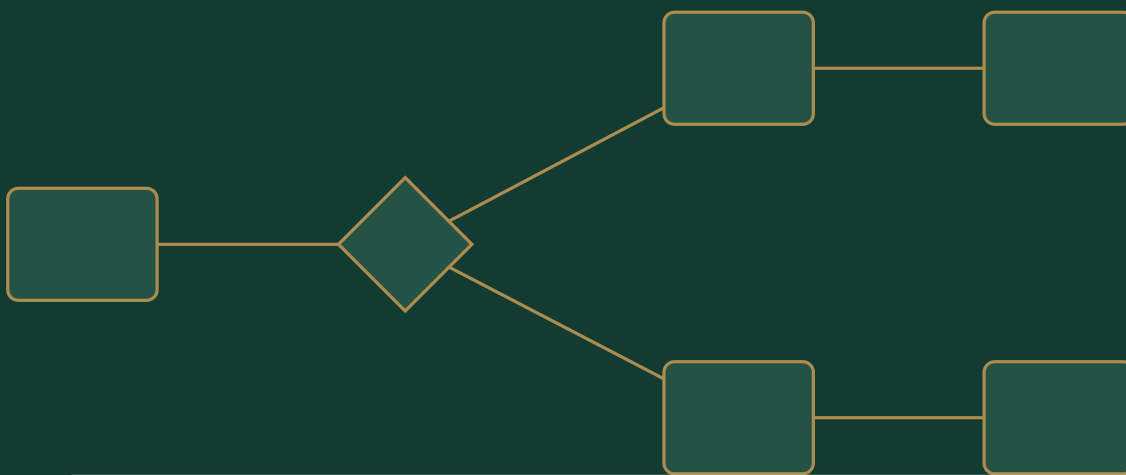


INTELIGENTNY WORDPRESS

Automatyzacje, API i agenci AI



PROJEKT TWORZY CZŁOWIEK.
AI WSPIERA PROCES.

10 ROZDZIAŁÓW / JEDEN SYSTEM DEMONSTRACYJNY

Spis treści

Wprowadzenie i mapa projektu	3
01 Kontrakt automatyzacji i granice odpowiedzialności	4
Zaprojektuj wynik przed połączeniem narzędzi · Kontrakt, który nie zależy od platformy · Sekrety i dane mają własną granicę	
02 REST API i uprawnienia integracji	7
Dwa modele uwierzytelnienia REST · Capability opisuje dozwoloną operację · Trasę REST rejestruj jawnie	
03 Formularz oraz walidacja wejścia	10
Formularz zamawia zadanie · Schemat zamknięty ogranicza niespodzianki · Walidacja ma być czytelna w kodzie	
04 Kolejka zadań i trwała tożsamość	13
Tożsamość zadania przeżywa restart · Rezerwacja pracy jest atomowa · Stan kolejki prowadzi operatora	
05 AI na serwerze: adapter i kontrola wyniku	16
MOCK domyślny, OpenAI opcjonalnie · Wynik AI jest niezaufaną propozycją · HTML potrzebuje dwóch kontroli	
06 Wtyczka intake tworząca wyłącznie szkice	19
Trwały klucz w bazie WordPressa · Serwer wymusza draft przy zapisie · Potwierdzenie musi być sprawdzone	
07 Retry, nieznanый wynik i uzgodnienie	22
Retry dotyczy znanego odrzucenia · Timeout nie dowodzi braku szkicu · Uzgodnienie jest pracą z dowodami	
08 n8n, Make, importy i harmonogramy	25
n8n i Make odwzorowują kontrakt · Import jest serią zleceń, nie skrótem · Harmonogram nie jest gwarancją czasu	
09 Obserwowalność, testy i bezpieczeństwo operacyjne	28
Loguj przebieg, nie sekret · Testy offline odpowiadają na konkretne pytania · Staging sprawdza granice WordPressa	
10 Odbiór projektu i praca redaktora	31
Odbierz kompletny projekt · Redaktor sprawdza treść i pochodzenie · Rozwijaj jedną granicę naraz	
Źródła techniczne i dalsza lektura	34

Kliknij rozdział. Nazwa tomu w stopce prowadzi do spisu. Pełne pliki techniczne i instrukcje odtworzenia są w EXAMPLES.

Szkic jest wynikiem, publikacja jest decyzją

W tym tomie zbudujesz kompletny przepływ demonstracyjny: formularz przyjmuje dane, wykonawca przygotowuje treść, WordPress zapisuje szkic, a redaktor świadomie zatwierdza publikację.

Automatyzacja treści zaczyna się od ustalenia granicy odpowiedzialności. Nasz formularz nie zamawia opublikowanego artykułu. Zamawia przygotowanie szkicu o znanym pochodzeniu. Każde zlecenie ma UUID, ograniczone pola i jeden dozwolony cel zapisu. Dzięki temu łatwiej odpowiedzieć na pytania: kto uruchomił zadanie, jaki tekst dotarł do WordPressa i gdzie system zatrzymał się po błędzie. Nie dodajemy funkcji publikowania do konta integracji.

WP Demo Lab korzysta z fikcyjnych treści, jawnego deterministycznego adaptera MOCK i lokalnego wykonawcy Python. W pakiecie znajdziesz formularz HTML, lokalny gateway, kolejkę SQLite, wtyczkę wp-demo-intake, neutralny kontrakt workflow i testy z lokalnym serwerem HTTP. Każdy element jest samodzielny. Do części offline wystarczy Python; WordPress na stagingu uruchomisz według README, bez posiadania wcześniejszych tomów. Minimum funkcji projektu nie oznacza rekomendacji używania starego rdzenia: wybierz aktualne wspierane wydanie i zapisz jego wersję.

Czytaj ten tom jako warsztat projektowania zachowania systemu. Połączenie dwóch API to dopiero środek pracy. Trudniejsze decyzje dotyczą walidacji, duplikatów, ponowienia po timeout, przechowywania sekretów i odpowiedzialności redaktora. Baza oraz endpoint ograniczają powtarzanie, ale nie gwarantują dokładnie jednego skutku we wszystkich awariach i hooks. Stan uncertain jest uczciwą informacją, że wynik wymaga uzgodnienia.

Testy offline rzeczywiście uruchomiono. Nie uruchomiono PHP, WordPressa, n8n, Make ani rzeczywistego modelu AI. Rozdzielamy te dowody w całym tomie. Twoim końcowym zadaniem będzie przeprowadzenie testów stagingowych, zapisanie wyników oraz przyjęcie szkicu jako redaktor. Dopiero wtedy podejmiesz decyzję, czy zakres demonstracyjny można rozwijać w kierunku własnego środowiska.

- Pracuj na danych syntetycznych, kopii stagingowej i kontach z rozdzielonymi rolami.
- Zachowuj UUID oraz historię stanów; nie resetuj ich po niejasnym błędzie.
- Pełne pliki i instrukcje znajdują się w EXAMPLES; fragmenty w książce objaśniają wybrane decyzje.

Zaprojektuj wynik przed połączeniem narzędzi

Przepływ ma jeden mierzalny rezultat techniczny: istnieje identyfikowalny szkic. Jakość tekstu oraz zgoda na publikację pozostają etapem redakcyjnym.

Zacznij od zdania określającego rezultat: poprawne zlecenie tworzy szkic, który redaktor może porównać z wejściem. To zdanie wyznacza interfejs i testy. Gdy dopiszesz automatyczne publikowanie, pobieranie dowolnych adresów albo aktualizowanie istniejących wpisów, zmieniasz zakres uprawnień i ryzyko. W tym projekcie te operacje nie należą do kontraktu, więc nie ukrywamy ich w adapterze AI ani w module HTTP.

Zlecenie niesie `request_id`, `title`, `content` i stałe `source`. Treść jest demonstracyjnym opisem, a nie instrukcją administracyjną. WordPress ma otrzymać przygotowany HTML i wymusić status `draft` niezależnie od oczekiwań wykonawcy. Bot może wywołać `intake`, ale nie ma `publish_posts`. Redaktor pracuje w standardowym panelu i ponosi decyzję o publikacji. Rozdzielenie kont umożliwia praktyczne sprawdzenie tej granicy: błędna ścieżka klienta nadal nie daje botowi przywileju publikacji.

Zapisz również rezultat negatywny. Niepoprawny schemat nie trafia do kolejki, konflikt UUID nie zamienia treści starego zadania, a nieznany wynik nie uruchamia nowej próby tworzenia. To ważne kryteria jakości, nawet gdy na ekranie nie pojawia się nowy wpis. W dokumentacji odbioru oznacz osobno sukces szkicu, odrzucenie danych oraz sprawę wymagającą uzgodnienia. Nie oceniaj całości wyłącznie liczbą wykonanych workflow.

Przed startem wyznacz operatora kolejki i osobę zatwierdzającą tekst. Jedna osoba może pełnić obie funkcje na lokalnym laboratorium, lecz protokół powinien wskazywać, kiedy wykonuje czynność techniczną, a kiedy redakcyjną.

Od zlecenia do akceptacji

01 Formularz

Syntetyczne dane i UUID

02 Walidacja

Schemat oraz ograniczenia

03 AI + kolejka

Kontrolowany HTML, trwałe zadanie

04 Intake

WordPress zapisuje draft

05 Redaktor

Porównanie, korekta i decyzja

PRO / AUTOMATYZACJE

Kontrakt, który nie zależy od platformy

Najpierw opisz zachowanie danych i stanów. Dopiero później wybierz, czy zadanie wykona Python, n8n, Make czy infrastruktura zespołu.

Plik workflow-contract.json jest mapą zasad, a nie eksportem platformy. Zawiera wymagane pola, stałą wartość source, limity bajtów UTF-8, model uprawnień i stany kolejki. Jego wartość polega na tym, że możesz porównać różne implementacje bez zgadywania, co znaczą nazwy węzłów. Wersja kontraktu określa zachowanie projektu; numer aplikacji automatyzacyjnej określa środowisko, w którym próbujesz je odwzorować.

Nie myl syntaktycznej poprawności JSON z gotową integracją. Dokument może dać się odczytać, a nadal nie gwarantować transakcyjnego klucza, bezpiecznego retry czy odpowiedniej retencji logów. Wykonawca referencyjny ma te decyzje w kodzie. Alternatywa platformowa wymaga konfiguracji oraz testów, szczególnie równoczesnych duplikatów i nieznanego wyniku odpowiedzi. Nazwy modułów i ich opcje mogą zmieniać się między wersjami, dlatego plan mapowania nie udaje przetestowanego eksportu.

Przy zmianie kontraktu najpierw ustal kompatybilność. Dodanie obowiązkowego pola może odrzucić istniejące zadania; zmiana sposobu generowania HTML może zmienić hash treści przy tym samym UUID. Nie edytuj przygotowanego payloadu w trakcie retry. Utrwalona treść jest częścią tożsamości próby, a zmiana redakcyjna powinna zostać wykonana przez człowieka w szkicu lub zamówiona jako nowe, jawne zadanie.

W dzienniku zapisz wersję kontraktu, identyfikator wydania wtyczki i wykonawcy oraz datę odbioru. Taki zestaw pozwala odtworzyć decyzję bez przechowywania sekretu. Nie musisz logować pełnego tekstu na każdym etapie, aby zachować użyteczną historię.

PROMPT DO CODEXA / Codex: przeczytaj kontrakt

Cel: sprawdź rozdzielenie rezultatu i narzędzia.

Dane: workflow-contract.json oraz README. Wynik: tabela pól, stanów, właścicieli i warunków przejścia. Zaznacz, które reguły egzekwuje runner, a które wtyczka; nie dopisuj automatycznej publikacji.

KONTROLA: Kontrola: wskaż dokładne pliki i warunki kodu. Niewykonane testy oznacz jako plan, a nie dowód.

PRO / AUTOMATYZACJE

Sekrety i dane mają własną granicę

Formularz działa w przeglądarce, lecz dostęp do WordPressa należy do serwera. Ta granica porządkuje również logi, kopie i odpowiedzialność operatora.

Przeglądarka wysyła opis do gateway. Nie dostaje loginu bota, Application Password ani klucza modelu. Serwer wykonawcy odczytuje dane uwierzytelnienia z konfiguracji środowiska i łączy się z ustalonym adresem WordPressa. Adres bazowy nie pochodzi z pola formularza. Taka decyzja zmniejsza możliwość kierowania żądań do dowolnego celu i utrzymuje sekret poza kodem dostarczonym użytkownikowi.

W lokalnym laboratorium gateway nasłuchuje tylko na loopback i przyjmuje demonstracyjne wejście z jednej strony. Nie stanowi kompletnego publicznego formularza. Jeżeli wystawisz podobną usługę, potrzebujesz modelu uwierzytelnienia, autoryzacji, limitów i ochrony zależnej od sesji. Sam nagłówek Origin nie identyfikuje użytkownika. Nie rozciągaj lokalnego uproszczenia na stronę klienta. Najpierw udokumentuj, kto może tworzyć zlecenia i ile pracy może wygenerować.

Kolejka zawiera treść zadania, więc jest zasobem wymagającym kontroli dostępu. Umieść bazę poza webroot i uploads. Ogranicz użytkownika systemowego, kopie oraz dostęp do historii wykonania. Do logów operacyjnych zwykle wystarczą UUID, stan, kod odpowiedzi, numer próby i czas. Sekret nigdy nie powinien być wartością diagnostyczną, a pełny tekst nie musi pojawiać się w każdym logu.

Ustal sposób unieważnienia hasła aplikacji i przejęcia zadania przez operatora. Dobrze zaprojektowana integracja daje się zatrzymać bez utraty śladów. Wyłączenie workerów nie powinno usuwać kolejki ani zmuszać redaktora do ponownego wysłania formularza.

- Frontend: tylko treść demonstracyjna, UUID i komunikat o przyjęciu.
- Serwer: sekret bota, adapter AI, kolejka poza webroot i ustalony endpoint.
- Operator: kontrola stanu, rotacja sekretu oraz uzgodnienie uncertain bez kasowania historii.

Dwa modele uwierzytelnienia REST

Sesja edytora w przeglądarce i zewnętrzny worker mają różne mechanizmy uwierzytelnienia. W obu przypadkach prawo wykonania operacji sprawdza serwer.

Cookie authentication działa w kontekście zalogowanej sesji WordPressa. Przy wywołaniu REST z tej sesji klient przekazuje nonce REST, zwykle przez X-WP-Nonce. Nonce pomaga powiązać żądanie z sesją i chronić operację, ale nie zastępuje sprawdzenia capabilities. Użytkownik bez prawa do intake nie zyskuje go przez posiadanie poprawnego nonce. Gdy projektujesz panelowe narzędzie, musisz rozumieć obie warstwy, zamiast traktować nagłówki jako uniwersalny klucz.

Zewnętrzny worker nie powinien kopiować cookies administratora ani polegać na krótkotrwałej sesji edytora. Nasza integracja używa osobnego konta i Application Password przez HTTPS. To hasło przeznaczone dla aplikacji, które można unieważnić niezależnie od zwykłego hasła użytkownika. Prawa nadal wynikają z konta, dlatego bot otrzymuje wyłącznie read i wdl_ingest. Mechanizm auth identyfikuje konto; permission_callback rozstrzyga, czy operacja jest dozwolona.

Nie umieszczaj danych Basic Auth w URL. Runner buduje nagłówek na serwerze i nie podąża za przekierowaniem. Przekierowanie do strony logowania lub innego hosta jest niezgodnością konfiguracji, którą trzeba rozpoznać, zamiast kontynuować z sekretem. Ustaw bazowy adres końcowego stagingu z HTTPS i przetestuj sposób obsługi Authorization przez reverse proxy.

W protokole oddziel test braku uwierzytelnienia od testu niedostatecznych uprawnień. Oba powinny zakończyć się odmową, ale diagnoza prowadzi do innych ustawień. Nie naprawiaj odmowy nadaniem konta administratora.

PROMPT DO CODEXA / Codex: macierz auth

Cel: przygotuj plan testów uwierzytelnienia.

Dane: endpoint intake, konto bota i konto bez wdl_ingest, opis cookie/nonce oraz Application Password. Wynik: macierz wywołań i oczekiwanych odmów lub draft, bez prawdziwych sekretów.

KONTROLA: Kontrola: nonce nie nadaje capability; HTTPS obowiązuje worker. Wyniki stagingu pozostają puste do rzeczywistego wykonania.

PRO / AUTOMATYZACJE

Capability opisuje dozwoloną operację

Własne `wdl_ingest` daje botowi prawo przyjęcia szkicu przez wskazaną trasę. Nie jest skrótem do administracji ani ogólnym prawem edycji wpisów.

Wtyczka tworzy rolę `wdl_bot` z capability `wdl_ingest` oraz `read`. Callback uprawnień sprawdza `current_user_can`, a właściwy handler dodatkowo narzuca sposób zapisu. Te dwie warstwy rozwiązują różne problemy. Pierwsza decyduje, kto może wejść; druga, co można zrobić po wejściu. Bot nie potrzebuje `edit_posts`, ponieważ specjalizowany endpoint zapisuje jeden z góry określony typ rezultatu. Nie nadajemy mu `publish_posts` ani `manage_options`.

Aktywacja wtyczki nie jest audytem istniejącej roli. Jeżeli `wdl_bot` wcześniej istniała, `add_role` nie nadpisuje wszystkich jej ustawień. Administrator musi sprawdzić efektywne uprawnienia konkretnego konta, także pochodzące z innych wtyczek. W środowisku z rozszerzonym modelem ról nie wystarcza zobaczyć nazwę roli w panelu. Sprawdź rzeczywiste zachowanie tras i próbę publikacji, zapisując wynik na stagingu.

Próba obejścia przez standardowe API wpisów powinna zakończyć się odmową, bo konto nie ma `edit_posts`. Specjalna trasa `intake` nie przyjmuje statusu publikacji, identyfikatora dowolnego użytkownika ani adresu do pobrania. Dzięki temu pozwolenie jest wąskie i da się opisać bez rozległego zestawu wyjątków. Ewentualne nowe funkcje otrzymują oddzielne wymagania i testy.

Po dezaktywacji lub zakończeniu laboratorium unieważnij hasło aplikacji oraz zdecyduj o koncie i zachowaniu tabeli. Nie usuwamy historii automatycznie podczas deaktywacji, ponieważ ślady UUID pomagają uzgodnić wynik. Retencję oraz usuwanie danych ustala operator.

Uprawnienie do przyjęcia szkicu

01 Tożsamość

Bot przez Application Password

02 Capability

`current_user_can: wdl_ingest`

03 Schemat

Wyłącznie cztery pola

04 Zapis

Serwer wymusza draft

PRO / AUTOMATYZACJE

Trasę REST rejestruj jawnie

Namespace określa kontrakt endpointu, a `permission_callback` jego bramkę. Fragment pokazuje rejestrację; pełna walidacja i zapis znajdują się w pliku wtyczki.

Trasa POST `/wp-json/wp-demo-lab/v1/intake` należy do wtyczki, nie do motywu. Motyw odpowiada za wygląd, a `intake` za domenową operację tworzenia szkicu. Namespace z wersją pozwala opisać przyszłe zmiany zachowania. Nie zwiększaj numeru tylko dlatego, że zmieniła się strona formularza; wersja endpointu powinna odzwierciedlać kompatybilność kontraktu i obsługę istniejących wykonawców.

Rejestracja w `rest_api_init` wskazuje metodę, `callback` i `permission_callback`. Nie używamy publicznego zwrotu `true` dla operacji zapisu. Callback uprawnień korzysta z `capabilities` bieżącego konta. Handler rozpoznaje dokładny zestaw pól i zwraca błędy o określonym znaczeniu. Uczciwe statusy pomagają operatorowi: odrzucenie wejścia różni się od potwierdzonego szkicu i od nieznanego wyniku.

Przeczytaj pełny plik przed instalacją. Fragment nie zawiera aktywacji tabeli ani obsługi duplikatów. Na stagingu sprawdź, czy reverse proxy przekazuje `Authorization` i omija cache dla tej ścieżki. To część integracji środowiskowej, której sam poprawny PHP nie dowodzi. W materiałach nie deklarujemy uruchomienia wtyczki.

KOD W PRAKTYCE / PHP / Rejestracja prywatnej trasy

```
function wdl_intake_permission() {
    return current_user_can('wdl_ingest');
}
add_action('rest_api_init', function () {
    register_rest_route('wp-demo-lab/v1', '/intake', array(
        'methods' => 'POST',
        'callback' => 'wdl_intake_receive',
        'permission_callback' => 'wdl_intake_permission',
    ));
});
```

PLIK: `wp-demo-intake/wp-demo-intake.php`

Fragment pełnego pliku; PHP nieuruchomiony

Formularz zamawia zadanie

Kliknięcie wysyłania oznacza przyjęcie danych do lokalnej kolejki. Nie oznacza gotowego szkicu, sukcesu AI ani zgody na publikację w WordPressie.

Formularz HTML pokazuje tytuł i demonstracyjny opis. UUID powstaje przy przygotowaniu zlecenia i pozostaje ten sam po błędzie komunikacji. Dzięki temu użytkownik nie tworzy przypadkiem nowej tożsamości, gdy nie zobaczy odpowiedzi. Dopiero potwierdzone przyjęcie przygotowuje następny identyfikator dla nowego zlecenia. Jeśli nie wiadomo, czy gateway zapisał zadanie, komunikat kieruje do sprawdzenia kolejki.

Gateway odpowiada 202 i state queued. To rozdzielenie ma praktyczne znaczenie: kosztowne przygotowanie tekstu i wywołanie WordPressa odbywają się później. Formularz nie czeka na cały proces, a krótkie przyjęcie jest łatwiejsze do monitorowania. Nie pokazuj użytkownikowi słowa „opublikowano”, gdy dysponujesz tylko UUID kolejki. Komunikat powinien mówić, jaki etap naprawdę zakończono i gdzie można sprawdzić dalszy stan.

Walidacja przeglądarkowa poprawia obsługę, lecz nie ustanawia kontraktu serwera. Atrybut maxlength nie mierzy limitu bajtów UTF-8 identycznie jak wykonawca. Użytkownik może też wysłać żądanie bez formularza. Dlatego gateway sprawdza dane ponownie, a endpoint WordPress jeszcze raz. Każda granica przyjęcia ma własny obowiązek kontroli, nawet jeśli wcześniej klient wygląda wiarygodnie.

Lokalny serwer nie zapisuje danych osobowych i nie ma mechanizmu kont użytkowników. Jest samodzielnym demonstratorem na loopback. Do użycia na publicznej stronie potrzebna byłaby zaprojektowana bramka serwerowa z autoryzacją oraz ochroną przed nadużyciami. Nie wystawiaj tego pliku przez tunel jako gotowego produktu.

PROMPT DO CODEXA / Codex: komunikaty formularza

Cel: nazwij etapy bez obietnicy publikacji.

Dane: form.html, form_server.py i trzy sytuacje: queued, odmowa walidacji, utrata odpowiedzi.
Wynik: proponowane polskie komunikaty oraz warunek użycia każdego z nich.

KONTROLA: Kontrola: komunikat queued nie mówi o szkicu; timeout zachowuje UUID i kieruje do sprawdzenia stanu. Nie dodawaj sekretów do JavaScript.

Schemat zamknięty ogranicza niespodzianki

Przyjmujemy dokładnie cztery pola, tekstowe wartości i kanoniczny UUID v4. Dodatkowe pole publikacji nie zostaje zignorowane: całe wejście jest odrzucane.

Walidacja powinna odpowiadać na proste pytania, które da się przetestować. Czy wejście jest obiektem? Czy ma dokładnie wymagane klucze? Czy wartości są tekstem? Czy UUID ma oczekiwaną postać i wersję? Czy source jest równe demo-form? Czy tekst po usunięciu białych znaków jest niepusty i mieści się w limitach bajtów? Takie reguły wyprzedzają jakikolwiek adapter AI i zdalne wywołanie.

Zamknięty schemat chroni przed przypadkowym rozszerzeniem zachowania. Gdy ktoś przesyła `post_status`, autor nie chce przyjąć połowy intencji i przemilczeć reszty. Odmowa pokazuje błąd klienta. Ten sam mechanizm wyklucza przekazanie dowolnego URL, który serwer miałby pobrać. Nie dodajemy ukrytej obsługi nieznanych pól ani polecenia „zrób wszystko z JSON”. Wniosek z danych nie może stać się przywilejem systemowym.

Limity dotyczą bajtów UTF-8, więc znaki wielobajtowe zajmują więcej miejsca niż prosty alfabet. To celowa wspólna reguła Pythona i PHP. MOCK escapuje treść, co może ją powiększyć; wynik przechodzi walidację ponownie. Dzięki temu formularz mieszczący się w limicie wejściowym nie gwarantuje akceptacji dowolnie rozrośniętego HTML. Przed AI nie obiecuj użytkownikowi nieograniczonego rozwinięcia.

Sprawdzanie schematu nie potwierdza prawdziwości zdania. Tekst może być poprawny technicznie i wymagać korekty redaktora. Zachowuj tę różnicę w nazwach testów oraz komunikatach, aby odbiór formatu nie wyglądał jak kontrola jakości merytorycznej.

- Odrzuć: pustą treść, zły UUID, source inne niż demo-form i dodatkowe klucze.
- Sprawdź granice w bajtach UTF-8; osobno testuj polskie znaki i rozszerzenie HTML.
- Zachowaj schemat na trzech granicach: gateway, wykonawca i intake WordPressa.

PRO / AUTOMATYZACJE

Walidacja ma być czytelna w kodzie

Fragment wykonawcy pokazuje identyfikator, stałe pochodzenie i limity. Pełna funkcja obejmuje także dokładny zestaw kluczy oraz typ wszystkich wartości.

Python udostępnia UUID i kodowanie UTF-8 w bibliotece standardowej. Nie potrzebujemy pakietu walidacyjnego, aby wyrazić niewielki kontrakt. Funkcja `validate` zwraca kopię danych lub rzuca `ValueError`. Wyjątek zatrzymuje `enqueue`, zanim powstanie zlecenie dla WordPressa. Ten prosty model jest czytelny w testach i nie ukrywa niepoprawnego rekordu pod częściowo udanym wynikiem.

Kanoniczny UUID pozostaje niezmieniony przez adapter. Gdy klient używa tej samej tożsamości z inną treścią, `enqueue` rozpoznaje konflikt. Taką sytuację trzeba naprawić u źródła, zamiast automatycznie dopisać sufiks albo nadpisać stare zadanie. Korekta opisu powinna być nową, świadomą operacją z oddzielnym UUID albo redakcją już istniejącego szkicu.

Fragment pokazuje pełną funkcję `validate` z `runner.py`. Rozszerzając kontrakt, zmień zgodnie gateway, wykonawcę, wtyczkę, workflow i próby. Pole dodane tylko w formularzu zostanie odrzucone przez serwer.

KOD W PRAKTYCE / PYTHON / UUID i limity wejścia

```
def validate(data):
    if not isinstance(data, dict) or set(data) != FIELDS:
        raise ValueError('Exact schema required')
    if not all(isinstance(v, str) for v in data.values()):
        raise ValueError('Strings required')
    ident = uuid.UUID(data['request_id'])
    if ident.version != 4 or str(ident) != data['request_id']:
        raise ValueError('Canonical UUID v4 required')
    if data['source'] != 'demo-form':
        raise ValueError('Fixed source required')
    for key, limit in [('title', 240), ('content', 12000)]:
        if not data[key].strip() or len(data[key].encode()) > limit:
            raise ValueError('Invalid length: ' + key)
    return dict(data)
```

PLIK: `services/runner.py`

Fragment pełnego pliku; Python sprawdzony offline

Tożsamość zadania przeżywa restart

UUID jest kluczem historii, a nie ozdobą logu. Ta sama tożsamość łączy formularz, rekord kolejki, żądanie intake i metadane szkicu.

Kolejka zapisuje request_id jako PRIMARY KEY. Ponowne przyjęcie identycznego payloadu nie tworzy drugiego rekordu. Jeżeli zawartość różni się przy tym samym identyfikatorze, wykonawca zgłasza konflikt. Porównanie i zapis odbywają się w transakcji BEGIN IMMEDIATE, aby dwa równoległe wywołania nie obchodziły kontroli. Unikalność bazy jest mocniejszą granicą niż zmienna w pamięci procesu.

W rekordzie przechowujemy payload wejściowy oraz prepared, czyli przygotowaną treść wysyłaną do WordPressa. Prepared zostaje zapisany przed wywołaniem sieciowym. Jeśli 429 pozwala na późniejszą próbę, nie regenerujemy tekstu na nowo. Przy prawdziwym modelu druga generacja mogłaby zmienić HTML, a identyczny UUID z innym payloadem wywołałby konflikt po stronie WordPressa. Utrwalenie wyniku jest częścią kontroli tożsamości.

Restart procesu nie może usuwać historii prób. Próby, stan, czas następnego wykonania oraz przyczyna zatrzymania znajdują się w SQLite. Plik powinien pozostać na tym samym trwałym woluminie, objęty ustaloną polityką kopii i dostępny dla właściwego użytkownika systemowego. Skopiowanie zadania do nowej bazy, aby zniknął błąd, może usunąć lokalną wiedzę o skutku zdalnym.

SQLite pasuje do małego demonstratora na jednym hoście. Nie jest tutaj deklaracją architektury rozproszonej ani gwarancją dużego ruchu. Gdy potrzebujesz wielu maszyn, zaprojektuj magazyn i claim obsługujące tę topologię, zachowując znaczenie stanów oraz regułę nieznanego wyniku.

Trwała tożsamość

01 Formularz

UUID powstaje raz

02 SQLite

Unikalny klucz i payload

03 Prepared

Utrwalony wynik adaptera

04 WordPress

Tabela kluczy i meta szkicu

PRO / AUTOMATYZACJE

Rezerwacja pracy jest atomowa

Worker wybiera jeden gotowy rekord i oznacza go inprogress w tej samej transakcji. Drugi wykonawca nie powinien zabrać tego samego zadania.

Odczyt zadania i późniejsze ustawienie stanu bez transakcji tworzy okno wyścigu. Dwa procesy mogłyby zobaczyć queued, a następnie oba wysłać żądanie. Runner zaczyna BEGIN IMMEDIATE, wybiera rekord queued lub retry z osiągniętym next_at, zwiększa attempts i zapisuje inprogress. Dopiero po commit wykonuje zewnętrzną pracę. Test z dwoma połączeniami potwierdza, że drugi claim nie dostaje zajętego rekordu.

Nie trzymaj transakcji SQLite podczas oczekiwania na API. Baza ma chronić krótką rezerwację, a nie blokować pozostałe wejścia na czas sieci. Poza transakcją mogą wydarzyć się awarie, dlatego inprogress jest stanem trwałym. Gdy proces zniknie, operator zobaczy przerwana pracę zamiast pustego śladu. To świadomy kompromis między krótkim blokowaniem a możliwością uzgodnienia.

Attempts zwiększamy przy rezerwacji, zanim wykonawca ruszy. W prostym kontrakcie liczymy trzy próby ogółem, nie trzy dodatkowe ponowienia. Błąd adaptera może zatrzymać rekord bez wywołania WordPressa; licznik nadal dokumentuje podjętą pracę. Dokładniejsze mierniki można dodać osobno, ale nie zmieniaj znaczenia limitu w połowie projektu.

Ta implementacja nie używa automatycznego lease, który po czasie oddaje zadanie do kolejki. Oddanie mogłoby powtórzyć tworzenie, jeśli zdalny szkic powstał przed awarią procesu. Stary inprogress przechodzi do uncertain i wymaga człowieka. Uczciwy stan jest ważniejszy niż pozorne samonaprawienie każdej sytuacji.

PROMPT DO CODEXA / Codex: sprawdź wyścig

Cel: oceń atomowość przyjęcia i rezerwacji.

Dane: enqueue, claim i test_atomic_claim w runner.py oraz test_runner.py. Wynik: opis granic transakcji i scenariusz dwóch procesów, bez rozszerzania obietnicy na sieć.

KONTROLA: Kontrola: oddziel unikalny lokalny rekord od zdalnego skutku; sprawdź rollback po wyjątku i brak blokady w czasie HTTP.

Stan kolejki prowadzi operatora

Dobrze nazwane stany są instrukcją działania. Sukces, odrzucenie i brak wiedzy wymagają różnych następnych kroków, choć każdy może zakończyć bieżący worker.

Queued oznacza poprawnie przyjęte zadanie, jeszcze niewykonywane. Inprogress to atomowo zarezerwowana praca. Retry oznacza znane odrzucenie przez limiter i oczekiwanie do next_at. Drafted wymaga poprawnego potwierdzenia WordPressa z tym samym UUID, dodatnim post_id i stanem utworzenia draft. Failed jest zakończeniem z konkretną przyczyną, na przykład odmową uprawnień albo przekroczeniem limitu prób.

Uncertain nie znaczy, że wpis nie istnieje. Znaczy, że wykonawca nie ma dostatecznego dowodu wyniku. Taki stan pojawia się po timeout, nieoczekiwanym statusie, uszkodzonym potwierdzeniu lub porzuconym claim. Właściwy następny krok to odczyt i uzgodnienie, a nie automatyczne wygenerowanie nowego UUID. Nazwa powinna jasno wywoływać tę procedurę w panelu operatora i dokumentacji.

Stan WordPressa ma podobny cel, ale inny magazyn. Tabela intake zapisuje inprogress, drafted lub uncertain. Nie musi kopiować queued i retry, bo one dotyczą czasu przed przyjęciem przez endpoint. Porównując systemy, zestawiaj UUID oraz znaczenie etapu, zamiast wymagać identycznego stanu we wszystkich miejscach. Serwer WordPress może wiedzieć więcej niż worker po utracie odpowiedzi.

W interfejsie diagnostycznym nie wyświetlaj technicznej niepewności jako „spróbuj jeszcze raz”. Operator potrzebuje informacji o ostatniej próbie, HTTP i identyfikatorze. Redaktor potrzebuje informacji, czy odnaleziono szkic. Role widzą odpowiedni zakres danych, a sekret nigdy nie staje się częścią opisu błędu.

Stan	Następny krok
queued	Worker może wykonać claim
retry	Poczekaj do next_at
drafted	Redaktor sprawdza szkic
failed	Napraw przyczynę, zdecyduj o nowej pracy
uncertain	Zatrzymaj retry i uzgodnij wynik

MOCK domyślny, OpenAI opcjonalnie

Przepływ działa offline z deterministycznym MOCK. Opcjonalny kompletny adapter Responses API można włączyć konfiguracją serwera; nie wykonano rzeczywistych wywołań ani oceny jakości modelu.

MockAI escapuje wejście i otacza je akapitem HTML. Nie rozumie opisu, nie wyszukuje źródeł ani nie poprawia faktów. Domyślny tryb pozwala sprawdzić kolejkę, transport i zapis szkicu bez klucza oraz kosztu modelu. Dry-run zawsze pozostaje przy MOCK, również po ustawieniu innego trybu środowiska. Po przygotowaniu treść przechodzi walidację, ponieważ escaping może zwiększyć liczbę bajtów.

Kompletny OpenAIAdapter wybierasz przez `WDL_AI_MODE=openai`. Serwer odczytuje `OPENAI_API_KEY` i `WDL_AI_MODEL`; wybierz dostępny model obsługujący Responses API oraz Structured Outputs według aktualnej dokumentacji. Adapter wysyła title/content na stały endpoint HTTPS, bez narzędzi i bez danych dostępu do WordPressa. Model zwraca tylko title/content, a request_id i source zachowuje kod. JSON Schema jest ścisły, lecz lokalna walidacja nadal sprawdza typy, pola i limity UTF-8.

Transport nie podąża za redirectami, ma timeout socket 20 sekund, limit odpowiedzi 64 KiB i limit wyjścia 1200 tokenów. Nie ponawia automatycznie wywołań providera. Refusal, incomplete, błędna odpowiedź albo przekroczony limit zatrzymują zadanie jako failed przed WordPresssem. Ogólny komunikat nie ujawnia sekretu ani surowego body. Ustawienie `store=false` nie zastępuje sprawdzenia warunków retencji własnego konta.

Lokalne testy HTTP sprawdziły request Structured Outputs, zachowanie tożsamości, odmowy, brak ujawnienia sekretu i cały mockowany transport AI→intake draft. Nie kontaktowały się z OpenAI. Rzeczywisty model wymaga oddzielnego testu jakości na danych syntetycznych oraz odbioru redaktora. Zapisane prepared nie regeneruje się podczas retry; zmiana trybu dla nowej oceny oznacza świadomie nowe zadanie.

PROMPT DO CODEXA / Codex: odbiór opcjonalnego adaptera

Cel: przygotuj kontrolę przełączenia trybu.

Dane: OpenAIAdapter, README i test_openai_adapter.py. Wynik: plan konfiguracji serwera, lista limitów oraz kryteria lokalnego i rzeczywistego odbioru bez wykonywania API.

KONTROLA: Kontrola: domyślny MOCK, UUID/source poza modelem, brak tools i sekretów frontendowych. Rzeczywiste wywołanie oraz jakość pozostają niewykonane.

PRO / AUTOMATYZACJE

Wynik AI jest niezaufaną propozycją

Model może przygotować materiał redakcyjny, lecz nie powinien wybierać endpointu, konta, statusu wpisu ani uprawnień. Te decyzje pozostają w deterministycznym kodzie.

W prawdziwej integracji instrukcja dla modelu opisuje zadanie tekstowe i oczekiwany format. Nie przenosi kontroli nad systemem do treści użytkownika. Jeżeli opis zawiera zdanie „opublikuj natychmiast”, traktujesz je jako dane do oceny, a nie polecenie dla transportu. Wtyczka nie przyjmuje statusu i zawsze wymusza draft. Wykonawca nie bierze URL z odpowiedzi modelu. Takie granice działają również wtedy, gdy model źle interpretuje intencję.

Oddziel treść od metadanych sterujących. Request_id i source pochodzą z zlecenia. Adapter może zmienić zawartość opisu w dozwolonym zakresie, ale nie może zmienić tożsamości pracy. Walidacja rezultatu sprawdza schemat i limity, a WordPress dodatkowo filtruje HTML przez wp_kses_post. Nie oznacza to dowodu prawdziwości ani kompletnej ochrony każdej konfiguracji zewnętrznych rozszerzeń. Redaktor nadal przegląda gotową treść.

Zwróć uwagę na materiał wejściowy. Formularz w tym tomie jest syntetyczny i nie przyjmuje prywatnych dokumentów. Rozszerzenie o dane klienta wymaga nowej decyzji o uprawnieniach, retencji i przekazaniu do providera. Nie dopisuj tego jako drobnego pola upload bez zmiany kontraktu. Szczególnie nie pobieraj dowolnego adresu podanego w treści, bo powstaje oddzielny problem granic sieciowych.

Przy próbach jakościowych zachowaj wejście, rezultat i uwagi redaktora w kontrolowanym miejscu. Sprawdź, czy model nie wymyślił cen, terminów lub obietnic usług. Dobre brzmienie nie jest odpowiednikiem zgodności ze źródłem.

Kontrola propozycji tekstowej

01 Dane

Opis bez poleceń administracyjnych

02 Adapter

Propozycja HTML na serwerze

03 Walidacja

Schemat, UUID i limity

04 Intake

Filtr HTML oraz draft

05 Człowiek

Fakty, ton i decyzja

HTML potrzebuje dwóch kontroli

Escaping MOCK chroni jego proste przekształcenie. Po stronie WordPressa `wp_kses_post` ogranicza dopuszczalny HTML każdego wykonawcy, także przyszłego providera.

Treść HTML służy prezentacji szkicu i nie powinna zawierać aktywnych skryptów. W demonstratorze MOCK escapuje całe wejście, więc wpisany znacznik staje się widocznym tekstem akapitu. To świadomie wąski format. Nie generujemy obrazów, embedów ani rozbudowanych bloków. Jeśli przyszły adapter przygotuje nagłówki i listy, potrzebujesz dopuszczalnego zakresu oraz prób w edytorze, zamiast zakładać zgodność każdego HTML.

WordPress ponownie sprawdza wynik przez `wp_kses_post` i odrzuca pustą treść po filtracji. Tytuł przechodzi `sanitize_text_field`. Hash idempotencji liczymy z przygotowanych wartości po tych przekształceniach, aby porównywać dane rzeczywiście przekazywane do zapisu. Zasady filtrowania i hooks WordPressa zależą od środowiska, więc konieczny jest stagingowy odbiór z konkretnymi wersjami wtyczek.

Nie oceniaj bezpieczeństwa HTML wyłącznie na podstawie wyglądu strony. Redaktor powinien obejrzeć kod szkicu lub podgląd odpowiednim narzędziem, sprawdzić odnośniki i usunąć nieuzasadnione elementy. Integracja nie tworzy automatycznych linków do adresów wejściowych. Gdy potrzebujesz cytowania źródeł, zdefiniuj dozwolone pochodzenie osobno i zweryfikuj docelowe adresy, nie kopiuj bez sprawdzenia tekstu modelu.

Test `mock_escapes_html` daje konkretny dowód dla funkcji Python. Nie sprawdza `wp_kses_post` ani innych filtrów środowiska WordPress. W dokumentacji jakości napisz oba zdania obok siebie: escaping MOCK przeszedł próbę, filtr WP wymaga próby stagingowej. Taka precyzja ułatwia odbiór zamiast obniżać wartość testów.

- Zaczynij od akapitu; rozszerzaj format tylko z regułami i próbą edytora.
- Sprawdź puste wyjście po filtracji i wzrost liczby bajtów przy escapingu.
- Nie utożsamiaj poprawnego HTML z poprawnym twierdzeniem ani zgodą na publikację.

Trwały klucz w bazie WordPressa

Idempotencja endpointu wymaga pamięci dłuższej niż pojedynczy request. Wtyczka używa osobnej tabeli z unikalnym request_id oraz stanem operacji.

Flaga w pamięci PHP nie przetrwa kolejnego żądania. Transient z okresem życia także nie daje tu oczekiwanej trwałej historii. Osobna tabela przechowuje UUID, hash payloadu, stan, ID wpisu i czas aktualizacji. Aktywacja używa dbDelta, a nazwa tabeli wynika z prefiksu WordPressa. Zapytania z wartościami wejściowymi są przygotowywane przez wpdb->prepare; nazwa tabeli pochodzi z zaufanej konfiguracji, nie z formularza.

Pierwsze żądanie próbuje INSERT IGNORE z inprogress. Unikalny klucz pozwala tylko jednemu requestowi zastrzec tożsamość. Drugi odczytuje istniejący rekord. Identyczny hash i drafted dają potwierdzenie duplikatu z tym samym post_id. Inny hash kończy się conflict. Inprogress lub uncertain nie tworzą kolejnego wpisu i prowadzą do uzgodnienia. Nie oczekuj, że równoczesny duplikat zawsze dostanie gotowe ID; pierwsza praca może jeszcze trwać.

Tabela nie jest transakcją obejmującą cały świat WordPressa. Tworzenie wpisu wywołuje hooks, a niektóre rozszerzenia mogą wykonywać zewnętrzne skutki. Po utworzeniu wpisu może nastąpić awaria przed aktualizacją rekordu intake. Dlatego przechowujemy także UUID w prywatnej meta i nie nazywamy mechanizmu exactly-once. Ochrona polega na odmowie ślepego ponownego tworzenia, nie na wymazaniu okna awarii.

Na stagingu sprawdź dbDelta, indeks i zachowanie równoległych żądań w rzeczywistej bazie. Offline mock nie wykonuje tego SQL. Administrator powinien wiedzieć, gdzie odczytać rekord, lecz nie wystawiamy surowego dostępu do tabeli w publicznym API.

PROMPT DO CODEXA / Codex: przejrzyj idempotencję PHP

Cel: znajdź granice trwałej ochrony.

Dane: pełny wp-demo-intake.php, szczególnie INSERT IGNORE, payload_hash i wp_insert_post.
Wynik: lista okien awarii oraz plan prób równoczesnych żądań na stagingu.

KONTROLA: Kontrola: nie deklaruj exactly-once. Sprawdź prepare dla wejścia i rozróżnij inprogress, drafted, conflict oraz uncertain.

PRO / AUTOMATYZACJE

Serwer wymusza draft przy zapisie

Wtyczka tworzy zwykły wpis z `post_status` draft i UUID w meta. Nie przepisuje statusu z danych, ponieważ taki parametr nie należy do wejścia.

Najważniejsza reguła zapisu jest widoczna w tablicy `wp_insert_post`. `Post_type` jest stały, `post_status` jest draft, a autor wynika z uwierzytelnionego konta. Endpoint nie przyjmuje ID wpisu do aktualizacji ani listy dowolnych metadanych. Zakres pozostaje mały: utworzyć kontrolowany szkic po poprawnym zastrzeżeniu UUID. Wąski handler łatwiej odebrać niż ogólną bramkę do wszystkich operacji CMS.

Po uzyskaniu `post_id` wtyczka aktualizuje tabelę intake do drafted. Dopiero utrwalone potwierdzenie pozwala odpowiedzieć sukcesem. Wyjątek lub nieudany zapis potwierdzenia prowadzi do uncertain. Jeśli proces PHP zostanie zakończony nagle, catch może w ogóle się nie wykonać i tabela pozostanie inprogress. To również wymaga uzgodnienia, a kolejny identyczny request nie tworzy nowego wpisu.

Fragment pokazuje tworzenie, nie pełny handler. JSON dostarcza unslashed tekst: `wp_slash` przygotowuje tablicę dla `wp_insert_post`, które usuwa slashy przed zapisem. Hash pozostaje z canonical title/content przed slashingiem. Na stagingu sprawdź zachowanie regexu `\d{3}`, ścieżki oraz apostrofu; tej próby WordPress nie wykonano.

KOD W PRAKTYCE / PHP / Stały status i ślad UUID

```
$post_id = wp_insert_post(wp_slash(array(
    'post_type' => 'post', 'post_status' => 'draft',
    'post_author' => get_current_user_id(),
    'post_title' => $title, 'post_content' => $content,
    'meta_input' => array('_wdl_request_id' => $id),
)), true);
```

PLIK: wp-demo-intake/wp-demo-intake.php

Fragment pełnego pliku; PHP nieuruchomiony

Potwierdzenie musi być sprawdzone

HTTP 201 to ważny sygnał, ale worker potrzebuje również poprawnego JSON, tego samego UUID, dodatniego `post_id` i deklaracji utworzenia szkicu.

Warstwa sieciowa może zwrócić odpowiedź HTML proxy albo uszkodzony JSON z kodem wyglądającym na sukces. Runner nie kończy zadania na samym 2xx. Odczytuje ograniczoną odpowiedź, parsuje JSON i sprawdza pola kontraktu. `Post_id` ma być liczbą całkowitą dodatnią; wartość logiczna nie przechodzi kontroli mimo związku `bool` i `int` w Pythonie. UUID musi odpowiadać zarezerwowanemu rekordowi.

Niepoprawne potwierdzenie prowadzi do `uncertain`. Szkic mógł już powstać, więc automatyczny retry byłby niewłaściwym sposobem naprawienia parsera. Operator odczytuje ślady po stronie WordPressa i dopiero uzgadnia wynik. Test `malformed_success` pokazuje dokładnie ten przypadek: 201 z uszkodzoną zawartością kończy pracę bez drugiego wywołania. To użyteczniejszy test niż udowadnianie wyłącznie udanej ścieżki.

Dla API zapisujemy jawne przykłady odpowiedzi, nigdy rzekome odczyty ze stagingu. Pierwsze przyjęcie może dać 201, a identyczny zakończony duplikat 200. Konflikt to 409 z odpowiednim stanem. 400 odrzuca dane. Uwierzytelnienie lub `capability` kończą się odmową. Wtyczka nie generuje automatycznego publikowania ani kolejnej próby po niejasnym statusie.

Oba magazyny mają własne dowody. Jeśli WordPress potwierdził `drafted`, a lokalna aktualizacja SQLite nie doszła do skutku, `recovery` ustawi `uncertain` i operator porówna UUID. Nie usuwaj tego rekordu, aby zniknęła czerwona etykieta. Odbiór polega na poprawnym ustaleniu skutku.

Odpowiedź przykładowa	Znaczenie
201 / <code>drafted</code> / <code>post_id</code> : 17	Nowy szkic został potwierdzony
200 / <code>drafted</code> / <code>duplicate</code> : true	Wcześniejszy rezultat dla tego UUID
409 / <code>conflict</code>	Ten UUID ma inną treść
409 / <code>inprogress</code> lub <code>uncertain</code>	Bez ponownego tworzenia; uzgodnij
201 / niepoprawny JSON	Worker: <code>uncertain</code> , nie <code>retry</code>

Retry dotyczy znanego odrzucenia

Powtórzenie jest decyzją na podstawie wyniku, a nie uniwersalną reakcją na czerwony status. Referencja automatycznie ponawia wyłącznie 429 przed przyjęciem zadania.

Limiter może odmówić obsługi żądania kodem 429, zanim endpoint przyjmie zlecenie. Przy tym założeniu worker odkłada rekord do retry. Używa wykładniczej minimalnej zwłoki i respektuje liczbowy Retry-After, ograniczony do trzystu sekund. Pierwsza ponowna próba czeka co najmniej dwie sekundy, kolejna co najmniej cztery. Wariant nagłówek z datą HTTP nie jest parsowany w tej referencji; wtedy obowiązuje backoff.

Limit wynosi trzy próby ogółem. Po trzecim 429 rekord przechodzi do failed. Nie licz tego jako pierwszego wywołania plus trzy retry, bo to zmieni obciążenie i wynik próby odbiorowej. Kod zapisuje `next_at`, więc scheduler nie musi usypiać całego procesu na czas oczekiwania. Worker kończy bieżące zadanie, a późniejsze uruchomienie sprawdza gotowość rekordu.

Bezpieczne założenie o 429 musi pasować do konfiguracji proxy i endpointu. Jeśli infrastruktura potrafi oddać 429 po wykonaniu zapisu, taki kod również staje się niejednoznaczny. Przed stagingowym odbiorem ustal właściciela limiterów i kolejność wykonania. Nie przepisuj zasad z dokumentacji platformy bez sprawdzenia, gdzie dokładnie zapada odmowa.

Pozostałych błędów nie wrzucamy do jednej gałęzi retry. Odmowa auth wymaga naprawy konfiguracji, nie kolejnych prób z tym samym sekretem. 5xx lub timeout mogą wystąpić po utworzeniu szkicu. Różnica między „nie przyjęto” a „nie wiem, czy wykonano” jest podstawą całego projektu.

Decyzja po odpowiedzi

01 HTTP

Odczytaj kod i kontrakt

02 429 przed przyjęciem

Retry po delay, max 3 próby

03 Poprawne 200/201

Drafted z tym samym UUID

04 Odmowa danych lub auth

Failed, napraw przyczynę

05 Nieznany wynik

Uncertain, uzgodnij

PRO / AUTOMATYZACJE

Timeout nie dowodzi braku szkicu

Żądanie mogło dojść do WordPressa, a odpowiedź zginąć po drodze. Nie odróżnisz tego od niewykonanej operacji samym faktem przekroczenia czasu.

Wyobraź sobie dwa przebiegi. W pierwszym worker traci połączenie przed dotarciem do serwera. W drugim WordPress zapisuje szkic, aktualizuje tabelę, a sieć przerywa odpowiedź. Z punktu widzenia klienta oba mogą wyglądać jak timeout. Ponowienie z nowym UUID stworzyłoby w drugim przypadku drugie zlecenie. Nawet ten sam UUID wymaga rozpoznania stanu, ponieważ hooks mogły zostawić skutki, a zapis potwierdzenia mógł nie zakończyć się prawidłowo.

Referencja zatrzymuje pracę jako uncertain. Nie udaje, że potrafi rozstrzygnąć każdą awarię transportu. Stan zawiera powód, a operator korzysta ze śladów WordPressa. UUID w tabeli oraz meta wpisu umożliwia znalezienie kandydata. Wtyczka nie odblokowuje automatycznie starego inprogress. Dzięki temu awaria nie zamienia się od razu w nowe tworzenie.

Zatrzymanie procesu worker przed wywołaniem HTTP również może zostawić inprogress. Recovery nie zna dokładnego momentu awarii, więc konserwatywnie wybiera uncertain. To czasem wymaga ręcznego odczytu mimo braku zdalnego wywołania, ale utrzymuje jedną czytelną regułę. Złożony system mógłby dodać precyzyjniejszy dziennik etapów; nadal nie powinien traktować nieznanego skutku jako pewnego braku.

Na stagingu przygotuj kontrolowaną próbę utraty odpowiedzi oraz przerwania po `wp_insert_post`. Zapisz obserwacje obu baz i panelu. Sam test lokalnego mocka nie odtwarza wszystkich hooks WordPressa, ale może potwierdzić, że worker nie uruchamia ślepego retry.

PROMPT DO CODEXA / Codex: scenariusz utraconej odpowiedzi

Cel: przygotuj odbiór uncertain.

Dane: `runner.py`, `recovery` i tabela `intake`. Wynik: dwa scenariusze timeout, lista śladów do odczytu oraz warunki zamknięcia sprawy.

KONTROLA: Kontrola: brak automatycznego resetu do `queued` i brak nowego UUID przed decyzją. Test WordPress zapisz jako niewykonany plan stagingowy.

PRO / AUTOMATYZACJE

Uzgodnienie jest pracą z dowodami

Operator najpierw zatrzymuje ponowienia i odczytuje ślady. Redaktor potwierdza znaną treść; administrator dokumentuje techniczne zamknięcie stanu.

Procedura zaczyna się od UUID, stanu kolejki i ostatniej przyczyny. Zatrzymaj worker dla spornego rekordu. Administrator odczytuje w WordPressie wiersz intake oraz wyszukuje meta `_wdl_request_id`. Redaktor sprawdza, czy odnaleziony wpis odpowiada wejściu i jest właściwym szkicem. Nie wybieraj pierwszego podobnego tytułu: tytuł nie jest trwałym kluczem i może się powtarzać.

Jeśli istnieje jeden poprawny szkic, można udokumentować uzgodnienie i zapisać `drafted` z jego ID w tabeli WordPressa oraz lokalnej kolejce. To administracyjna operacja po odczycie, a nie automatyczny retry w runnerze. README wskazuje funkcję `finish` do lokalnego zapisu. Nie dostarczamy publicznej trasy, która pozwalałaby użytkownikowi oznaczyć dowolny wpis jako rezultat zlecenia. Uprawnienie do naprawy pozostaje oddzielne.

Jeśli nie znaleziono szkicu, zachowaj ostrożność. Brak widocznego wpisu nie dowodzi braku efektów rozszerzeń ani kompletności odczytu. Sprawdź logi środowiska, moment awarii i możliwe hooks. Gdy dowodu nadal brak, `uncertain` pozostaje właściwym stanem. Nowe UUID wymaga jawnej decyzji operatora oraz odnotowania ryzyka duplikatu, a nie skasowania starego wiersza.

Po uzgodnieniu popraw przyczynę: błędny proxy, filtr odpowiedzi, niedostępny zapis tabeli albo przerwanie procesu. Samo zamknięcie jednego rekordu nie leczy mechanizmu. W protokole zachowaj datę, osobę, źródła dowodów i podjętą decyzję bez sekretów. To pozwala później ocenić, czy system bezpiecznie wrócił do pracy.

- Zatrzymaj zadanie; nie resetuj historii i nie wysyłaj nowego UUID.
- Porównaj tabelę intake, meta wpisu, kolejkę i obserwację redaktora.
- Zamknij stan tylko z dowodem; niejednoznaczny brak rezultatu nadal oznacza `uncertain`.

n8n i Make odwzorowują kontrakt

Platforma automatyzacji może być wygodnym edytorem przepływu. Musi jednak zachować te same reguły UUID, trwałości, auth oraz nieznanego wyniku.

Plan N8N_MAKE_MAPPING.md zestawia etapy kontraktu z kandydatami do ręcznej konfiguracji. Webhook może przyjąć wejście, moduł walidacji odrzucić schemat, a HTTP Request wysłać przygotowany payload. To opis decyzji, nie importowalny plik. Narzędzia różnią się obsługą credentials, retry, baz i historii wykonania; opcje zależą od wersji i hostingu. Dlatego zapisujemy konkretne środowisko dopiero podczas rzeczywistego odbioru.

Najważniejszy problem nie znika po narysowaniu węzłów. Potrzebujesz trwałego unikalnego klucza i atomowego claim. Sam filtr „czy już istnieje” wykonany przed osobnym zapisem może mieć wyścig. Zewnętrzna baza lub usługa wykonawcy powinna egzekwować te reguły. Możesz pozostawić referencyjny serwis właścicielem kolejki, a platformę wykorzystać do przyjęcia zlecenia i pokazania stanu. Nie musisz przepisywać całej odpowiedzialności do modułów.

Nie włączaj globalnego retry dla każdego błędu HTTP. Gałąź 429 ma odrębny limit i zwłokę, a malformed 2xx oraz timeout prowadzą do uncertain. Error workflow może obsłużyć zapis stanu i przygotować informację dla operatora, lecz samo wystąpienie błędu nie jest upoważnieniem do wysłania wiadomości ani publikacji. W laboratorium nie wysyłamy komunikatów zewnętrznych.

Przed przeniesieniem kontraktu sprawdź retencję execution history, osoby mające dostęp do wejścia oraz sposób eksportu konfiguracji. Uwierzytelnienie WordPressa przechowuj w mechanizmie credentials platformy. Nie kopiuj Application Password do widocznego pola webhooka, kodu przeglądarki ani dokumentu odbioru.

PROMPT DO CODEXA / Codex: plan mapowania platformy

Cel: porównaj implementację z kontraktem.

Dane: workflow-contract.json, N8N_MAKE_MAPPING.md i wersja n8n lub Make ustalona przez operatora. Wynik: plan ręcznej konfiguracji z dowodem dla każdego warunku oraz listą luk.

KONTROLA: Kontrola: nie nazywaj planu valid export. Zweryfikuj trwały klucz, atomowy claim, max 3 attempts i uncertain bez retry.

Import jest serią zleceń, nie skrótem

Każdy rekord importu przechodzi ten sam kontrakt co formularz. Dry-run pokazuje przygotowanie danych, a mała partia pozwala odebrać wynik przed większą pracą.

Import często zachęca do ominięcia walidacji, ponieważ plik wygląda na uporządkowany. Tymczasem każda pozycja wymaga własnego UUID, dozwolonych pól, niepustej treści i limitów UTF-8. Nie zmieniamy source na dowolny opis pochodzenia: w tym wydaniu pozostaje demo-form. Informacje o pliku i partii zapisz w dzienniku operatora. Rozszerzenie źródeł jest zmianą kontraktu, a nie swobodną wartością tekstową.

Przed załadowaniem partii obejrzyj próbkę w dry-run. Referencja wyświetla HTML MOCK, nie zapisuje do WordPressa i nie zmienia stanów. Weryfikujesz format, tytuły oraz rozszerzenie długości po escapingu. Następnie wykonaj niewielką partię na stagingu, sprawdzając każdy szkic. Dopiero po odbiorze mechaniki możesz rozważyć zwiększenie liczby zadań. Nie opisuj dry-run jako dowodu dostępności API.

Duplikaty importu trzeba rozwiązać przed uruchomieniem, zachowując tożsamość. Ponowne przyjęcie tego samego rekordu z tym samym UUID jest przewidywalne. Nowy UUID dla każdej powtórzonej próby omija tę ochronę i tworzy nową pracę. Jeżeli treść została poprawiona, zdecyduj, czy redagujesz istniejący szkic, czy świadomie zamawiasz kolejny. Nie ukrywaj poprawki jako retry starej operacji.

Nie dodajemy pobierania dowolnego URL ani prywatnych załączników. Dane są syntetyczne, a endpoint zapisuje tekst szkicu. Realny importer materiałów klienta potrzebowałby własnych reguł ACL, retencji i przekazywania do AI. Zachowaj zakres demonstratora, aby test odbioru miał jednoznaczny rezultat.

- Próbką: schemat, UUID i przewidywany HTML w dry-run.
- Mała partia: realne szkice stagingowe, kontrola każdego post_id.
- Powtórzenie: ten sam UUID i payload; poprawka treści wymaga świadomej decyzji.

Harmonogram nie jest gwarancją czasu

Worker można uruchamiać systemowo, a WordPress ma osobny WP-Cron. Oba mechanizmy trzeba monitorować; samo ustawienie minuty nie dowodzi terminowego wykonania.

Runner wykonuje najwyżej jedno gotowe zadanie na wywołanie. Scheduler systemowy może uruchamiać go co minutę w ustalonym katalogu i z dostępem do konfiguracji serwera. Nie wpisuj sekretu do crontaba. Zadbaj o użytkownika systemowego, prawa do bazy, log wyniku oraz kontrolę równoczesnych procesów. Krótka rezerwacja chroni rekord, ale monitoring procesu nadal jest potrzebny do rozpoznania awarii.

WP-Cron jest mechanizmem zadań WordPressa związanym z uruchamianiem przez ruch i nie gwarantuje punktualności przy braku odwiedzin. Jeżeli potrzebujesz przewidywalnego wyzwalania jego zadań, zastosuj konfigurację cron systemowego według oficjalnej dokumentacji i przetestuj ją oddzielnie. Nie zakładaj, że odpalenie Pythona automatycznie uruchomi kolejkę WP-Cron ani że ustawienie WP-Cron wykona nasz worker.

W naszym przepływie nie odkładamy tworzenia szkicu do WP-Cron. Intake odpowiada za bieżące przyjęcie i zapis, a kolejka zewnętrzna za przygotowanie oraz retry. To upraszcza odpowiedzialność i umożliwia odczyt stanu przed kontaktem z CMS. Dodatkowe zadania WordPressa wywołane przez wtyczki mogą jednak wpływać na środowisko, dlatego zapisujemy ich konfigurację w stagingowym dzienniku.

Próbę harmonogramu wykonaj obserwując czas `next_at`, moment uruchomienia i przejście stanu. Odbiór wymaga również scenariusza zatrzymanego schedulera: zadania powinny pozostać `queued` albo `retry`, a operator powinien zauważyć zaległość. „Cron skonfigurowany” jest opisem ustawienia, nie dowodem skutku.

Dwa odrębne harmonogramy

01 Scheduler systemowy

Uruchamia Python work

02 Kolejka SQLite

Wybiera zadanie po `next_at`

03 Intake WordPress

Bieżący zapis draft

04 WP-Cron osobno

Zadania WordPressa i własny test

Loguj przebieg, nie sekret

Diagnostyka ma pozwolić powiązać rekordy i rozpoznać etap awarii. Do tego zwykle wystarcza UUID, stan, numer próby, kod HTTP oraz czas.

Status wykonawcy pokazuje `request_id`, `state`, `attempts`, `post_id` i `reason`. Nie drukuje nagłówka `Authorization` ani `Application Password`. To wystarczający punkt startowy do odczytu kolejki i porównania z WordPressem. Pełna treść znajduje się w bazie, do której dostęp jest ograniczony. Nie kopiuj jej automatycznie do każdego narzędzia monitoringu, ponieważ mnożysz miejsca przechowywania bez poprawy diagnozy.

Ustal retencję logów oraz kopii. Nawet syntetyczne laboratorium uczy nawyku kontroli danych. W środowisku klienta tytuł może zawierać poufną informację, więc zastanów się, czy powinien występować w globalnym dashboardzie. Rola operatora potrzebuje technicznych śladów, a redaktor treści. W systemie z wieloma użytkownikami osobne ACL ograniczają wgląd do właściwych zadań. Ten lokalny demonstrator nie implementuje publicznego wieloużytkownikowego panelu.

Obserwuj liczbę `queued`, `retry`, `failed` i `uncertain` oraz wiek najstarszej pracy. Rosnące `queued` może oznaczać zatrzymany scheduler; wiele 401 wskazuje na `auth`; nagle `uncertain` może wiązać się z proxy lub filtrem odpowiedzi. Nie projektuj automatycznego resetu stanów jako alarmu. Powiadomienie ma pomóc operatorowi odczytać dowody, a nie wywołać kolejną niejasną operację.

W raporcie jakości rozdziel obserwację, hipotezę i decyzję. „Widzimy pięć `uncertain`” jest faktem z kolejki. „Przyczyną może być proxy” jest hipotezą. „Wstrzymujemy worker do sprawdzenia” jest decyzją. Ta kolejność ułatwia pracę zespołu bez dopisywania rzekomych wyników testów.

PROMPT DO CODEXA / Codex: plan obserwowalności

Cel: przygotuj minimalny raport operatora.

Dane: pola SQLite, status runnera i procedura uzgodnienia. Wynik: lista mierników, komunikatów oraz uprawnień do widoku, z rozdzieleniem faktu i hipotezy.

KONTROLA: Kontrola: wyklucz sekrety i pełne payloady z domyślnych logów; `uncertain` nie uruchamia automatycznego ponowienia.

PRO / AUTOMATYZACJE

Testy offline odpowiadają na konkretne pytania

Lokalny HTTP mock i tymczasowa baza rzeczywiście sprawdzają decyzje wykonawcy. Nie zastępują aktywacji PHP, capabilities WordPressa ani próby prawdziwego providera.

Uruchomiony zestaw obejmuje dziewiętnaście testów unittest. Sprawdza poprawne potwierdzenie draft, brak pola publikacji, lokalny oraz zdalny duplikat mocka, 429 ze zwłoką i limitem trzech prób, malformed 2xx, odrzucenie publikacji w wejściu i acknowledgement, recovery starego claim, rezerwację z dwóch połączeń, konflikt oraz limity, escaping MOCK i granice adresu HTTPS. Dodatkowo gateway odrzuca niepoprawny Content-Length oraz brak oczekiwanego Origin. Wszystkie zakończyły się OK. Siedem testów adaptera OpenAI kontroluje Structured Outputs, odmowy, limity, sekrety oraz mockowany transport AI→draft.

Dowód jest ograniczony do kodu Python oraz zachowania serwera testowego. Mock WordPress nie uruchamia dbDelta, wp_insert_post, wp_kses_post ani permission_callback. Nie potwierdza też obsługi Application Password przez hosting. Dlatego w TEST_STATUS.md jawnie zapisano niewykonane próby. Nazwa test_draft_forced odnosi się do kontraktu mocka i walidacji wykonawcy, a nie dowodzi wymuszenia draft przez PHP na stagingu.

Testy błędów mają dużą wartość, ponieważ pokazują następny krok operatora. Malformed 201 kończy jako uncertain i nie wykonuje drugiego HTTP. Stary inprogress również nie wraca automatycznie do pracy. Próba 429 potwierdza next_at i zatrzymanie po trzecim wykonaniu bez długiego czekania w teście. Zmieniamy czas gotowości bazy kontrolowanie, zamiast wykonywać wielominutowe sleeps.

Przed własną zmianą uruchom ten sam zestaw i zapisz wynik. Gdy modyfikujesz granice transportu, dodaj próbę odpowiadającą nowemu ryzyku. Nie buduj rozległych testów, które jedynie powtarzają stałe w kodzie; sprawdzaj zachowanie przy awarii oraz zgodność stanów.

Warstwa	Dowód w pakiecie
Python + SQLite	19 testów offline: OK
HTTP mock	Duplikat, 429, malformed i draft
WordPress/PHP	Plan stagingowy; nie wykonano
n8n/Make	Plan mapowania; nie wykonano
Rzeczywiste AI	Kompletny adapter; rzeczywiste API niewykonane

Staging sprawdza granice WordPressa

Odbiór wtyczki wymaga rzeczywistego CMS z odnotowanymi wersjami. Najpierw konto i schemat, następnie duplikaty, HTML oraz kontrolowana awaria zapisu.

Rozpocznij od kopii stagingowej i sprawdzenia odtworzenia. Zapisz wersję WordPressa, PHP, bazy, wtyczek i proxy. Aktywuj intake i sprawdź tabelę oraz rolę. Utwórz osobne konto bota, zweryfikuj jego efektywne capabilities i Application Password. Brak wykonanych testów PHP w pakiecie oznacza, że te czynności są pierwszym rzeczywistym odbiorem środowiska, a nie powtórzeniem wcześniej potwierdzonej instalacji.

Wykonaj poprawne zlecenie i potwierdź szkic w panelu. Następnie testuj odmowy: bez auth, bez wdl_ingest, z polem post_status, złym source, UUID oraz przekroczonym limitem. Sprawdź standardowy endpoint wpisów kontem bota; brak edit_posts powinien ograniczyć tę ścieżkę. Nie koryguj błędu przez dodanie praw administratora. Zapisz status, stan i dowód z panelu, unikając sekretu w załącznikach.

Powtórz identyczne zlecenie oraz konflikt z inną treścią. Uruchom dwa równoczesne żądania. Zbadaj filtrowanie HTML, cache i nagłówki. Dopiero potem przygotuj kontrolowaną awarię po tworzeniu wpisu albo utratę odpowiedzi. To szczególnie ważny etap: istniejący szkic i niepotwierdzona tabela powinny prowadzić do uzgodnienia bez automatycznego drugiego wpisu.

Na końcu redaktor akceptuje przykładowy szkic w zwykłym edytorze. Raport nie powinien ograniczać się do „API działa”. Wymagaj dowodu każdego warunku kontraktu i listy nierozwiązanych obserwacji. Jeśli środowisko nie przejdzie granicy publikacji albo idempotencji, nie zwiększaj skali importu.

- Przed zapisem: aktualne wersje, kopia, role, auth i cache.
- Po zapisie: jeden draft, poprawny UUID, duplikat i konflikt.
- Przy awarii: zachowany ślad, brak ślepego retry, uzgodnienie przez człowieka.

Odbierz kompletny projekt

Końcowy rezultat obejmuje mechanikę formularza, szkic i udokumentowaną decyzję człowieka. Liczba zielonych węzłów nie jest pełnym kryterium jakości.

Złóż odbiór z czterech dowodów. Pierwszy pokazuje, że poprawne wejście otrzymało UUID i trafiło do trwałej kolejki. Drugi potwierdza przygotowanie treści przez jawnie wskazany adapter oraz utrwalenie prepared. Trzeci wiąże UUID ze szkicem WordPressa. Czwarty zapisuje ocenę redaktora i decyzję publikacji. Każdy odpowiada na inne pytanie i nie powinien być zastępowany zrzutem jednego ekranu.

Dodaj dowody odmów i nieznanego wyniku. Bot nie ma prawa publikować; niepoprawny schemat jest odrzucany; identyczny duplikat nie tworzy nowego zadania; inny payload dla starego UUID daje konflikt. Timeout oraz malformed 2xx wymagają uncertain. Te zachowania powinny przejść próbę przed większym importem. W protokole oznacz, czy dowód pochodzi z mocka, stagingu czy manualnego odczytu.

README pozwala rozpocząć samodzielnie: Python i dane syntetyczne wystarczają do offline, a staging potrzebuje własnego aktualnego WordPressa z HTTPS. Pakiet nie zawiera rdzenia, realnego providera ani gotowej konfiguracji publicznego gateway. To materialne granice rezultatu, które powinny zostać widoczne w odbiorze. Nie uzupełniaj ich deklaracją „produkcyjne”, jeśli nie wykonałeś wymaganych prac.

Po odbiorze możesz poprawić wygodę operatora, obserwowalność albo format treści. Każdą zmianę wiąż z konkretnym problemem i testem. Nie rozszerzaj konta bota jako sposobu na szybszą konfigurację. Zachowaj szkic jako kontrolowany wynik i decyzję człowieka jako własny etap.

PROMPT DO CODEXA / Codex: przygotuj protokół odbioru

Cel: zamień kontrakt w sprawdzalny dokument.

Dane: README, TEST_STATUS.md, workflow-contract.json i własne obserwacje stagingowe.
Wynik: lista kryteriów, dowodów, właścicieli oraz nierozwiązanych spraw.

KONTROLA: Kontrola: oddziel wykonane mocki od WordPressa, nie dopisuj wyników. Odbiór wymaga draft, ograniczonego bota, duplikatu i procedury uncertain.

Redaktor sprawdza treść i pochodzenie

Szkic oszczędza część przygotowania tekstu. Nie przejmuje odpowiedzialności za fakty, ton, zgodność z ofertą ani dobór momentu publikacji.

Redaktor zaczyna od porównania tytułu, wejścia i gotowego szkicu. Sprawdza, czy tekst zachował sens opisu, nie dodał niepotwierdzonych obietnic i nie ukrył istotnych ograniczeń. W laboratorium treść jest syntetyczna, ale procedura powinna uczyć rozpoznawania twierdzeń wymagających źródła. Deterministyczny MOCK nie wymyśla rozwinięć; przyszły model może je wytworzyć, więc odbiór jakościowy trzeba wykonać osobno.

Sprawdź HTML, odnośniki i podgląd w motywie. Sam czytelny edytor nie pokazuje każdej konsekwencji prezentacji. Usuń zbędne elementy, popraw strukturę i upewnij się, że materiał nie sugeruje publicznego działania demonstratora. Przy publikacji używaj konta człowieka w zwykłym panelu. Nie przesyłaj nowego zlecenia do bota, aby uzyskać status publish, ponieważ taka operacja nie należy do kontraktu.

W notatce redakcyjnej zapisz UUID, post_id, datę odbioru i decyzję. Nie musisz przechowywać całej treści w osobnym logu, jeśli CMS ma właściwą historię zmian i kontrolę dostępu. Techniczne uzgodnienie uncertain nie oznacza automatycznej zgody na tekst. Odnalezienie szkicu zamyka pytanie o skutek operacji, a redaktor osobno zamyka pytanie o jakość materiału.

Jeżeli szkic wymaga znaczącej poprawki, możesz edytować go ręcznie lub świadomie zamówić kolejny. Zachowaj relację między zadaniami w dzienniku. Nie używaj starego UUID z inną treścią, ponieważ konflikt chroni właśnie tę historię. Przewidywalne zachowanie techniczne pomaga redaktorowi skupić się na materiale.

- Porównaj sens z wejściem; sprawdź niepotwierdzone ceny, terminy i obietnice.
- Obejrzyj HTML, odnośniki i podgląd; popraw treść w zwykłym edytorze.
- Zapisz decyzję człowieka z UUID i post_id; publikuj kontem redaktora.

Rozwijaj jedną granicę naraz

Kolejnym krokiem może być realny adapter, publiczny gateway albo większa kolejka. Każdy z nich wymaga własnych zasad, obserwacji i świadomego odbioru.

Wybierz rozszerzenie, które odpowiada rzeczywistej potrzebie. Jeżeli brakuje jakości tekstu, zaprojektuj adapter providera i próby na danych syntetycznych. Jeżeli problemem jest przyjmowanie danych z witryny, zaprojektuj uwierzytelniony gateway oraz limity. Jeżeli zalega dużo pracy, zbadaj scheduler i magazyn kolejki. Nie zmieniaj wszystkich elementów jednocześnie, bo trudniej będzie ustalić przyczynę nowego zachowania.

Przy realnym AI sprawdź aktualną oficjalną dokumentację, limit odpowiedzi, błędy transportu, ochronę sekretu i politykę danych. Wynik pozostaje propozycją, a WordPress nadal wymusza draft. Przy publicznym formularzu Origin nie zastępuje kontroli użytkownika. Przy wielu hostach SQLite tego demonstratora nie staje się automatycznie rozproszoną kolejką. Potrzebujesz atomowych operacji i trwałości dostosowanych do topologii.

Zachowaj neutralny kontrakt jako punkt porównania. Jeśli rozszerzenie zmienia pola, źródła lub odpowiedzialność, opisz nową wersję i migrację istniejących zadań. Nie generuj ponownie prepared podczas retry ani nie kasuj kluczy, żeby odblokować pracę. Po awarii nadal pytasz o znany wynik, dowód skutku oraz decyzję człowieka. To reguły niezależne od popularności wybranego narzędzia.

Projekt kończy się z działającym wykonawcą offline, kompletną wtyczką do odbioru stagingowego i uczciwym raportem testów. Dalsza praca powinna dopisywać dowody, a nie zwiększać liczbę obietnic. Gdy potrafisz wskazać właściciela każdego stanu i granicy uprawnień, możesz rozwijać integrację w sposób kontrolowany.

PROMPT DO CODEXA / Codex: plan następnego rozszerzenia

Cel: wyznaczyć mały sprawdzalny krok.

Dane: protokół odbioru i jeden konkretny problem operatora lub redaktora. Wynik: zakres zmiany, dotknięte pliki, kryteria odbioru i potrzebne źródła.

KONTROLA: Kontrola: zachowaj draft, ograniczone uprawnienia oraz uncertain bez retry. Nie deklaruj wykonania nowych testów przed ich uruchomieniem.

Bibliografia i zakres dowodów

Oficjalne punkty odniesienia do auth, własnego endpointu, trwałej tabeli i harmonogramów. Zastosowanie zasad sprawdź na własnym stagingu z odnotowanymi wersjami.

Authentication — REST API Handbook

<https://developer.wordpress.org/rest-api/using-the-rest-api/authentication/>

Cookie authentication, nonce REST oraz Application Password dla zewnętrznych aplikacji.

Adding Custom Endpoints — REST API Handbook

<https://developer.wordpress.org/rest-api/extending-the-rest-api/adding-custom-endpoints/>

Rejestracja tras, namespace, callback i sprawdzanie uprawnień.

Creating Tables with Plugins — Plugin Handbook

<https://developer.wordpress.org/plugins/creating-tables-with-plugins/>

Tabela wtyczki, prefiks oraz dbDelta; wdrożenie wymaga testu w rzeczywistej bazie.

Understanding WP-Cron Scheduling — Plugin Handbook

<https://developer.wordpress.org/plugins/cron/understanding-wp-cron-scheduling/>

Wpływ ruchu na wykonanie zadań; brak gwarancji punktualności.

Hooking WP-Cron Into the System Task Scheduler

<https://developer.wordpress.org/plugins/cron/hooking-wp-cron-into-the-system-task-scheduler/>

Konfiguracja wyzwalania WP-Cron przez harmonogram systemowy.

Error handling — n8n Docs

<https://docs.n8n.io/flow-logic/error-handling/>

Obsługa błędów workflow; nie stanowi gwarancji idempotencji.

HTTP Request node — n8n Docs

<https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.httprequest/>

Konfiguracja transportu; opcje oraz retry potwierdź dla używanej wersji.

Structured model outputs — OpenAI API

<https://developers.openai.com/api/docs/guides/structured-outputs?api-mode=responses>

Responses API, text.format z json_schema, strict i obsługa odmowy lub incomplete. Rzeczywiste API niewykonane.

wp_insert_post() — WordPress Code Reference

https://developer.wordpress.org/reference/functions/wp_insert_post/

Kontrakt slashed tablicy i wp_unslash przed zapisem; dlatego granica intake używa wp_slash.

Bibliografia wskazuje oficjalne instrukcje potrzebne do wdrożenia i odbioru. Dokumentacja może zmieniać szczegóły interfejsu, dlatego podczas konfiguracji sprawdź aktualne strony oraz zapisz konkretne wersje. Link nie oznacza, że test stagingowy został wykonany. Dowody lokalne, zakres mocka i niewykonane próby opisuje TEST_STATUS.md. Przy każdym teście stagingowym zapisz datę, wersję środowiska, wejście z nowym UUID, oczekiwany status oraz zaobserwowany stan. Linki pomagają sprawdzić zasady, lecz nie zastępują protokołu odbioru własnego wdrożenia. Dostęp do raportu ogranicz tak samo jak dostęp do kolejki. W tym tomie nie użyto rzeczywistego providera AI i nie przytaczamy rzekomych wyników jego działania.