WHEN length(opis_produktu) < 50 THEN 'krotki_opis'
      WHEN opis_produktu ~* 'sugerowany|preferowany' THEN 'z_hintem'
      WHEN opis_produktu ~* 'zestaw|komplet|paczka' THEN 'edge_case_zestaw'
      ELSE 'ogolny_opis'
    END AS kategoria
  FROM twoja_tabela
  WHERE status = 'pending'
)
SELECT DISTINCT ON (kategoria) id, kategoria, opis_produktu
FROM categorized
ORDER BY kategoria, random()
LIMIT 5;
```

---

4. Drzewo decyzyjne — kiedy co robić

Sytuacja	Co zrobić
Mam ponad 5 produktów do przetworzenia naraz	Zacznij od próbki 5, dobranej po typach. Akceptacja wzrokowa. Dopiero potem reszta.
Mam 10-50 błędnych wyników do naprawy	Najpierw analiza zapisanych odpowiedzi i naprawa lokalnym skryptem. Ponowne uruchomienie AI tylko jeśli to nie wystarczy.
Mam plan, którego jeszcze nie pisałem w kodzie	Adwokat diabła z drugiego modelu AI. Pięć pytań. Odpowiedź na każde przed pierwszą linijką kodu.

Sytuacja	Co zrobić
Przerabiam coś, co już działa, na inną technologię	Najpierw odtwórz 1:1. Zmierz, że działa identycznie. Optymalizuj dopiero po weryfikacji.
Kończę sesję pracy	Krótka notatka do `.ai/handoffs/`. Konkretna pierwsza akcja w nowej sesji.
Złapałem błąd kosztujący więcej niż jedną próbę naprawy	Dopisz wpis do "Lessons Learned" w pliku konfiguracyjnym projektu. Jednolinijka.
Klient pyta "skąd ten konkretny wynik?"	Sprawdź zapisaną surową odpowiedź z bazy. Tam jest cała historia.
Mam ponad 30 minut przerwy w projekcie	Wpisz `wznów` w nowej sesji. Asystent załaduje najnowszą notatkę.

---

5. Plan startowy dla nowego projektu (30 minut)

Checklist do skopiowania. Po wykonaniu nowy projekt ma od pierwszego dnia: system notatek, zapis surowych odpowiedzi i drugi model AI do recenzji.

□ **(2 min)** Utwórz katalog `.ai/handoffs/` w repozytorium: `mkdir -p .ai/handoffs && touch .ai/handoffs/.gitkeep`

□ **(3 min)** Dodaj sekcję "Lessons Learned" do pliku konfiguracyjnego projektu (np. `CLAUDE.md` dla asystenta Claude, lub `README-AI.md`)

□ **(5 min)** Dodaj kolumnę zapisującą surowe odpowiedzi do każdej tabeli rozmawiającej z AI (SQL z sekcji 2.4)

□ **(5 min)** Wpnij wrapper logujący do swojego kodu (Python lub TypeScript, wzór w sekcji 2.5)

□ **(2 min)** Sprawdź, czy masz lokalną AI: `ollama list | grep qwen` — jeśli puste, zainstaluj według sekcji 3.1

□ **(5 min)** Zapisz pierwszą notatkę końca sesji (nawet krótką — żeby asystent w nowej sesji widział, że konwencja istnieje)

□ **(8 min)** Przed pierwszą funkcją — adwokat diabła z planem (wzór w sekcji 2.3)

Po tych 30 minutach masz fundamenty. Reszta przyjdzie w trakcie.

---

6. Co czytać dalej

- **Claude Code (Anthropic)** — narzędzie, które pozwala asystentowi AI Claude pracować z plikami na Twoim komputerze. Automatycznie wczytuje konfigurację projektu i globalną przy starcie sesji. Notatki, skille, pamięć — gotowe od razu.
- **Ollama** (`ollama.com`) — aplikacja, która pozwala uruchamiać modele AI lokalnie na komputerze. Darmowa, prywatna, działa szybko na nowszych Macach (M1/M2/M3 z 32+ GB RAM).
- **Supabase** (`supabase.com`) — baza danych z gotową infrastrukturą do zapisywania surowych odpowiedzi w formacie JSONB (kolumna, w której można zapisać dowolną strukturę). Bez dodatkowej konfiguracji.
- **Perplexity API** — usługa AI, która wyszukuje informacje w internecie. Z odpowiednimi parametrami (filtr domen, wymuszenie struktury odpowiedzi) rozwiązuje problem z pobieraniem cen ze sklepów blokujących boty.

Toolkit jest częścią artykułu "Sto godzin, 400 dolarów i 5 lekcji. Anatomia jednego narzędzia AI". Jeśli chcesz zrozumieć kontekst każdej rekomendacji — wróć do artykułu.