Agnieszka Janota
Scallora AI

Sebastian Kondracki
Deviniti / Bielik AI

PROMPTY W PRAKTYCE

automatyzacja i struktura

BIELIK.AI DEVINITI ScalloraAI MIND

Wstęp

Automatyzacja z wykorzystaniem dużych modeli językowych (LLM) to fascynujący, ale też wymagający kierunek rozwoju. W tradycyjnym dialogu z modelem mamy komfort korekty - możemy doprecyzować, poprawić, dopowiedzieć. Jednak w automatyzacji tego luksusu nie ma. Tutaj prompt musi działać za pierwszym razem.

W procesach takich jak integracje w n8n, UiPath, czy innych narzędziach automatyzujących, model nie jest naszym rozmówcą - jest komponentem systemu, który ma zwracać dane w sposób powtarzalny, przewidywalny i ustrukturyzowany. To właśnie dlatego tak ważne staje się zrozumienie formatu JSON, pracy ze schematami i budowania promptów, które gwarantują stabilność i jednoznaczność wyników.

Ta książka nie jest wprowadzeniem do promptowania. Zakładam, że potrafisz już skutecznie korzystać z modeli językowych. Teraz chcesz pójść krok dalej - przenieść swoje prompty z poziomu eksperymentów do poziomu automatyzacji, w której każda odpowiedź ma znaczenie dla działania większego procesu.

Nie będziemy bać się złożonych promptów. Wręcz przeciwnie - rozbudowane, dobrze przemyślane prompty są kluczem do jakości zarówno pod względem formy (strukturyzacji odpowiedzi), jak i merytorycznej spójności. Nauczysz się, jak zaprojektować je tak, aby model nie tylko rozumiał Twoje intencje, ale też przekładał je na konkretne, użyteczne dane.

„Prompty w praktyce: automatyzacja i struktura” to przewodnik dla tych, którzy chcą, by sztuczna inteligencja stała się integralnym elementem ich procesów - nie tylko inspiracją, ale i narzędziem do tworzenia prawdziwej, działającej automatyzacji.

Część I – Trochę o wydajniejszym promptowaniu

Promptowanie to nie tylko sztuka zadawania pytań. To przede wszystkim proces projektowania komunikacji z modelem, w którym precyzja i struktura decydują o jakości odpowiedzi. W tej części przyjrzymy się kilku sposobom na tworzenie promptów, które są bardziej powtarzalne, zrozumiałe i dają się łatwo przenosić do środowisk automatyzacji.

Wydajność w kontekście promptowania oznacza nie tylko szybsze uzyskanie właściwego wyniku, ale też mniejsze ryzyko niejednoznacznych odpowiedzi i większą kontrolę nad strukturą wyjścia. W automatyzacji to szczególnie ważne - model musi wiedzieć dokładnie, czego od niego oczekujesz.

Lekko o frameworkach promptowania

Jednym z najskuteczniejszych sposobów na poprawienie jakości promptów jest korzystanie z tzw. frameworków promptowania - czyli prostych schematów, które pomagają uporządkować sposób, w jaki formułujesz zadania dla modelu.

Frameworki te nie są sztywnymi zasadami - to raczej wzorce, które możesz dowolnie łączyć i dostosowywać do własnych potrzeb. Dzięki nim Twoje prompty stają się bardziej jednoznaczne, a model lepiej rozumie kontekst i oczekiwaną formę odpowiedzi.

RTF (Role – Task – Format)

To prosty, a jednocześnie bardzo skuteczny schemat, który pozwala jasno określić kontekst, zadanie i formę odpowiedzi.

Struktura:

- Role (Rola) – Kim ma być model
- Task (Zadanie) – Co ma zrobić
- Format (Format) – W jakiej formie ma zwrócić wynik

Przykład:

- **Rola:** Jesteś doświadczonym specjalistą ds. content marketingu.
- **Zadanie:** Przygotuj listę tematów na artykuły blogowe dotyczące trendów w automatyzacji pracy biurowej.
- **Format:** Zwróć wynik w formie listy punktowanej, każdy punkt to krótki tytuł i jednozdaniowy opis tematu.

Ten framework sprawdza się idealnie w automatyzacjach, ponieważ daje kontrolę zarówno nad tonem odpowiedzi, jak i nad strukturą danych, które można następnie łatwo wykorzystać w kolejnych etapach procesu. Warto jednak dobrze poznać format JSON, aby móc jednoznacznie zdefiniować oczekiwany sposób prezentacji wyników.

TAG (Task – Action – Goal)

Framework TAG jest prostszy i bardziej operacyjny. Świetnie nadaje się do krótszych, konkretnych zadań – np. w marketingu, HR czy analizie tekstu - gdzie zależy nam na szybkim efekcie, a nie na rozbudowanej strukturze promptu.

Struktura:

- Task (Zadanie) – Co ma zrobić model
- Action (Działanie) – Jak ma to zrobić
- Goal (Cel) – Jaki efekt końcowy chcemy uzyskać

Przykład:

- **Zadanie:** Przygotuj propozycję krótkiego posta promującego nowy kurs o automatyzacji procesów biurowych.
- **Działanie:** Użyj prostego języka, zachowaj ton ekspercki, dodaj wezwanie do działania.
- **Cel:** Otrzymać gotowy tekst, który można opublikować w mediach społecznościowych, zachęcający do zapisania się na kurs.

Framework TAG pomaga zachować klarowność intencji nawet w bardzo krótkim promptcie dokładnie określasz, co ma być zrobione, jak ma być zrobione i po co. Jednak wadą jego jest brak określonego formatu który jest kluczowy dla automatyzacji. Jednak są inne techniki które dowiesz się w następnych rozdziałach.

Inne popularne frameworki (do samodzielnego zgłębienia)

Jeśli chcesz poznać inne podejścia do strukturyzowania promptów, warto poszukać informacji o frameworkach takich jak:

- APE (Action, Purpose, Expectation),
- COAST (Context, Objective, Audience, Style, Tone),
- RACE (Role, Action, Context, Expectation),
- STAR (Situation, Task, Action, Result),
- TRACE (Task, Requirements, Audience, Context, Evaluation),
- ROSES (Role, Objective, Style, Example, Scenario).

Każdy z nich akcentuje coś innego - jedne pomagają w tworzeniu dłuższych, bardziej narracyjnych treści, inne skupiają się na budowaniu kontekstu lub precyzyjnym zdefiniowaniu kryteriów jakości odpowiedzi.

Nauczyliśmy się więc, że **trzymanie się struktury promptu to nie tylko kwestia porządku, ale i efektywności**. Dzięki temu praca automatyka czy twórcy agentów staje się prostsza, bardziej przewidywalna i skalowalna.

Pozostaje jednak pytanie: jak utrzymać tę strukturę, by zawsze wiedzieć, jakiego schematu użyliśmy, móc go łatwo rozbudowywać i ponownie wykorzystać? Tu właśnie z pomocą przychodzi markdown.

Markdown – porządek i czytelność w promptach

W świecie automatyzacji i pracy z dużymi modelami językowymi czytelność promptu to nie tylko kwestia estetyki - to sposób na lepsze zrozumienie polecenia przez człowieka i przez model. Markdown jest w tym kontekście idealnym narzędziem: lekki, prosty, powszechnie wspierany, a jednocześnie wystarczająco bogaty, by nadać strukturę i hierarchię nawet złożonym promptom.

Co to jest Markdown?

Markdown to lekki język znaczników służący do formatowania tekstu w sposób prosty i czytelny. Zamiast przycisków i pasków narzędzi (jak w Wordzie), używa prostych znaków takich jak: #, *, > czy ```, które dodają strukturę i styl bez ukrywania treści pod warstwą formatowania. Został stworzony z myślą o pisaniu czytelnych tekstów w surowej formie, które można łatwo konwertować na HTML, PDF czy inne formaty.

Dzięki temu stał się standardem w świecie programistów, dokumentacji technicznej i... coraz częściej także w promptowaniu.

Dlaczego Markdown jest lepszy niż Word w pracy z promptami?

- Czysty tekst, żadnych ukrytych znaczników
    - Word dodaje wiele niewidocznych formatowań (style, tagi XML, metadane), które mogą zaburzać działanie promptu, zwłaszcza w automatyzacji.
    - Markdown to czysty tekst - dokładnie to, co widzisz, to to, co wchodzi do modelu.
- Zachowuje strukturę w dowolnym środowisku
    - Plik .md otworzysz w dowolnym edytorze, narzędziu automatyzacyjnym, systemie kontroli wersji (np. GitHub), a także w zwykłym notatniku.
    - Dzięki temu Twoje prompty pozostają uniwersalne i przenośne.
- Idealny do wersjonowania i współpracy
    - W Markdownie łatwo porównywać różnice między wersjami promptów - linia po linii.
    - W Wordzie takie zmiany giną w warstwie wizualnej i trudno je śledzić w repozytorium.

- Minimalizm sprzyja skupieniu
    - Nie rozpraszasz się formatowaniem. Wszystko, co robisz, jest celowe - jeśli coś jest pogrubione, to dlatego, że jest naprawdę ważne dla promptu.
- Naturalnie współgra z kodem i danymi
    - Możesz w tym samym pliku mieć tekst, kod (```json, ```python), nagłówki i listy - czyli dokładnie to, czego potrzebujesz, tworząc prompty techniczne.

Markdown w praktyce promptowej

Markdown pozwala:

- utrzymać stałą strukturę promptu (np. nagłówki: Rola, Zadanie, Format),
- wyróżniać najważniejsze instrukcje dla modelu,
- prezentować przykłady danych w czytelny sposób,
- oznaczać ustrukturyzowane wyjście np. CSV, XML, JSON,
- a przede wszystkim – ułatwiać automatyzację dzięki przewidywalnej strukturze tekstu.

Poniżej znajduje się przykład prostego tekstu z wykorzystaniem znaczników Markdown. Jak widać, mimo obecności technicznych symboli, tekst pozostaje w pełni czytelny - łatwo można wychwycić, które elementy są ważne i z jakich sekcji się składa (np. oznaczonych nagłówkiem ###).

```
### Automatyzacja – więcej czasu na to, co naprawdę ważne

**Automatyzacja procesów biurowych** staje się jednym z
najważniejszych kierunków rozwoju współczesnych organizacji. Dzięki
niej firmy mogą *zredukować ilość rutynowych zadań* i skupić się na
działaniach, które przynoszą realną wartość – strategii, kreatywności
czy obsłudze klienta.

Automatyzacja nie oznacza zastąpienia człowieka, lecz **wsparcie jego
pracy**.
Systemy automatyzujące wykonują powtarzalne czynności szybciej,
dokładniej i bez błędów.
```

```
To z kolei przekłada się na:
- większą wydajność zespołów,
- mniejsze ryzyko błędów,
- lepsze wykorzystanie potencjału pracowników.

> *Każda minuta odzyskana dzięki automatyzacji to minuta, którą można poświęcić na rozwój.*

Podsumowując, automatyzacja to nie tylko technologia - to **zmiana podejścia do pracy**.
W świecie, w którym liczy się elastyczność i tempo, właśnie ona pozwala zachować równowagę między efektywnością a jakością.
```

Co ważne, w Internecie dostępnych jest wiele edytorów, w których można tworzyć tekst w formacie Markdown i jednocześnie na bieżąco podglądać jego sformatowaną wersję. Przykładowo:

https://markdownlivepreview.com/

```
### Automatyzacja – więcej czasu na to, co naprawdę ważne

**Automatyzacja procesów biurowych** staje się jednym z najważniejszych kierunków rozwoju współczesnych organizacji.
Dzięki niej firmy mogą *zredukować ilość rutynowych zadań* i skupić się na działaniach, które przynoszą realną wartość – strategii, kreatywności czy obsłudze klienta.

Automatyzacja nie oznacza zastąpienia człowieka, lecz **wsparcie jego pracy**.
Systemy automatyzujące wykonują powtarzalne czynności szybciej, dokładniej i bez błędów.
To z kolei przekłada się na:

- większą wydajność zespołów,
- mniejsze ryzyko błędów,
- lepsze wykorzystanie potencjału pracowników.

> *Każda minuta odzyskana dzięki automatyzacji to minuta, którą można poświęcić na rozwój.*

Podsumowując, automatyzacja to nie tylko technologia – to **zmiana podejścia do pracy**.
W świecie, w którym liczy się elastyczność i tempo, właśnie ona pozwala zachować równowagę między efektywnością a jakością.

```

Automatyzacja – więcej czasu na to, co naprawdę ważne

Automatyzacja procesów biurowych staje się jednym z najważniejszych kierunków rozwoju współczesnych organizacji.
Dzięki niej firmy mogą *zredukować ilość rutynowych zadań* i skupić się na działaniach, które przynoszą realną wartość – strategii, kreatywności czy obsłudze klienta.

Automatyzacja nie oznacza zastąpienia człowieka, lecz **wsparcie jego pracy**.
Systemy automatyzujące wykonują powtarzalne czynności szybciej, dokładniej i bez błędów.
To z kolei przekłada się na:

- większą wydajność zespołów,
- mniejsze ryzyko błędów,
- lepsze wykorzystanie potencjału pracowników.

> *Każda minuta odzyskana dzięki automatyzacji to minuta, którą można poświęcić na rozwój.*

Podsumowując, automatyzacja to nie tylko technologia — to **zmiana podejścia do pracy**.
W świecie, w którym liczy się elastyczność i tempo, właśnie ona pozwala zachować równowagę między efektywnością a jakością.

Rysunek 1. Zrzut ekranu z edytora Markdown Live Preview

No dobrze, pamiętacie pierwszy przykład promptu opartego na frameworku RTF (czyli Rola, Zadanie, Format)?

W markdown mógłby wyglądać tak:

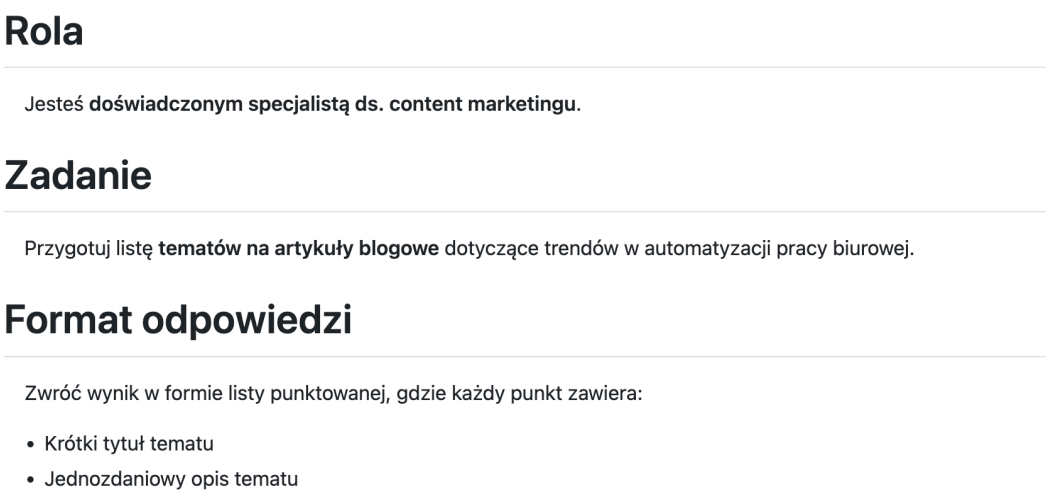
```
# Rola
Jesteś doświadczonym specjalistą ds. content marketingu.

# Zadanie
Przygotuj listę tematów na artykuły blogowe dotyczące trendów
w automatyzacji pracy biurowej.

# Format odpowiedzi
Zwróć wynik w formie listy punktowanej, gdzie każdy punkt zawiera:
- Krótki tytuł tematu
- Jednozdaniowy opis tematu
```

Oczywiście był to prosty, wręcz naiwny przykład - dlatego użycie markdowna może wydawać się tu trochę na wyrost. Jednak gdy tylko sprofesjonalizujemy nasz prompt,

całość zaczyna wyglądać znacznie lepiej: jest czytelna, łatwa do rozwoju i możliwa do ponownego wykorzystania nawet po dłuższym czasie.



Rysunek 2. **Wizualizacja prompta w edytorze Markdown Live Preview**

Oczywiście w promptowaniu zalecamy korzystanie z podstawowych znaczników, takich jak nagłówki, pogrubienia, kursywa, listy czy bloki kodu (np. z JSON-em). Markdown jednak oferuje znacznie więcej - umożliwia także proste osadzenie tabel, obrazów i linków. Zachęcam do zanurzenia się w świat prostych, ale niezwykle użytecznych znaczników formatujących.

Przykład prostej tabeli w Markdown:

```
| Imię      | Stanowisko        | Dział         |
|-----------|-------------------|---------------|
| Anna Kowalska | Specjalista ds. marketingu | Marketing |
| Jan Nowak     | Analityk danych          | R&D       |
| Piotr Wiśniewski | Kierownik projektu     | Sprzedaż  |
```

Imię	Stanowisko	Dział
Anna Kowalska	Specjalista ds. marketingu	Marketing
Jan Nowak	Analityk danych	R&D
Piotr Wiśniewski	Kierownik projektu	Sprzedaż

Rysunek 3. **Wizualizacja tabeli w edytorze Markdown Live Preview**

JSON – język, którym mówią automatyzacje

Automatyzacje, integracje i sztuczna inteligencja mają jedną wspólną cechę - wszystkie potrzebują danych. Ale same dane to za mało. Żeby mogły swobodnie przepływać między różnymi systemami, muszą być zapisane w sposób, który każdy z nich potrafi zrozumieć. Właśnie do tego służy JSON – prosty, czytelny i niezwykle elastyczny format zapisu informacji, który stał się uniwersalnym językiem współczesnych automatyzacji.

JSON to nie technologia dla programistów, ale narzędzie dla ludzi, którzy chcą, żeby systemy rozumiały się bez tłumacza.

Proste zasady - czym właściwie jest JSON

JSON (czyt. dżej-son) to sposób zapisywania danych w postaci tekstu - tak, aby zarówno komputer, jak i człowiek mogli je łatwo odczytać. Można pomyśleć o nim jak o arkuszu Excela zapisanym w formie tekstu. W Excelu masz kolumny (nazwy pól) i wiersze (wartości), a w JSON-ie - pary klucz-wartość, które pełnią dokładnie tę samą rolę.

Przykład:

```
{ "imię": "Anna", "wiek": 28 }
```

Gdybyśmy zapisali to w Excelu, wyglądałoby to tak:

imię	wiek
Anna	28

Tutaj:

- imię to nazwa kolumny (czyli klucz),
- "Anna" to wartość w tej kolumnie,
- wiek to kolejne pole z wartością 28.

To właśnie zasada klucz-wartość - fundament, na którym opiera się JSON.

Dlaczego więc nie używać po prostu Excela?

Bo Excel to zamknięty plik, z własnym formatem i strukturą, którego inne systemy nie zawsze potrafią odczytać. JSON natomiast jest otwartym tekstem - dane w nim zapisane można przesyłać między aplikacjami, systemami czy modelami językowymi bez utraty struktury. Każdy, kto ma dostęp do pliku lub odpowiedzi, może je zobaczyć, odczytać, przetworzyć lub zintegrować z kolejnym narzędziem.

JSON jest więc uniwersalnym językiem wymiany danych, którym „rozmawiają" ze sobą automatyzacje, aplikacje i modele AI. Kiedy system sklepu wysyła dane do CRM-u albo chatbot przekazuje wynik analizy, bardzo często robi to właśnie w tym formacie.

Obiekty - dane opisane nazwami

Obiekt w JSON to zestaw informacji opisujących jeden element - na przykład produkt, klienta lub zamówienie. Można myśleć o nim jak o wierszu w Excelu, wizytówce lub formularzu z polami do wypełnienia.

Przykład prostego obiektu:

```
{
  "nazwa": "Laptop",
  "cena": 4200,
  "dostępny": true
```

```
}
```

Ten obiekt opisuje jeden produkt. Każde pole ma nazwę i przypisaną wartość:

- "nazwa" – to tekst,
- "cena" – to liczba,
- "dostępny" – to wartość logiczna (tak/nie).

W arkuszu Excela wyglądałoby to tak:

nazwa	cena	dostępny
Laptop	4200	tak

Każdy taki wiersz w arkuszu to właśnie pojedynczy obiekt JSON.

Nazwane obiekty – czyli dane z kontekstem

W praktyce bardzo często obiekt nie występuje „luzem", lecz jest nazwany – stanowi część większej struktury. Wtedy obiekt staje się wartością pola (klucza), co pozwala określić jego znaczenie.

Przykład:

```
{
  "klient": {
    "imię": "Anna",
    "nazwisko": "Kowalska",
    "wiek": 28
  }
}
```

Tutaj:

- główny klucz to "klient",
- jego wartością jest obiekt zawierający dane tej osoby.

To bardzo popularny zapis, ponieważ:

- nadaje danym kontekst (wiemy, że to klient, a nie np. pracownik czy dostawca),
- pozwala tworzyć zagnieżdżone struktury,
- ułatwia automatyzacjom i API rozpoznawanie typu danych bez dodatkowych opisów.

W Excelu można by to porównać do osobnego arkusza o nazwie Klient, w którym znajdują się kolumny imię, nazwisko i wiek – to ta sama idea, tylko w wersji tekstowej.

Lista - czyli wiele obiektów w jednym miejscu

Czasem potrzebujemy zapisać więcej niż jeden obiekt – np. listę produktów, klientów czy faktur. Wtedy używamy listy (tablicy), czyli zbioru kilku obiektów o tej samej strukturze.

Przykład:

```
{
  [
    { "nazwa": "Laptop", "cena": 4200 },
    { "nazwa": "Monitor", "cena": 890 },
    { "nazwa": "Klawiatura", "cena": 220 }
  ]
}
```

lub w przypadku tzw. obiektów nazwanych:

```
{
  "produkty": [
    { "nazwa": "Laptop", "cena": 4200 },
    { "nazwa": "Monitor", "cena": 890 },
    { "nazwa": "Klawiatura", "cena": 220 }
  ]
}
```

To po prostu lista trzech produktów. Każdy element jest osobnym obiektem, ale wszystkie mają te same pola. W Excelu wyglądałoby to tak:

nazwa	cena
Laptop	4200
Monitor	890
Klawiatura	220

W JSON taka struktura to lista obiektów - bardzo wygodna forma, gdy przesyłamy dane między aplikacjami lub systemami automatyzacji.

Kilka bardziej rozbudowanych przykładów

JSON pozwala też łączyć różne struktury – np. w jednym obiekcie umieścić listę lub inny obiekt. Dzięki temu może przypominać miniaturową bazę danych, w której dane są logicznie powiązane i czytelnie zorganizowane.

Warto jednak wiedzieć, że w praktyce klucze (nazwy pól) zapisujemy po angielsku. To standard w świecie automatyzacji i API – angielskie słowa są jednoznaczne, nie mają odmiany i działają w każdym systemie. Wartości natomiast mogą być w dowolnym języku, w tym po polsku.

Przykład:

```
{
  "customer": {
    "name": "Anna Kowalska",
    "orders": [
      {
        "id": 101,
        "product": "Laptop",
        "status": "wysłane"
      },
      {
        "id": 102,
        "product": "Myszka",
        "status": "oczekujące"
```

```
        }
      ]
    }
}
```

Tutaj:

- główny obiekt to customer (klient),
- pole orders zawiera listę jego zamówień,
- każda pozycja w liście (order) to osobny obiekt z danymi produktu i statusem.

Ten sposób zapisu jest najbardziej uniwersalny. Dzięki użyciu angielskich nazw pól, taki JSON:

- można łatwo połączyć z dowolnym API lub narzędziem automatyzacji,
- będzie poprawnie interpretowany przez modele językowe,
- zachowa pełną czytelność – nawet dla osób nietechnicznych, bo wartości nadal mogą być po polsku.

Taki zapis można traktować jak „międzynarodowy język danych" - systemy rozumieją klucze, a ludzie widzą czytelne wartości.

Praca z JSON-em jest prostsza, niż się wydaje

Nawet jeśli na początku struktura JSON-a wygląda trochę technicznie, nie ma się czego bać. Większość edytorów tekstowych (np. Visual Studio Code, Sublime, Notepad++) oraz same modele językowe – takie jak ChatGPT – automatycznie kolorują składnię JSON-a, dzięki czemu od razu widzisz, które elementy są kluczami, a które wartościami. To sprawia, że praca z JSON-em jest przejrzysta i intuicyjna.

Rysunek 4. **Wizualizacja struktury JSON-a w ChatGPT**

W Internecie znajdziesz też wiele walidatorów JSON – darmowych narzędzi online, w których możesz wkleić swój kod i sprawdzić, czy struktura jest poprawna. Takie narzędzia (np. jsonlint.com czy jsonformatter.org) pomogą Ci szybko znaleźć drobne błędy, np. brakujący przecinek lub nawias.

Promptowanie i JSON

Kiedy zaczynamy automatyzować pracę z dużymi modelami językowymi, zauważamy, że sam tekst przestaje wystarczać. Chcemy, by model nie tylko napisał coś sensownego, ale też zwrócił wynik w ustrukturyzowany sposób, który można od razu wykorzystać w dalszym procesie - np. w n8n, UiPath czy Make. Najprostszy sposób, by to osiągnąć, to połączenie promptu z JSON-em.

Zamiast prosić o zwykły tekst, określamy w promptcie, że wynik ma zostać zwrócony w formacie JSON - czyli w postaci danych, które system automatyzacji odczyta bez problemu. Aby model dobrze to zrozumiał, wystarczy użyć znaczników Markdown i bloku kodu z oznaczeniem: `` ```json [twój json] ``` ``.

Przykład:

```
Zwróć wynik w formacie JSON:

```json
{

}
```
```

Dzięki temu model wie, że ma przygotować dane w strukturze, a nie w formie tekstowej odpowiedzi. To drobny szczegół, ale ma ogromne znaczenie - poprawia spójność, ułatwia testowanie i pozwala wpiąć wynik w kolejne etapy automatyzacji.

Przykład: jak przekształcić klasyczny prompt w wersję JSON-ready

Weźmy prosty przykład promptu w formacie RTF (Rola, Zadanie, Format):

```
Jesteś doświadczonym specjalistą ds. content marketingu. Przygotuj
listę tematów na artykuły blogowe dotyczące trendów w automatyzacji
pracy biurowej. Zwróć wynik w formie listy punktowanej - każdy punkt
to krótki tytuł i jednozdaniowy opis tematu.
```

Ten prompt jest poprawny, ale nieprzydatny w automatyzacji - wynik będzie po prostu tekstem. Przekształćmy go tak, by model zwrócił dane w formacie JSON, gotowe do dalszego przetwarzania:

```
Jesteś doświadczonym specjalistą ds. content marketingu.Przygotuj
listę tematów na artykuły blogowe dotyczące trendów w automatyzacji
pracy biurowej. Zwróć wynik w formacie JSON. Każdy element listy
powinien mieć dwa pola:

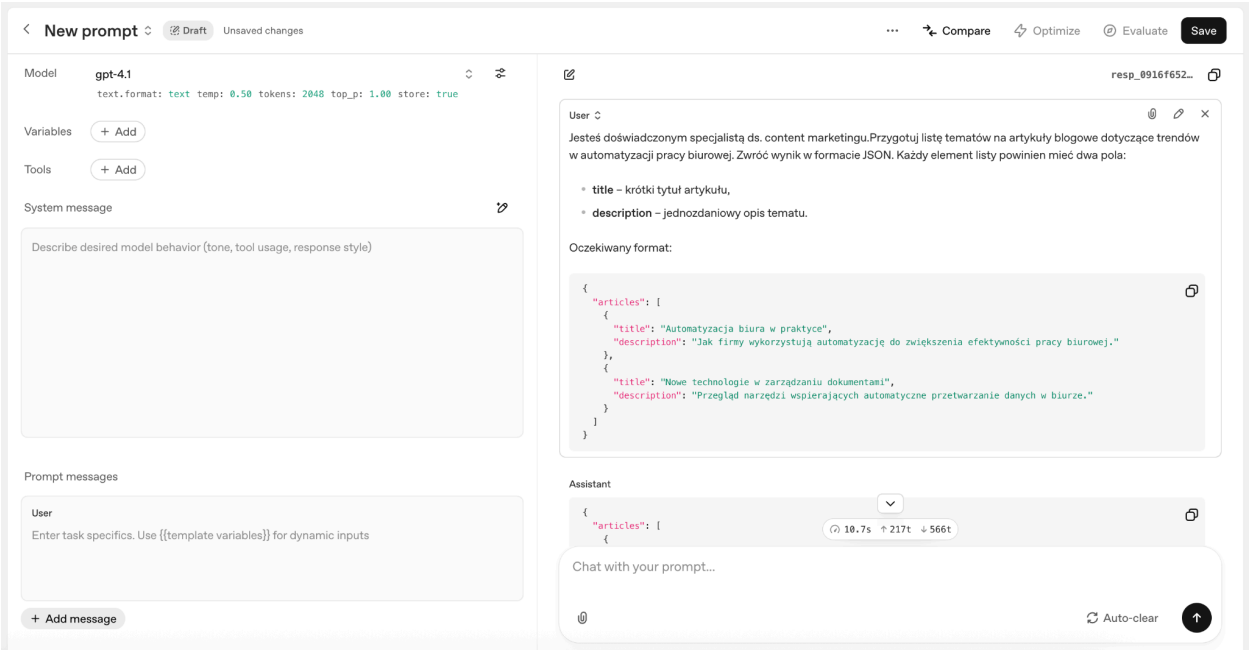
- **title** – krótki tytuł artykułu,
- **description** – jednozdaniowy opis tematu.

Oczekiwany format:
```json
{
  "articles": [
    {
      "title": "Automatyzacja biura w praktyce",
```
```

```
      "description": "Jak firmy wykorzystują automatyzację do
zwiększenia efektywności pracy biurowej."
    },
    {
      "title": "Nowe technologie w zarządzaniu dokumentami",
      "description": "Przegląd narzędzi wspierających automatyczne
przetwarzanie danych w biurze."
    }
  ]
}
```

Dlaczego to działa

- Markdown pomaga utrzymać porządek - zarówno człowiek, jak i model widzą dokładnie, jak ma wyglądać wynik.
- Oznaczenie bloku jako json powoduje, że modele automatycznie zwracają dane w prawidłowej składni JSON.
- Ustrukturyzowane wyjście można natychmiast wykorzystać w automatyzacji - np. przesłać do bazy danych, arkusza, CRM-u albo jako dane wejściowe do kolejnego kroku procesu.
- Klarowność i standaryzacja - wiesz, że każda odpowiedź ma tę samą strukturę, więc nie musisz jej ręcznie poprawiać.



Rysunek 5. **Wizualizacja promptu z blokiem json w środowisku OpenAI Platform**

Na ilustracji widać, jak prompt został czytelnie sformatowany w Markdownie (np. pogrubione podpunkty), a sam blok kodu oznaczony jako ***json*** spowodował automatyczne kolorowanie składni. Dzięki temu zarówno treść promptu, jak i zwrócony wynik - również w formacie JSON - są przejrzyste i łatwe do analizy.

Kolorowanie składni od razu pozwala odróżnić klucze od wartości, co znacząco ułatwia dalszą integrację i pracę z danymi.

Niestety, nawet najlepiej napisany prompt nie zawsze gwarantuje perfekcyjny wynik. Modele językowe (LLM) potrafią czasem zwrócić JSON z drobnymi błędami składni, takimi jak dodatkowy przecinek, brak cudzysłowów czy niezamknięty nawias. To zupełnie normalne - pamiętaj, że model nie jest parserem, tylko generatorem tekstu, który uczy się wzorców. W takich sytuacjach z pomocą przychodzi ***JSON Schema*** oraz ***mechanizmy tzw. ustrukturyzowanego wyjścia***, które pozwalają narzucić modelowi dokładny format danych. Dzięki nim możemy mieć pewność, że wynik będzie poprawny technicznie i gotowy do użycia w automatyzacji - o czym szerzej w kolejnych rozdziałach.

Na tym etapie **najważniejsze to ćwiczyć promptowanie z JSON-em**. Twórz różne struktury, testuj ich złożoność, zmieniaj nazwy pól i eksperymentuj z formatowaniem. Im częściej będziesz to robić, tym szybciej zrozumiesz, jak model „myśli" o strukturze danych - i jak sprawić, by pisał dokładnie tak, jak tego potrzebujesz.

Meta-promptowanie - prompt, który tworzy prompty

W pewnym momencie pracy z dużymi modelami językowymi zaczynamy zauważać, że nie chodzi już tylko o tworzenie pojedynczych promptów, ale o projektowanie sposobu, w jaki je tworzymy. Tu właśnie pojawia się pojęcie meta-promptowania - czyli tworzenia promptów, które pomagają nam pisać... jeszcze lepsze prompty.

Meta-prompt to nic innego jak „prompt-architekt". Taki, który pomaga Ci doprecyzować opis zadania, zasugerować strukturę odpowiedzi, a nawet sformatować wszystko w Markdownie lub JSON-ie. To praktyczne narzędzie, które uczy Cię myśleć tak, jak model: krok po kroku, logicznie i strukturalnie.

Lepszy opis zadania

Zaczynamy od tego, co najtrudniejsze: opisania, czego naprawdę chcemy. Meta-prompt może Ci w tym pomóc, zadając pytania uzupełniające lub prosząc o doprecyzowanie celu. Zamiast samodzielnie tworzyć rozbudowany opis, możesz po prostu poprosić model:

```
Pomóż mi sformułować prompt, który opisze analizę opinii klientów w
jasny i precyzyjny sposób
```

Model zaproponuje Ci gotowy szkielet, który potem możesz łatwo dostosować do swoich potrzeb. Aby nadać tej strukturze większą spójność, warto od razu dodać w poleceniu, że prompt ma być zbudowany zgodnie z frameworkiem RTF (Role, Task, Format). Przykładowe polecenie może wyglądać tak:

```
Pomóż mi stworzyć prompt w frameworku Role Task Format, który opisze analizę opinii klientów - uwzględniając rolę, zadanie i format wyjścia
```

Dzięki temu model nie tylko precyzuje treść, ale też uporządkuje prompt w logiczną strukturę, którą później łatwo rozwijasz, modyfikujesz lub automatyzujesz.

Automatyczne formatowanie w Markdown

Gdy masz już opis zadania, meta-prompt może pomóc Ci w uporządkowaniu jego struktury. Wystarczy dodać prośbę:

```
Sformatuj powyższy prompt w Markdown, z nagłówkami Rola, Zadanie, Format
```

W efekcie otrzymujesz czytelny, dobrze zorganizowany prompt, który możesz natychmiast wkleić do systemu lub repozytorium .md. To nie tylko poprawia przejrzystość, ale też ułatwia dalszą automatyzację.

Sprawdzenie struktury JSON-a na wyjściu

Na koniec meta-prompt może pełnić funkcję walidatora - poproszony o analizę wygenerowanego JSON-a, potrafi wskazać błędy składni, brakujące klucze czy niespójne typy danych. Przykładowe polecenie:

```
Sprawdź, czy poniższy JSON jest poprawny i zgodny z zasadami struktury obiektu.
```

Model potrafi nawet automatycznie poprawić format, dodać brakujące nawiasy lub zaproponować lepszą strukturę.

Meta-promptowanie w praktyce – rozmowa z modelem

Największą siłą meta-promptowania jest to, że możesz tworzyć i doskonalić swoje prompty w dialogu z modelem, np. w narzędziu takim jak ChatGPT lub Claude Sonnet. Nie musisz pisać wszystkiego od razu - możesz prowadzić rozmowę, w której wspólnie z modelem:

- doprecyzowujesz założenia,
- pytasz o dobre praktyki,
- testujesz różne wersje,
- prosisz o sugestie dotyczące struktury, formatu czy tonacji.

To proces interaktywny - model staje się Twoim współautorem i doradcą. Dzięki temu powstają prompty lepiej dopasowane do kontekstu, spójne i zgodne z najlepszymi praktykami.

Dlaczego to ważne

Meta-promptowanie pozwala przejść na wyższy poziom pracy z AI. Zamiast tworzyć każdy prompt ręcznie, zaczynasz budować system, który wspiera Twój sposób myślenia. To trochę tak, jakbyś nauczył model pełnić rolę Twojego redaktora, inżyniera i testera jednocześnie. W efekcie Twoje prompty stają się spójne, lepiej przemyślane i technicznie przygotowane do automatyzacji. To pierwszy krok w stronę projektowania promptów jako procesów, a nie pojedynczych poleceń.

Ale nic tak dobrze nie podsumowuje tego rozdziału jak praktyczny przykład. Pamiętacie wcześniejszy prompt:

```
Jesteś doświadczonym specjalistą ds. content marketingu. Przygotuj
listę tematów na artykuły blogowe dotyczące trendów w automatyzacji
pracy biurowej. Zwróć wynik w formacie JSON ...
```

A teraz spójrzmy na to z perspektywy meta-promptowania. Zamiast pisać taki prompt samodzielnie, możemy po prostu poprosić model o jego stworzenie. Wystarczy opisać zadanie, wskazać, jakie założenia ma mieć wynikowy JSON, i pozwolić modelowi wygenerować resztę. W praktyce może to wyglądać tak:

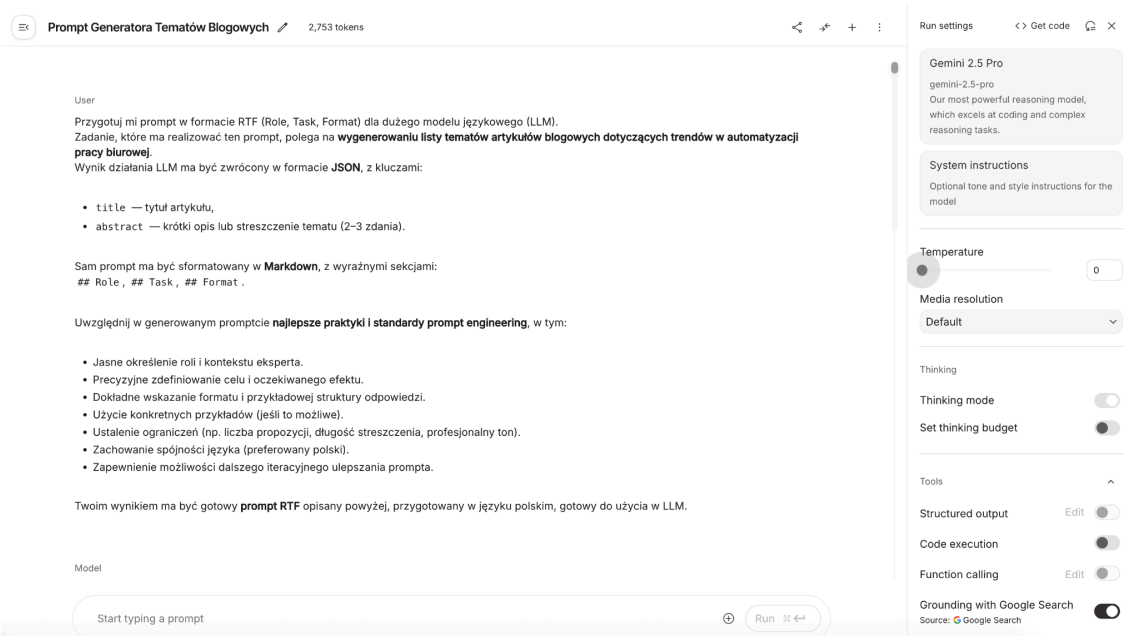
```
Przygotuj mi prompt w formacie RTF (Role, Task, Format) dla dużego
modelu językowego (LLM).
Zadanie, które ma realizować ten prompt, polega na **wygenerowaniu
listy tematów artykułów blogowych dotyczących trendów w automatyzacji
pracy biurowej**.
Wynik działania LLM ma być zwrócony w formacie **JSON**, z kluczami:
- `title` - tytuł artykułu,
- `abstract` - krótki opis lub streszczenie tematu (2–3 zdania).

Sam prompt ma być sformatowany w **Markdown**, z wyraźnymi sekcjami:
`## Role`, `## Task`, `## Format`.

Uwzględnij w generowanym promptcie **najlepsze praktyki i standardy
prompt engineering**, w tym:
- Jasne określenie roli i kontekstu eksperta.
- Precyzyjne zdefiniowanie celu i oczekiwanego efektu.
- Dokładne wskazanie formatu i przykładowej struktury odpowiedzi.
- Użycie konkretnych przykładów (jeśli to możliwe).
- Ustalenie ograniczeń (np. liczba propozycji, długość streszczenia,
profesjonalny ton).
- Zachowanie spójności języka (preferowany polski).
- Zapewnienie możliwości dalszego iteracyjnego ulepszania prompta.

Twoim wynikiem ma być gotowy **prompt RTF** opisany powyżej,
przygotowany w języku polskim, gotowy do użycia w LLM.
```

Model sam zbuduje cały prompt w odpowiedniej strukturze (np. RTF), doda instrukcje dotyczące formatu i zadba o spójność poleceń. Spróbujmy zatem czegoś nowego - tym razem przetestujmy Google AI Studio i model Gemini 2.5 Pro. Po wklejeniu meta-promptu do piaskownicy widzimy, że edytor automatycznie sformatował Markdown, dzięki czemu struktura promptu jest od razu przejrzysta i gotowa do użycia.



Rysunek 6. **Środowisko Google AI Studio**

Naciskamy „Run” i po chwili otrzymujemy gotowy prompt - poprawnie sformatowany, czytelny i w pełni przygotowany do automatyzacji.

```
## Rola

Jesteś doświadczonym analitykiem trendów technologicznych i
strategiem content marketingu, specjalizującym się w automatyzacji
procesów biznesowych (BPA) i robotyzacji (RPA). Twoim zadaniem jest
identyfikowanie najbardziej angażujących i przyszłościowych tematów,
które przyciągną uwagę menedżerów, dyrektorów IT oraz specjalistów
ds. operacyjnych, poszukujących innowacyjnych rozwiązań dla swoich
organizacji.

## Zadanie

Twoim zadaniem jest wygenerowanie listy **5-7 unikalnych i
interesujących tematów na artykuły blogowe**. Tematy te powinny
dotyczyć najnowszych i przyszłych trendów w automatyzacji pracy
biurowej. Skup się na innowacjach, praktycznych zastosowaniach,
wpływie technologii na rynek pracy oraz strategicznych korzyściach
dla przedsiębiorstw.
```

````
Każda propozycja musi zawierać:
1.  **Chwytliwy tytuł (`title`)**, który wzbudzi ciekawość i będzie
zoptymalizowany pod kątem wyszukiwarek.
2.  **Krótki abstrakt (`abstract`)**, składający się z 2–3 zdań,
który wyjaśnia, co czytelnik znajdzie w artykule i dlaczego jest to
dla niego wartościowe.

Utrzymuj profesjonalny, ale przystępny i angażujący ton. Unikaj
ogólników i skup się na konkretnych, wartościowych informacjach.

## Format

Odpowiedź zwróć **wyłącznie w formacie JSON**. Wynik powinien być
listą obiektów, gdzie każdy obiekt reprezentuje jeden temat artykułu
i zawiera klucze `title` oraz `abstract`. Nie dodawaj żadnych
dodatkowych wyjaśnień, wstępów ani tekstu przed lub po bloku kodu
JSON.

**Przykład struktury odpowiedzi:**

```json
[
  {
    "title": "Hiperautomatyzacja: Więcej niż RPA. Jak przygotować
firmę na kolejny krok w ewolucji automatyzacji?",
    "abstract": "Artykuł wyjaśnia, czym jest hiperautomatyzacja i
jakie technologie, takie jak AI i machine learning, łączy w sobie,
aby wyjść poza tradycyjne RPA. Czytelnicy dowiedzą się, jak
strategicznie wdrożyć to podejście, aby osiągnąć skokowy wzrost
efektywności operacyjnej. Analizujemy kluczowe wyzwania i korzyści na
podstawie rzeczywistych case studies."
  },
  {
    "title": "Citizen Developers: Czy pracownicy bez wiedzy IT
zautomatyzują Twoje biuro? Potencjał platform Low-Code/No-Code",
    "abstract": "Platformy low-code i no-code demokratyzują dostęp do
automatyzacji, pozwalając pracownikom biznesowym tworzyć własne
rozwiązania. W artykule omawiamy, jak zjawisko 'citizen development'
zmienia oblicze firm i jakie narzędzia warto wdrożyć. Przedstawiamy
również, jak zarządzać tym procesem, aby zapewnić bezpieczeństwo i
spójność technologiczną."
  }
]```
````
````

To właśnie esencja meta-promptowania: uczenie modelu, by tworzył narzędzia, które później wspierają Twoją pracę. Jeśli masz uwagi do wygenerowanego promptu - dyskutuj, dopytuj, iteruj. Przykładowo, nie przepadam za liczbowymi wskazówkami „składający się z 2–3 zdań". Zamiast limitu długości lepiej dodać wartość porównawczą, która ukierunkowuje długość treści.

Zamiast:

„Napisz opis składający się z 2–3 zdań."

Ulepsz:

„Napisz zwięzły opis z takimi elementami jak ..."

Klucz jest prosty: eksperymentuj - zarówno z meta-promptem, jak i promptami, które model dla Ciebie tworzy. Iteruj drobnymi krokami, dodawaj porównania, liczby i kryteria sukcesu, a szybko zobaczysz, jak rośnie jakość i użyteczność wyników.

Część II – Automatyzacja

Kiedy nauczymy się pisać przemyślane i dobrze sformatowane prompty, naturalnym kolejnym krokiem jest zautomatyzowanie ich działania. Nie chodzi już tylko o to, by model generował odpowiedź, ale by potrafił ją przekazać dalej - do kolejnego kroku procesu, integracji, raportu czy systemu. Aby to było możliwe, potrzebujemy dwóch rzeczy:

1. **Kontekstu** - czyli informacji, które model powinien znać przed wykonaniem zadania,
2. **Ustrukturyzowanego wyjścia** - czyli danych, które można automatycznie przetworzyć.

Kontekst – czyli co model powinien wiedzieć

Każdy prompt działa lepiej, gdy model zna kontekst - czyli tło, dane lub cel, do którego ma się odnieść. To właśnie kontekst nadaje sens odpowiedzi: bez niego model „zgaduje", z nim - rozumie po co i dla kogo coś robi. **W świecie automatyzacji kontekst ma jednak szczególne znaczenie. Nie chodzi już tylko o pojedynczy przypadek użycia, ale o masowe przetwarzanie danych, w którym jeden prompt jest wykorzystywany wielokrotnie - z różnymi zestawami informacji.**

W praktyce oznacza to, że do jednego promptu może trafiać setki, a nawet tysiące różnych kontekstów:

- jednego dnia mogą to być fragmenty kilku raportów, które trzeba streścić,
- innego - zestawy opinii klientów, które należy pogrupować lub zanalizować,
- a w e-commerce - nawet setki tysięcy opisów produktów, które trzeba przekształcić, przetłumaczyć lub ujednolicić.

Dlatego w automatyzacji prompt przestaje być pojedynczym poleceniem, a staje się szablonem, do którego dynamicznie wstawiamy zmienny kontekst.

To kluczowa zmiana myślenia: Nie piszesz promptu dla jednego przypadku projektujesz go tak, by mógł pracować dla setek danych wejściowych dziennie.

Właśnie dlatego tak ważne jest, by kontekst był czytelnie oddzielony i jasno zdefiniowany - np. przez sekcję KONTEKST: w Markdownie lub pole "context" w strukturze JSON.

W praktyce wygląda to tak, że programista lub inżynier automatyzacji bierze nasz prompt i umieszcza go w kodzie - np. w Pythonie. Skrypt pobiera dokument, wyodrębnia tekst, łączy go z promptem, wysyła zapytanie do modelu LLM i zapisuje wynik w formacie JSON w bazie danych. Takie operacje mogą być wykonywane setki lub tysiące razy dziennie, przetwarzając raporty, wiadomości czy dane produktowe. Tę samą logikę można zrealizować bez kodowania, np. w narzędziach takich jak n8n czy UiPath.

Wtedy tworzymy flow, które automatycznie:

- odczytuje nowe dokumenty,
- przekazuje ich treść jako kontekst do agenta AI,
- uruchamia prompt,
- a wynik (np. streszczenie, analiza, rekomendacja) zapisuje w określonym miejscu.

Dzięki temu prompt działa jak silnik inteligentnego przetwarzania danych - zasilany kolejnymi porcjami kontekstu.

Przykład prostego kontekstu w Markdown:

```
KONTEKST:
Firma zajmuje się wdrażaniem narzędzi do automatyzacji procesów
biurowych.
Głównym celem jest pokazanie, jak nowoczesne technologie mogą
oszczędzać czas i zredukować koszty.
```

W takim podejściu kontekst może pochodzić z wielu różnych źródeł - w zależności od tego, co analizuje lub przetwarza Twoja automatyzacja.

Najczęściej są to:

- dane o firmie, produkcie lub użytkowniku,
- ekstrakcje z dokumentów (czyli czysty tekst uzyskany np. z plików PDF lub systemu OCR),
- treści wiadomości e-mail lub zapisy rozmów,
- dane wejściowe z arkuszy, CRM-ów, API lub baz danych.

Wszystkie te elementy mogą być dynamicznie przekazywane do promptu jako zmienny kontekst. Model w każdym wywołaniu otrzymuje inny zestaw danych, ale korzysta z tego samego szablonu promptu - dzięki czemu cały proces jest powtarzalny, zautomatyzowany i łatwy do kontrolowania.

Kiedy już wiemy, jak przekazać kontekst, pora połączyć go z właściwym promptem i wyjściem w JSON. Załóżmy, że masz fragment (ekstrakcję) raportu o trendach w pracy zdalnej i chcesz, aby model wypunktował najważniejsze tezy - w formacie JSON.

Tworzymy zatem prompt typu RTF (Rola, Zadanie, Format) i dodajemy sekcje KONTEKST

```
# ROLA:
Jesteś analitykiem biznesowym specjalizującym się w ekstrakcji
kluczowych informacji z raportów korporacyjnych.

# ZADANIE:
Wypunktuj najważniejsze tezy z poniższego fragmentu (KONTEKST),
przedstaw je w zwięzłej formie i zwróć wynik w formacie JSON.

# FORMAT:
Zwróć wynik w następującej strukturze JSON:
```json
{
  "key_points": [
    "Przykładowa teza 1",
```
```

```
    "Przykładowa teza 2",
    "Przykładowa teza 3"
  ]
}
```

```
# KONTEKST:
"""W 2025 roku automatyzacja biurowa stanie się kluczowym elementem
pracy zdalnej. Firmy inwestują w narzędzia do zarządzania przepływem
dokumentów, raportowaniem czasu pracy i komunikacją wewnętrzną.
Z badań wynika, że 67% firm planuje zwiększyć budżety na
automatyzację procesów administracyjnych."""
```

Ten prompt został zaprojektowany tak, aby był czytelny, modularny i gotowy do automatyzacji. Każdy element pełni tu konkretną funkcję i został umieszczony w określonej kolejności nieprzypadkowo.

Połącz zadanie z kontekstem

Sekcja ZADANIE powinna jasno wskazywać, jak model ma pracować z kontekstem. Zamiast ogólnego *„streść tekst”*, lepiej napisać: *„Wypunktuj najważniejsze tezy z poniższego fragmentu (KONTEKST), przedstaw je w zwięzłej formie i zwróć wynik w formacie JSON.”*

Taka konstrukcja sprawia, że model wie:

- z której części promptu ma pobrać dane (sekcja KONTEKST),
- co z nimi zrobić (analiza, streszczenie, ekstrakcja informacji),
- oraz w jakiej formie ma zwrócić wynik (JSON).

W automatyzacji to kluczowe - prompt nie zostawia miejsca na domysły, tylko precyzyjnie definiuje proces przetwarzania danych.

Umieść kontekst na końcu

Kontekst to zmienna część promptu, dlatego warto dać na końcu. Stała pozostaje struktura: rola, zadanie i format (lub inny framework). Dzięki temu prompt można wykorzystywać wielokrotnie, a jedynie podmieniać treść w sekcji KONTEKST.

To podejście pozwala:

- łatwo zautomatyzować przetwarzanie setek dokumentów,
- utrzymać spójność i powtarzalność procesu,
- w prosty sposób testować różne źródła danych.

W skrócie: **część stała (logika) u góry, część zmienna (dane) na dole.**

Wydziel kontekst w czytelny sposób

Sekcja KONTEKST została ujęta w potrójne cudzysłowy """ ... """. To prosty sposób na oddzielenie dłuższej treści od instrukcji i zachowanie porządku w prompcie. Dzięki temu model łatwiej rozpoznaje, gdzie kończy się polecenie, a zaczyna kontekst.

Ten zapis ma kilka zalet:

- Potrójne cudzysłowy to zapożyczenie z języków programowania, takich jak Python.
- Pozwalają uniknąć błędów podczas wklejania lub formatowania treści, które zawierają cudzysłowy ("), ponieważ nie trzeba ich poprzedzać znakiem ukośnika (\").

To tzw. „stara szkoła promptowania", ale wciąż bardzo przydatna - szczególnie wtedy, gdy nie korzystasz z Markdowna lub pracujesz w środowisku tekstowym (np. w edytorze kodu, terminalu). Potrójne cudzysłowy pomagają zachować przejrzystość i uniknąć błędów składni, które często pojawiają się przy dużych promptach zawierających cytaty, znaki specjalne lub długie konteksty.

Alternatywnie możesz użyć:

- bloku Markdown (```text)
- albo JSON-a (```json) – gdy dane przychodzą z systemu i chcesz je przetwarzać automatycznie.

Ustrukturyzowane wyjście i JSON Schema – czyli pełna kontrola nad danymi

Kiedy zaczynamy łączyć promptowanie z automatyzacją, szybko okazuje się, że nie chodzi już tylko o to, co model wygeneruje, ale w jakiej formie to zrobi. Jeśli wynik ma zostać automatycznie zapisany w bazie danych, przekazany do CRM-u czy arkusza, to musi być technicznie poprawny, spójny i przewidywalny.

I tu właśnie spotykają się dwie techniki:

- prompt w formacie JSON, który pozwala jasno określić strukturę wyjścia,
- oraz JSON Schema, które dodatkowo wymusza, by model trzymał się tej struktury.

W połączeniu dają nam pełną kontrolę nad danymi - model nie tylko tworzy treść, ale też sam waliduje jej techniczną poprawność. To kolejny krok w stronę promptów, które nie tylko generują tekst, ale zachowują się jak część systemu.

JSON Schema – czyli jak opisać strukturę danych

Skoro wiemy już, że JSON to sposób zapisu danych, to JSON Schema jest jego „instrukcją obsługi". Można o nim myśleć jak o szablonie albo formularzu, który określa, jakie pola powinny się znaleźć w danych, jaki mają mieć typ (np. tekst, liczba, lista) i które są obowiązkowe.

Innymi słowy:

- JSON to dane,
- JSON Schema to opis tych danych - ich struktury i zasad.

Prosty przykład

Załóżmy, że model ma zwracać informacje o produkcie: jego nazwę, cenę i dostępność.

Wtedy JSON Schema może wyglądać tak:

```
{
  "type": "object",
  "properties": {
    "product_name": { "type": "string" },
    "price": { "type": "number" },
    "available": { "type": "boolean" }
  },
  "required": ["product_name", "price", "available"]
}
```

Ten schemat mówi modelowi:

- wynik ma być obiektem (***object***),
- musi zawierać trzy pola: ***product_name***, ***price***, ***available***,
- każde z pól ma określony typ danych (tekst, liczba, wartość logiczna),
- które pola są wymagane, które opcjonalne.

W efekcie model wie dokładnie, jak ma wyglądać odpowiedź. Przykładowy wynik, zgodny z tym schematem:

```
{
  "product_name": "Laptop X100",
  "price": 4299.99,
  "available": true
}
```

Dlaczego to takie ważne

W automatyzacji i integracjach JSON Schema pełni podobną rolę jak formularz w aplikacji: nie pozwala zapisać błędnych danych.

Dzięki niej:

- model nie doda przypadkowego przecinka,
- nie pomyli tekstu z liczbą,
- i nie pominie wymaganego pola.

Dla automatyzacji to ogromna różnica - bo poprawne dane można od razu przekazać do kolejnych kroków procesu (bazy danych, CRM-u, raportu itp.), bez ręcznego sprawdzania.

Ustrukturyzowane wyjście w aplikacji

Warto pamiętać, że JSON Schema i tzw. ustrukturyzowane wyjście nie są częścią samego promptu. To funkcja aplikacji, która uruchamia model - to ona „pilnuje", by dane zwrócone przez model miały dokładnie taki kształt, jaki został określony w schemacie. Oznacza to, że po naszej stronie - w prompcie - nadal możemy prosić o wynik w formacie JSON, ale to aplikacja zajmuje się sprawdzeniem, czy model rzeczywiście zwrócił poprawny i kompletny wynik. Jeśli nie - automatycznie poprosi go o poprawkę i zwróci prawidłową odpowiedź.

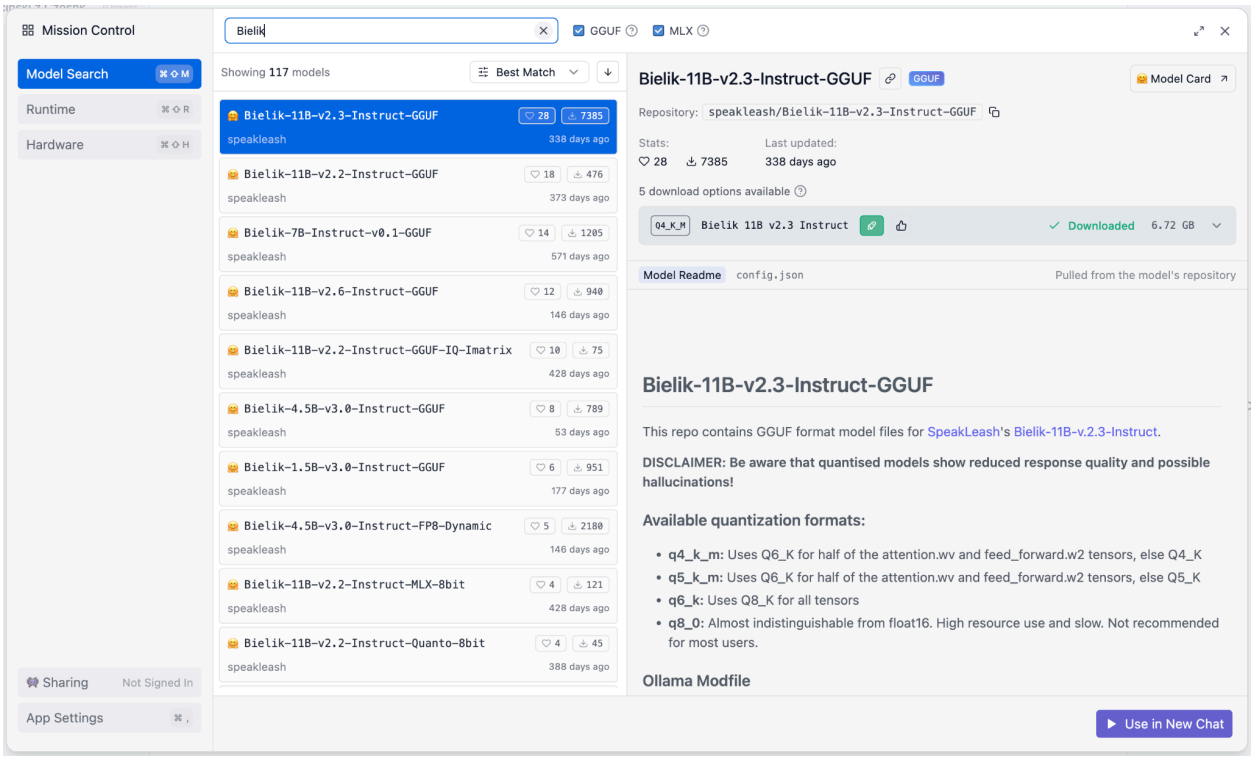
Taką możliwość mają między innymi popularne narzędzia, z których korzystają twórcy automatyzacji: LM Studio, Ollama, OpenAI Platform, Gemini Studio, czy narzędzia oparte na bibliotekach takich jak LangChain i LlamaIndex. W każdym z nich można wkleić lub załączyć własny plik ze schematem JSON i uruchomić model w trybie Structured Output - czyli z walidacją odpowiedzi. Dzięki temu nie trzeba już martwić się o brakujący przecinek czy cudzysłów. To aplikacja przejmuje kontrolę nad poprawnością, a my możemy skupić się na samym promptowaniu i projektowaniu procesu.

Przykład: LM Studio i ustrukturyzowane wyjście

Zacznijmy od prostego przykładu. Załóżmy, że chcesz, aby model rozpoznawał z krótkiego opisu nazwę produktu i jego cenę.

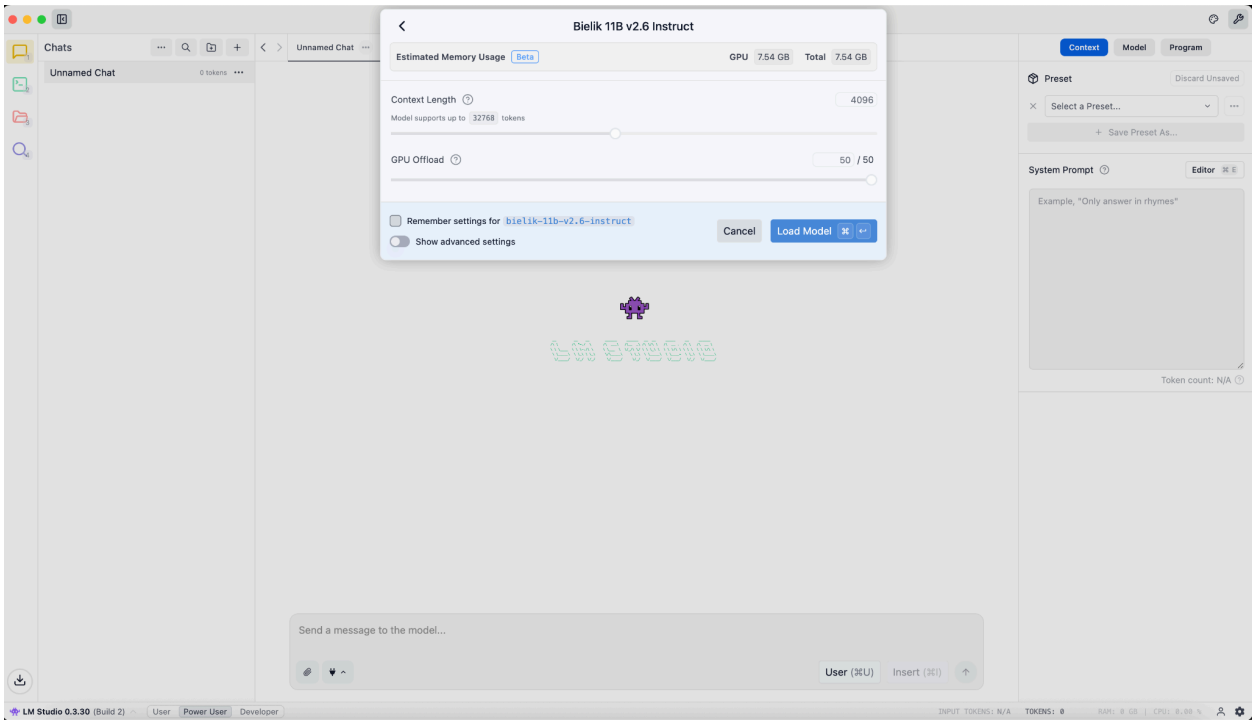
Zainstaluj **LM Studio** - pobierz aplikację ze strony lmstudio.ai i uruchom ją na swoim komputerze.

Następnie wybierz opcję **„Discover”** (ikona lupki), wyszukaj model, np. **Bielik**, i wybierz **Bielik-11B-v2.6-Instruct-GGUF**. Kliknij **„Download”** i poczekaj, aż model się pobierze.



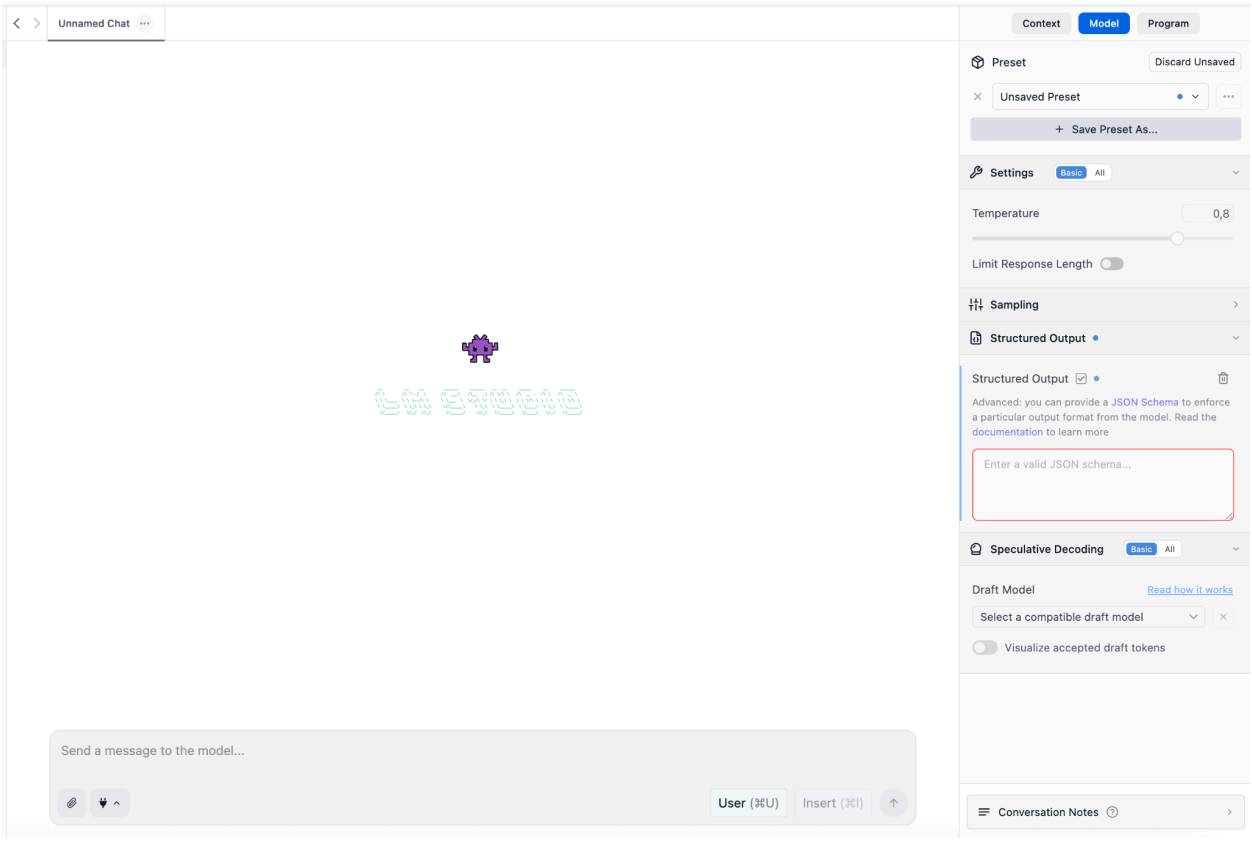
Rysunek 7. **LM Studio - wyszukiwanie modelu**

Gdy pobieranie się zakończy, wybierz **„Use in new chat”** lub **„Create new chat”**, a następnie kliknij **„Load model”**, by załadować go do okna rozmowy.



Rysunek 8. **LM Studio - wybór modelu**

Po prawej stronie, w tzw. sidebarze, otwórz zakładkę **„Model”** i zaznacz przycisk **„Structured output”**.



Rysunek 9. **LM Studio - aktywacja "Structured Output"**

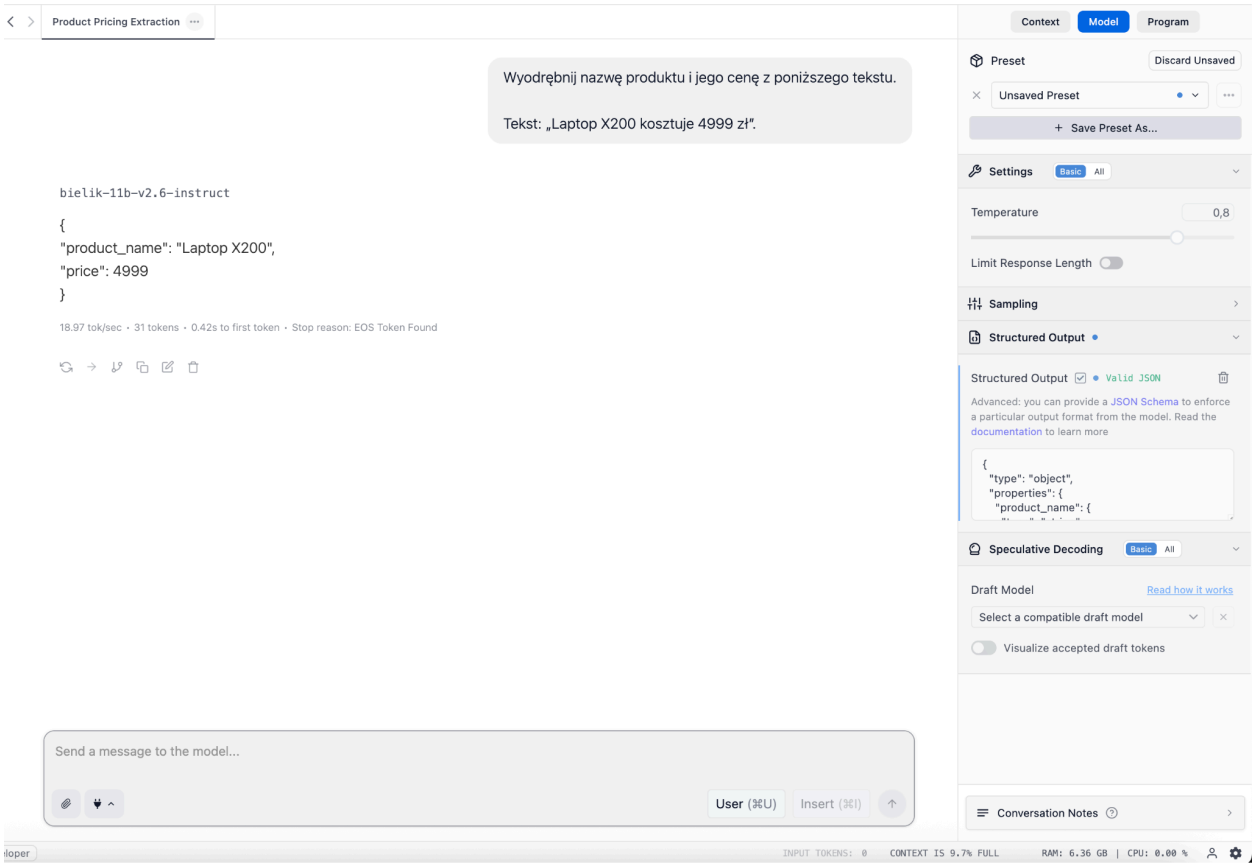
W okno tekstowe poniżej wklej swoją JSON Schema.

```
{
  "type": "object",
  "properties": {
    "product_name": {
      "type": "string"
    },
    "price": {
      "type": "number"
    }
  },
  "required": [
    "product_name",
    "price"
  ]
}
```

Teraz w okienku czatu wpisz najprostszy prompt, aby przetestować, czy wszystko działa.

```
Wyodrębnij nazwę produktu i jego cenę z poniższego tekstu.
Tekst: „Laptop X200 kosztuje 4999 zł”.
```

I proszę - otrzymaliśmy wynik w poprawnej, ustrukturyzowanej formie JSON.



Rysunek 10. **LM Studio - gotowa odpowiedź**

Zwróć uwagę, że nie musiałeś już dodawać w prompcie instrukcji typu „zwróć wynik w JSON” - aplikacja sama pilnuje struktury danych. To właśnie jest ustrukturyzowane

wyjście w praktyce.

Łączenie promptu z JSON-em i aplikacji z JSON Schema

Najlepsze efekty w pracy z dużymi modelami uzyskujemy wtedy, gdy połączymy obie techniki: prompt, który jasno określa format odpowiedzi, oraz aplikację, która pilnuje, by ten format był zawsze zachowany.

W praktyce wygląda to tak:

- w samym prompcie pokazujemy modelowi, jak ma wyglądać wynik,
- a w aplikacji, w której model działa, podajemy JSON Schema, które definiuje techniczną strukturę tych danych.

Dzięki temu mamy pełną kontrolę nad procesem:

- prompt ustala logikę i strukturę komunikacji,
- a aplikacja gwarantuje techniczną poprawność i spójność wyników.

To rozwiązanie szczególnie ważne w automatyzacjach, gdzie model uruchamiany jest dziesiątki czy setki razy dziennie. Dzięki połączeniu promptu z JSON-em oraz aplikacji z JSON Schema unikamy błędów, zyskujemy powtarzalność i możemy bezpiecznie łączyć modele z innymi systemami - od CRM-ów po arkusze i raporty.

Podsumowanie – od pomysłu do gotowego promptu

Przeszliśmy już przez wszystkie najważniejsze elementy skutecznego promptowania - od frameworków i struktury, przez Markdown i JSON, aż po automatyzację i ustrukturyzowane dane. Czas zatem zebrać to wszystko w jeden, praktyczny proces.

Po pierwsze, ustal kontekst

Zastanów się, z jakim materiałem ma pracować model: czy to będzie jedna strona raportu, cały dokument, fragment JSON-a, HTML, czy może po prostu czysty tekst. Kontekst to Twoje dane wejściowe - bez niego nawet najlepiej napisany prompt nie ma sensu.

Po drugie, zaprojektuj strukturę wyniku

Zastanów się, jak chcesz, by wyglądała odpowiedź: czy potrzebujesz listy punktów, obiektu z nazwanymi polami, czy może pełnej tabeli. Zapisz to w formacie JSON, określając pola i ich typy. Gotowy JSON możesz sprawdzić w jednym z darmowych walidatorów online - upewnisz się, że ma poprawną składnię i strukturę.

Po trzecie, dobierz framework promptowania

Dla większości przypadków doskonale sprawdza się RTF (Role Task Format) - prosty i logiczny. Zdefiniuj, kto jest Twoim asystentem (rola), jakie ma zadanie (task) i w jakim formacie ma zwrócić wynik (format).

Po czwarte, stwórz meta-prompt

To Twój pomocnik - prompt, który pomaga Ci pisać lepsze prompty. Poproś model, by zbudował dla Ciebie kompletny prompt w stylu np. Role Task Format a w opisie zadania postaraj się być jak najbardziej precyzyjny - typ/format kontekstu, cel, ograniczenia i oczekiwany styl odpowiedzi. Gdy masz już zarys promptu, poproś model o sformatowanie go w Markdownie. To drobiazg, ale znacząco poprawia czytelność i

pozwala Ci później łatwo odnaleźć strukturę, nawet po dłuższym czasie.

Po piąte, testuj.

Uruchom LM Studio lub dowolną piaskownicę modelu (np. OpenAI Platform, Google AI Studio) i wklej swój prompt wraz z przykładowym tekstem, raportem czy obiektem JSON. Sprawdź, czy model zwraca dane dokładnie w takiej strukturze, jak zaplanowałeś. Jeśli coś nie działa - popraw sekcję „Format" w prompcie lub doprecyzuj polecenie w „Task". To normalna część procesu: prompt to mały system, który ewoluuje.

Po szóste, przetestuj na wielu kontekstach

Weź kilkanaście różnych przykładów - różne długości, języki, formaty tekstu i sprawdź, czy Twój prompt nadal daje spójne wyniki. Dzięki temu upewnisz się, że jest odporny na zmiany i nadaje się do automatyzacji.

Na koniec, stwórz JSON Schema dla swojego wyjścia

To schemat, który dokładnie opisuje, jak mają wyglądać dane zwracane przez model i który możesz wykorzystać w aplikacjach takich jak LM Studio, Ollama, czy integracjach API.

Od tej chwili masz wszystko:

- prompt z dobrze opisanym zadaniem,
- czytelny format w Markdownie,
- strukturę danych w JSON,

oraz schemat JSON Schema, który gwarantuje poprawność techniczną. I to właśnie jest kompletne podejście do pracy z dużymi modelami językowymi w automatyzacji. Zaczynasz od prostego pomysłu, a kończysz na rozwiązaniu, które można uruchomić setki razy dziennie. Zawsze z tym samym, przewidywalnym i poprawnym wynikiem.

Checklist: Twój gotowy prompt do automatyzacji

1. **Ustal kontekst**

☐ Wybierz, co model ma analizować - jedna strona raportu, cały dokument, JSON czy czysty tekst.

☐ Upewnij się, że dane są dobrze przygotowane (bez błędów, OCR, duplikatów).

2. **Zaprojektuj strukturę wyniku**

☐ Zastanów się, jakie pola mają się pojawić w odpowiedzi.

☐ Zapisz przykład w formacie JSON.

☐ Sprawdź go w darmowym walidatorze online (np. jsonlint.com).

3. **Dobierz framework promptowania**

☐ Wybierz strukturę - najlepiej RTF (Role - Task - Format).

☐ Określ rolę modelu, zadanie i format zwracanych danych.

4. **Stwórz meta-prompt**

☐ Poproś model, aby pomógł Ci zbudować pełny prompt w stylu RTF.

☐ Doprecyzuj opis zadania i poproś o sformatowanie w Markdownie.

5. **Testuj w piaskownicy lub LM Studio**

☐ Wklej prompt i przetestuj go na kilku przykładowych tekstach lub obiektach.

☐ Sprawdź, czy wynik jest zgodny z Twoim JSON-em.

☐ Wprowadź drobne poprawki w sekcji Task lub Format, jeśli trzeba.

6. **Utwórz JSON Schema i włącz Structured Output**

☐ Zbuduj schemat JSON, który opisuje strukturę danych.

☐ Włącz tryb Structured Output w aplikacji (np. LM Studio, Ollama, OpenAI Platform).

☐ Uruchom test końcowy - wynik powinien być poprawny technicznie i spójny

logicznie.

https://markdownlivepreview.com/	Markdown Live Preview to darmowy, internetowy edytor Markdown, który nie wymaga logowania ani instalacji. Wystarczy wpisać tekst w lewej części ekranu, a po prawej stronie od razu zobaczysz jego sformatowany podgląd - dokładnie tak, jak w edytorze tekstu typu Word. Świetny do nauki i testowania formatowania promptów przed użyciem ich w automatyzacjach.
https://jsonlint.com/	JSONLint to prosty walidator online do sprawdzania poprawności plików JSON. Wklejasz swój kod, klikasz Validate JSON, i natychmiast widzisz, czy składnia jest poprawna. Świetny sposób, by upewnić się, że Twój promptowy JSON lub JSON Schema nie zawiera błędów.
https://platform.openai.com/playground	OpenAI Playground to interaktywne środowisko testowe dla modeli GPT. Pozwala eksperymentować z różnymi stylami promptów, ustawiać temperaturę, długość odpowiedzi oraz format wyjścia. Idealne do szybkiego sprawdzenia działania promptu, zanim włączysz go do automatyzacji.
https://aistudio.google.com/	Google AI Studio to platforma internetowa do pracy z modelami m.in. Gemini, Nano Banana. Pozwala tworzyć, testować i udoskonalać prompty w przeglądarce, bez potrzeby programowania. Możesz eksperymentować z różnymi wersjami modeli Gemini, zapisywać prompty, testować format JSON oraz korzystać z gotowych przykładów. Świetne narzędzie do nauki i szybkiego prototypowania - szczególnie, gdy chcesz sprawdzić, jak model radzi sobie z ustrukturyzowanym wyjściem lub meta-promptami.

https://lmstudio.ai/	LM Studio to darmowa aplikacja desktopowa, która pozwala uruchamiać lokalnie duże modele językowe (LLM). Nie wymaga połączenia z Internetem - możesz pobierać i testować modele takie jak Mistral, Llama, czy Bielik bezpośrednio na swoim komputerze. Wspiera Structured Output, co pozwala testować prompty z JSON Schema w pełnym środowisku automatyzacji.

Zakończenie

Promptowanie, nawet to najbardziej techniczne i ustrukturyzowane, nie jest nauką ścisłą - to sztuka komunikacji z modelem, w której teoria ma sens tylko wtedy, gdy ją sprawdzasz w praktyce. Ta książka miała Ci pokazać, że skuteczność nie wynika z magii ani z „sekretnych promptów", ale z świadomego projektowania i konsekwentnego eksperymentowania. Każdy framework, każde formatowanie w Markdown, każdy plik JSON - to tylko narzędzia, które pozwalają Ci szybciej dojść do momentu, w którym model zaczyna naprawdę rozumieć Twój zamiar.

Automatyzacja i metapromptowanie otwierają ogromne możliwości - od prostych przepływów w n8n, po rozbudowane systemy wspierające procesy biznesowe w sprzedaży, marketingu, HR czy R&D. Ale prawdziwa wartość nie tkwi w gotowych schematach. Tkwi w eksperymencie: w tym, że **przetestujesz, zmienisz, ulepszysz** - aż prompt zadziała dokładnie tak, jak chcesz.

- Nie bój się błędów.
- Nie bój się zbyt rozbudowanych promptów.
- Nie bój się wracać do tego samego problemu z nowym pomysłem.

Każdy „nieidealny" wynik to dane do nauki - informacja zwrotna, która pozwala Ci lepiej zrozumieć zarówno model, jak i własny proces myślenia.

- Eksperymentuj z formą.
- Twórz łańcuchy promptów i automatyzacji.
- Testuj różne formaty danych.
- Twórz prompty, które generują inne prompty.

I niech każdy kolejny projekt, flow czy automatyzacja stanie się dowodem na to, że najlepszy prompt to ten, który powstał w wyniku prób, błędów i ciekawości.