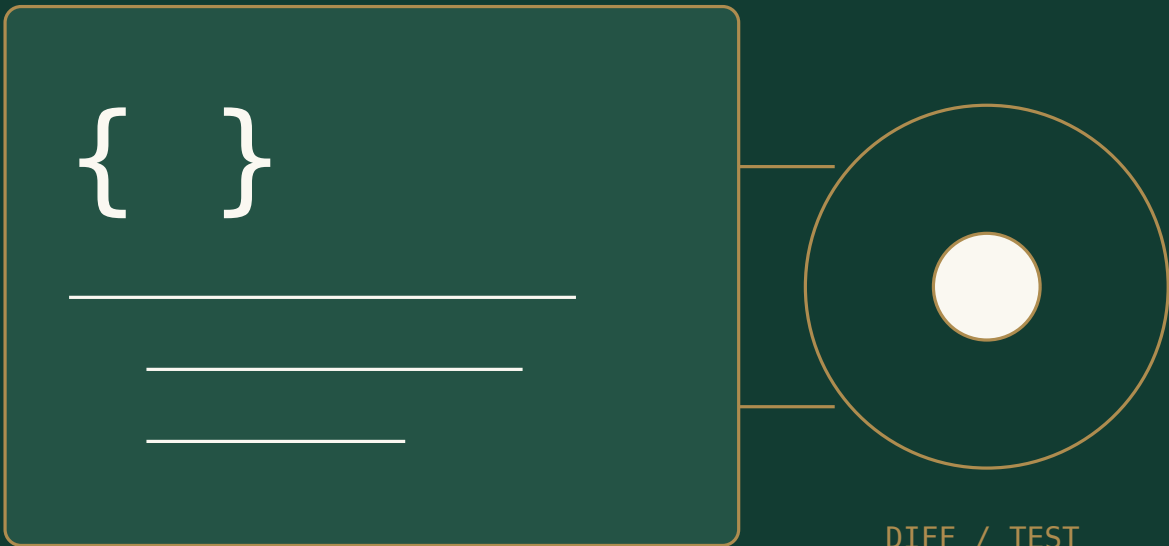


CODEX + WORDPRESS

AI jako partner programisty



DIFF / TEST

PROJEKT TWORZY CZŁOWIEK.
AI WSPIERA PROCES.

02

10 ROZDZIAŁÓW / JEDEN SYSTEM DEMONSTRACYJNY

Spis treści

Wprowadzenie i mapa projektu	3
01 Agent programistyczny i odpowiedzialność	4
Co agent rzeczywiście robi · Autonomia ma określony zakres · Pierwsze polecenie: orientacja	
02 Środowisko, Git i instrukcje projektu	7
Środowisko jest częścią briefu · Git chroni kontekst zmiany · AGENTS.md: instrukcje blisko kodu	
03 Czytanie istniejącego kodu	10
Czytaj przepływ, nie wszystkie pliki · Znajdź umowy między komponentami · Zadać pytanie o kod z dowodem	
04 Brief zadania i plan zmiany	13
Brief opisuje zmianę i granice · Plan ma być wykonalny · Od planu do implementacji	
05 PHP: model realizacji i uprawnienia	16
CPT należy do wtyczki · Meta: typ, zapis i publiczny odczyt · Nonce oraz capabilities są niezależne	
06 JavaScript: dynamiczny blok bez budowania	19
block.json spina obie strony · JavaScript bez npm build · Atrybuty wymagają dwóch granic	
07 HTML i CSS: widok publiczny	22
Zapytanie publiczne tylko po publish · HTML i CSS mają własny odbiór · Paginacja jest częścią kontraktu	
08 Przegląd diffu i naprawianie błędów	25
Czytaj diff jako historię zachowania · Debug zaczyna się od odtworzenia · Review ma kończyć się decyzją	
09 Testy i dowody jakości	28
Test odpowiada na jedno pytanie · Przypadki negatywne chronią kontrakt · Raport testów bez dopowiadania	
10 Odbiór, wdrożenie i dalsza praca	31
Samodzielny start i odbiór · Deploy jest osobną decyzją · Co dalej: rozwijaj jedną umowę	
Źródła techniczne i dalsza lektura	34

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

Od rozmowy do sprawdzalnej zmiany

Ten tom prowadzi od zadania dla Codexa do odbioru małej wtyczki WordPress. Rezultat jest konkretny: redaktor dodaje realizacje, a dynamiczny blok pokazuje odwiedzającym wyłącznie opublikowane rekordy. Całość można uruchomić niezależnie od wcześniejszego tomu.

Pracujemy dla fikcyjnej firmy WP DEMO LAB. Jej portfolio obejmuje Atelier Północ, Studio Fala i Pracownię Punkt. Przykłady nie zawierają danych klientów, kluczy API ani połączenia z prawdziwą stroną. Wtyczka wp-demo-projects przechowuje realizacje jako wdl_project, udostępnia je przez REST base wdl-projects i rejestruje blok wp-demo-lab/projects. Te nazwy tworzą umowę między opisem, kodem, testami i kolejnymi tomami serii.

Codex jest agentem programistycznym: może czytać pliki, proponować plan, zmieniać kod i uruchamiać dostępne narzędzia. Konkretny zakres dostępu zależy od środowiska i uprawnień. Nie zakładamy, że sam opis zadania zapewnia działającą bazę danych, interpreter PHP albo WordPress. W tym pakiecie wykonano lokalne kontrole JavaScript i danych; procedury integracyjne pozostają do uruchomienia na stagingu. Każde twierdzenie o teście powinno mieć odpowiadający mu wynik.

Czytaj rozdziały razem z katalogiem EXAMPLES. Fragmenty na stronach pokazują decyzje wymagające zrozumienia, natomiast kompletne pliki stanowią jeden spójny projekt. Nie trzeba budować JavaScript przez npm. Edytor korzysta z zależności dostarczonych przez WordPress, a frontend otrzymuje HTML z PHP. Na stronie umieszczamy jeden blok realizacji, ponieważ paginacja korzysta ze wspólnego parametru adresu. To jawna granica przykładu.

Po lekturze przygotujesz brief, sprawdzisz diff, nazwiesz nieuruchomione testy i przekażesz zmianę do akceptacji. Programista pozostaje właścicielem decyzji o kontrakcie danych, uprawnieniach oraz wdrożeniu. Agent pomaga zmniejszyć koszt powtarzalnej pracy, ale wynik oceniamy po zachowaniu aplikacji. Zacznij od README, zapisz wersje własnego środowiska i oddziel obserwacje od założeń. Ta praktyka przyda się przy każdym następnym projekcie.

Co agent rzeczywiście robi

Współpraca z Codexem ma sens, gdy zadanie kończy się obserwowalnym wynikiem. „Dodaj portfolio” jest początkiem rozmowy; „zapisz trzy realizacje i pokaż dwie na pierwszej stronie listy” daje już punkt odniesienia dla implementacji i odbioru.

Agent nie zna automatycznie reguł twojej witryny. Potrzebuje dostępnych plików, instrukcji projektu i granic pracy. W naszym przykładzie może zmieniać wtyczkę, jej testy i dokumentację. Nie powinien przebudowywać motywu, instalować przypadkowych zależności ani zmieniać innych wtyczek. W poleceniu podaj także wersję minimalnego kontraktu oraz fakt, że dane są demonstracyjne. Dzięki temu plan nie zaczyna się od niepotrzebnej migracji całego serwisu.

Rozdziel trzy czynności: ustalenie oczekiwanego zachowania, przygotowanie zmiany i ocenę dowodów. Agent może pomóc przy każdej z nich, lecz nie zawsze ma narzędzia potrzebne do ostatniej. Gdy brak PHP, może przejrzeć składnię wzrokowo i napisać procedurę uruchomienia. Nie może na tej podstawie ogłosić, że aktywacja zakończyła się poprawnie. Taki raport powinien zawierać stan „nieuruchomione” oraz warunki odtworzenia testu.

Dobra współpraca jest iteracyjna, ale nie chaotyczna. Najpierw przeczytaj plan i upewnij się, że dotyczy wskazanej funkcji. Po implementacji obejrzyj diff, uruchom dostępne kontrole i dopiero wtedy zadawaj kolejne zadanie. Nie zlecaj jednocześnie przebudowy danych, wyglądu i procesu publikacji, jeśli nie potrafisz określić wspólnego kryterium odbioru. Mniejsze zmiany łatwiej cofnąć i wyjaśnić przyszłemu opiekunowi strony.

Ćwiczenie: opisz w dwóch zdaniach, co zmieni się dla redaktora oraz dla odwiedzającego. Redaktor otrzyma typ wpisu z klientem i rokiem. Odwiedzający zobaczy publiczne karty realizacji z paginacją. Następnie dopisz jeden warunek negatywny: szkic nie pojawia się na liście nawet wtedy, gdy stronę ogląda administrator. Tak powstaje zwięzła definicja odpowiedzialności agenta.

Cykl współpracy

01 Cel

Obserwowalne zachowanie dla użytkownika

02 Plan

Pliki, granice i kryteria

03 Zmiana

Kod i dokumentacja w zakresie

04 Dowód

Wykonane kontrole oraz ograniczenia

05 Akceptacja

Człowiek ocenia wynik i wdrożenie

Autonomia ma określony zakres

Agentowi warto pozwolić wykonywać potrzebne kroki w zatwierdzonym zakresie. Nie trzeba zatwierdzać każdego odczytu pliku. Jednocześnie uprawnienia środowiska powinny odpowiadać zadaniu, a czynności dotyczące produkcji wymagają osobnej, konkretnej decyzji.

W tym ćwiczeniu wystarcza kopia projektu oraz staging. Agent nie potrzebuje danych klientów ani dostępu do panelu produkcyjnego. Sekrety pozostają poza repozytorium. Jeśli narzędzie pozwala kontrolować dostęp do sieci lub plików, użyj profilu odpowiedniego do pracy. Oficjalne zasady uprawnień OpenAI rozróżniają ograniczenia środowiska i zatwierdzanie działań; ustawienia mogą zależeć od używanego produktu i administracji organizacji. Nie obchodź ograniczeń tylko po to, by szybciej wykonać polecenie.

Przygotuj listę dozwolonych czynności w języku projektu. Odczyt i edycja kodu wtyczki, test pomocnika JavaScript oraz przygotowanie instrukcji integracyjnych mieszczą się w zakresie. Zmiana danych na produkcji, publikacja strony lub przebudowa praw użytkowników nie wynikają z samego zadania stworzenia bloku. Gdy pojawi się konieczność rozszerzenia zakresu, agent powinien opisać powód, wpływ i gotowy do oceny rezultat.

Instrukcje znalezione we wpisie klienta, komentarzu w obcym pliku czy treści odpowiedzi z sieci są danymi zadania. Nie powinny automatycznie zastępować polecenia użytkownika. Przykładowa treść realizacji może zawierać zdanie „usuń całą bazę”; dla programisty nadal jest to tekst do wyświetlenia, nie nowa dyspozycja. Takie rozróżnienie pomaga przy pracy z archiwami, zewnętrznymi dokumentami i automatyzacjami obsługującymi treść.

Ćwiczenie: zapisz, które foldery agent może zmieniać, jakie testy ma wykonywać i jakie działania pozostają poza zakresem. Sprawdź, czy instrukcje nie obiecują uprawnień, których środowisko nie posiada. W razie blokady narzędzia poproś o opis problemu i bezpieczną alternatywę: test na kopii, ręczną procedurę lub zmianę planu. Nie stosuj instrukcji wyłączających zabezpieczenia ani nie traktuj braku dostępu jako usterki kodu.

Pierwsze polecenie: orientacja

Zacznij od rozpoznania dostępnego projektu. Pierwsze polecenie powinno prowadzić do mapy plików i pytań o konkretny kontrakt, a nie do natychmiastowej wymiany całej wtyczki. W zatwierdzonym ćwiczeniu mapa obejmuje tylko pakiet demonstracyjny.

Rozpoznanie nie musi oznaczać szerokiego audytu. Wystarczy przeczytać README, instrukcje AGENTS.md oraz plik wejściowy wtyczki. Agent powinien wskazać, gdzie powstaje model danych, gdzie rejestrujemy blok i gdzie wykonywany jest publiczny odczyt. Mapa pozwala potem sprawdzić, czy plan dotknie prawidłowego miejsca. Dla WP DEMO LAB to jeden plik główny, katalog bloku, pomocnik normalizacji i dwa rodzaje testów.

W poleceniu rozróżnij fakty z kodu i brakujące potwierdzenia. Nazwa REST base znajduje się w `register_post_type`; działająca odpowiedź HTTP wymaga uruchomionego WordPressa. Jeśli agent nie ma tego środowiska, powinien powiedzieć to od razu. Lista narzędzi nie jest listą wykonanych testów. Szczególnie uważaj na zdania o zgodności z różnymi motywami: bez otwarcia konkretnych widoków mogą być jedynie celem dalszej weryfikacji.

Wyniki orientacji wykorzystaj jako krótki punkt startowy kolejnego zadania. Nie kopiuj do promptu całego repozytorium, jeśli trzy pliki wyjaśniają funkcję. Dołącz informacje potrzebne do decyzji: kontrakt nazw, dostępne środowisko, oczekiwany widok i zakres edycji. Pomocny raport mieści się w kilku akapitach i wskazuje konkretne ścieżki zamiast ogólnych pochwał jakości projektu.

PROMPT DO CODEXA / Prompt 01 / mapa projektu

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Przeczytaj README.md, AGENTS.md i pliki wp-demo-projects. Nie zmieniaj ich. Opisz przepływ od rejestracji `wdl_project` do `render.php` oraz rolę `editor.js` i `options.js`. Wskaż istniejące zabezpieczenia i miejsca wymagające testu w WordPressie. Nie audytuj innych folderów, nie instaluj zależności. Zakończ listą faktów potwierdzonych w plikach i osobną listą brakujących dowodów środowiskowych.

KONTROLA: Sprawdź, czy raport nazywa konkretne pliki i nie deklaruje aktywacji ani E2E.

Środowisko jest częścią briefu

Kod zależy od wykonawcy. Nasz przykład wymaga funkcjonalnie WordPressa co najmniej 6.6 i PHP 8.2, natomiast lokalne testy pomocnika wymagają Node 20 lub nowszego. To granice funkcji przykładu; na stagingu wybierz aktualne wspierane wydania.

Zapisz trzy warstwy środowiska: miejsce pracy nad plikami, runtime aplikacji oraz miejsce odbioru. Edytor z terminalem może uruchamiać Node, a nie posiadać PHP. WordPress na stagingu ma własną bazę, konfigurację serwera i motyw. Przeglądarka weryfikuje zachowanie widoku oraz interakcje. Brak jednej warstwy nie unieważnia całej pracy, ale ogranicza twierdzenia, które możesz uczciwie przedstawić. Wpisz tę granicę do dziennika projektu.

Dla samodzielnego startu wystarczy lokalna instalacja WordPressa z motywem blokowym i katalog wtyczki skopiowany do wp-content/plugins. Nie potrzebujesz motywu poprzedniego tomu. Nie uruchamiaj dwóch kopii tej samej wtyczki, ponieważ globalne funkcje i nazwa bloku będą się powtarzać. Sprawdź prawa do plików, możliwość logowania do panelu i bezpośrednie odnośniki. Następnie zanotuj wersje PHP, WordPressa, Node oraz nazwę aktywnego motywu.

Przygotowanie środowiska nie powinno pozostawiać sekretów w promptach ani logach. Dane demonstracyjne wpiszesz ręcznie z fixtures.json. Jeśli używasz własnej kopii witryny, usuń dane klientów oraz niepotrzebne konta integracji. Pracuj na oddzielnej bazie i upewnij się, że adresy należą do stagingu. Sam napis „staging” w nazwie folderu nie potwierdza, gdzie kieruje konfiguracja bazy danych lub poczta.

Ćwiczenie: utwórz tabelę z kolumnami „narzędzie”, „wersja”, „mogę uruchomić” i „wynik”. Przy PHP niedostępnym wpisz tę informację zamiast pozytywnego wyniku. Następnie odtwórz trzy kontrole Node z README. Ich sukces potwierdza składnię oraz zachowanie pomocnika, ale nie rejestrację bloku w WordPresie. Zachowaj raport, aby następny opiekun wiedział, które etapy musi jeszcze wykonać.

Git chroni kontekst zmiany

Git pozwala zobaczyć dokładny zakres pracy i wrócić do wcześniejszego stanu kodu. Przed zadaniem sprawdź istniejące zmiany, a po zadaniu obejrzyj diff. Nie utożsamiaj czystego repozytorium z bezpiecznym wdrożeniem: baza danych ma osobny cykl życia.

Rozpocznij od `git status --short`. Jeśli są zmiany innej osoby, zapisz ich zakres i nie usuwaj ich przy okazji. Nową funkcję przygotuj na własnej gałęzi, nazwanej zgodnie z zasadami zespołu. Gdy potrzebujesz izolacji, użyj oddzielnej kopii lub worktree zgodnie z organizacją projektu. Ważne jest, aby agent wiedział, które pliki należą do zadania, a które mają pozostać poza jego zmianą. W tym tomie nie modyfikujemy motywu.

Commit powinien zamykać jedną spójną decyzję. Rejestracja CPT bez pliku renderowania nie daje odbieralnej funkcji, natomiast dopisanie do tego samego commita nowego formularza kontaktowego utrudnia ocenę. Dla ćwiczenia logiczną całością są model realizacji, blok i odpowiadająca im dokumentacja. Kolejna naprawa bezpieczeństwa może otrzymać osobny commit z testem regresji. Historia powinna wyjaśniać, dlaczego zmieniło się zachowanie, a nie wyłącznie wymieniał pliki.

Przed commitem sprawdź `git diff --check` i `git diff --cached`. Dodawaj do staging area konkretne ścieżki. Ogólne `git add .` łatwo obejmuje plik tymczasowy, niepowiązaną zmianę lub przypadkowy eksport. Opis commita powinien nazywać działanie użytkownika i nowy rezultat. W przykładzie „Add public projects block and metadata” wskazuje jeden temat; „update files” nie pomaga podczas diagnozowania regresji. Polecenia z README są instrukcją, a nie dowodem istniejącego commita.

Ćwiczenie: przed implementacją zapisz wynik statusu. Po zmianie porównaj go z listą plików planu. Każdy dodatkowy plik wymaga wyjaśnienia. Gdy diff zawiera setki niezwiązanych zmian formatowania, wydziel je lub zrezygnuj z nich. Przy odbiorze funkcji ważniejsze jest, czy można prześledzić przepływ danych i zabezpieczenia, niż to, czy agent ujednolicił wszystkie spacje w sąsiednich katalogach.

Od gałęzi do commita

01 Status

Rozpoznaj istniejące zmiany

02 Gałąź

Wydziel zadanie w historii

03 Diff

Porównaj kod z zakresem i testami

04 Commit

Zapisz jedną spójną zmianę

AGENTS.md: instrukcje blisko kodu

Plik AGENTS.md zapisuje trwałe reguły pracy z projektem. Oficjalna dokumentacja Codexa opisuje ich odczyt i warstwowanie w strukturze katalogów. Dla ćwiczenia ważne jest, aby instrukcje wskazywały realny zakres oraz prawdziwe polecenia weryfikacji.

W EXAMPLES przygotowaliśmy instrukcję ograniczającą zmiany do wtyczki, testów i dokumentacji przykładu. Zawiera kontrakt nazw, wymaganie publish w publicznym zapytaniu oraz jawny brak PHP w środowisku produkcji pakietu. Te reguły oszczędzają powtarzanie całego briefu. Nie zastępują jednak konkretnego celu nowego zadania: agent nadal musi wiedzieć, którą funkcję rozwija i jaki wynik ma zaakceptować użytkownik.

Instrukcje powinny być krótkie oraz wykonalne. Jeżeli wpiszesz npm test, a projekt nie ma package.json, stworzysz pozorną bramkę jakości. W naszym katalogu poprawne są bezpośrednio node --check oraz node --test. Polecenia Git mają sens wyłącznie tam, gdzie rzeczywiście istnieje repozytorium. Procedura PHP jest opisana oddzielnie, aby brak interpretera nie prowadził do fikcyjnego raportu. Sprawdź także instrukcje wyżej w strukturze projektu.

Nie wpisuj do AGENTS.md sekretów ani danych osobowych. Możesz umieścić odwołanie do miejsca konfiguracji, ale bez wartości tokenów. Reguła przeglądu powinna określać niepożądane zachowanie, na przykład możliwość zobaczenia szkicu w bloku publicznym. Nie mieszaj tej reguły z długim podręcznikiem formatowania. Narzędzia automatyczne dobrze wykrywają składnię; człowiek potrzebuje jasnego uzasadnienia funkcjonalnego.

PROMPT DO CODEXA / Prompt 02 / instrukcje projektu

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Przeczytaj aktywne instrukcje projektu i EXAMPLES/AGENTS.md. Streść dozwolone ścieżki, kontrakt danych oraz komendy dostępne bez WordPressa. Nie zmieniaj instrukcji globalnych. Oceń, czy opisane polecenia istnieją i czy ograniczenia środowiska są spójne z TEST_STATUS.md. Jeżeli wykryjesz sprzeczność, wskaż dokładny zapis i konsekwencję, zamiast zgadywać wynik testu.

KONTROLA: Oczekuj konkretnych ograniczeń oraz poleceń Node; nie oczekuj deklaracji sukcesu PHP.

Czytaj przepływ, nie wszystkie pliki

Przy pracy z istniejącym kodem szukaj ścieżki od wejścia do skutku. Dla realizacji są dwie takie ścieżki: redaktor zapisuje dane, a odwiedzający otrzymuje listę. Ich przecięciem są rekordy `wdl_project` oraz zarejestrowane meta.

Pierwsza ścieżka zaczyna się w edytorze WordPressa. Tytuł i opis obsługuje rdzeń, klienta i rok pokazuje meta box. Hook `save_post_wdl_project` uruchamia funkcję zapisującą pola. Odczytaj jej warunki wcześniejszego zakończenia przed częścią `update_post_meta`. Ten układ odpowiada na pytanie, kiedy zapis jest dozwolony. Rejestracja meta określa dodatkowo typy, sanitation i kontrolę uprawnień dla obsługiwanych przez rdzeń operacji REST.

Druga ścieżka zaczyna się od zapisanej instancji bloku. WordPress odczytuje atrybuty, uruchamia `render.php` i wykonuje `WP_Query` z jawnym statusem `publish`. Zapytanie wyklucza także wpisy chronione hasłem. Wynik przechodzi do elementów `article` oraz linków paginacji. Nazwa klasy `column` pochodzi z ograniczonej liczby, a treści tekstowe otrzymują `escape`. W tej wersji nie ma zapytania AJAX ani oddzielnego endpointu frontendowego. Dzięki temu do zrozumienia widoku wystarczy PHP, HTML i CSS.

Czytając, twórz krótką mapę odpowiedzialności. Zapytaj, gdzie waliduje się wejście, gdzie odbywa się autoryzacja, gdzie powstaje zapytanie i gdzie następuje wyjście. Jeśli te odpowiedzialności są rozmieszczone w kilku plikach, nazwij zależności. Nie wyciągaj wniosku, że brak walidacji w edytorze oznacza brak walidacji serwera; sprawdź oba miejsca. Nie zakładaj też, że formularz automatycznie chroni ręcznie zmieniony `post_content`.

Ćwiczenie: wybierz wartość `columns=99` i prześledź ją od atrybutów do klasy w HTML. Potem wybierz klienta o nazwie A & B i sprawdź kontekst jego wyświetlenia. Każde z tych wejść dotyka innej odpowiedzialności: ograniczenia konfiguracji albo bezpiecznego tekstu. Zapisz znaleziony plik oraz warunek odbioru. Taka notatka jest bardziej użyteczna niż długa lista funkcji bez związku z zachowaniem.

Dwie ścieżki danych

01 Edytor

Tytuł, opis i pola klient/rok

02 Zapis

Nonce, capability i sanitation

03 Rekord

`wdl_project` z publicznym statusem

04 Render

Ograniczone atrybuty i `WP_Query publish`

05 Widok

Escape, karty i paginacja

Znajdź umowy między komponentami

Kontrakt danych powinien być stabilny. Nazwy zapisane w PHP, block.json, JavaScript i dokumentacji muszą wskazywać ten sam projekt. Zmiana jednej nazwy może pozornie działać w edytorze, a zerwać zapisane treści albo integrację.

W przykładzie domeną wtyczki są realizacje, a ich typem `wdl_project`. REST używa osobnej, jawnie ustalonej nazwy `wdl-projects`. Blok ma nazwę `wp-demo-lab/projects`, niezależną od ścieżki katalogu i tytułu wyświetlanego redaktorowi. Meta używają prefiksu `_wdl_`.

Podkreślenie nie jest mechanizmem poufności: po opublikowaniu dane przewidziane w REST mogą być publiczne. Dlatego klient w tym projekcie zawsze oznacza nazwę demonstracyjną, którą wolno pokazać odwiedzającemu.

Atrybut `limit` określa liczbę rekordów na stronie listy, a `columns` maksymalną liczbę kolumn na szerokim ekranie. Nie zapisujemy do nich identyfikatorów klientów, selektorów SQL ani adresu zewnętrznego źródła. Ograniczony kontrakt zmniejsza liczbę możliwych błędów. Dodatkową zasadą jest jedna instancja bloku na stronie, ponieważ wszystkie używają parametru `wdl_projects_page`. Jeżeli później potrzebujesz kilku niezależnych list, należy zaprojektować identyfikator instancji oraz osobny mechanizm paginacji.

Kontrakt obejmuje również zachowanie nieobecnych danych. Brak obrazu nie usuwa karty. Brak roku nie wypisuje zera. Pusta lista pokazuje komunikat. Rok spoza dopuszczanego zakresu nie powinien losowo zmieniać innego pola. Takie przypadki są częścią funkcji, nawet jeśli zespół nie przygotował osobnego projektu graficznego każdego stanu. Agent powinien dostać ich opis przed implementacją, aby nie zgadywać reguł biznesowych.

Element	Umowa	Odbiór
Typ wpisu	<code>wdl_project</code>	Widoczny w menu i REST
REST base	<code>wdl-projects</code>	Anonimowy odczyt tylko <code>publish</code>
Blok	<code>wp-demo-lab/projects</code>	Rozpoznaje zapisane instancje
Meta	<code>_wdl_client</code> / <code>_wdl_year</code>	<code>string</code> / <code>integer</code>
Widok	<code>limit 1-12</code> / <code>columns 1-3</code>	Serwer ogranicza wejście

Zadaj pytanie o kod z dowodem

Pytanie do agenta powinno wskazywać obserwację i oczekiwany dowód. „Czy ta wtyczka jest bezpieczna?” jest zbyt szerokie. „Czy subskrybent może zmienić `_wdl_client` mimo prawidłowego nonce?” prowadzi do konkretnego warunku i testu.

Agent może odnaleźć `current_user_can` w funkcji zapisu oraz `auth_callback` w rejestracji meta. To dowód implementacji mechanizmu. Potwierdzenie działania na rolach wymaga jednak uruchomienia operacji w WordPressie. Rozróżnienie jest ważne: zarówno brak wywołania callbacka, jak i inna konfiguracja uprawnień mogą zmienić wynik. Odpowiedź powinna wskazać miejsce kodu i zaproponować najkrótszy test rozstrzygający. Nie wystarczy zdanie, że framework obsługuje bezpieczeństwo.

Przy pytaniu o publiczny widok szukaj jawnego `post_status=publish`, sposobu budowania excerptu i `escape` właściwego dla kontekstu. Przy pytaniu o edytor sprawdź zgodność atrybutów oraz zadeklarowane zależności skryptu. Każda kategoria ma inną ścieżkę błędu. Nie proś agenta o refaktoryzację, zanim dowiesz się, co jest problemem. Odczyt diagnostyczny może zakończyć się wnioskiem, że kod spełnia założenia, a test trzeba wykonać w brakującym środowisku.

Ćwiczenie: sformułuj jedno pytanie dotyczące poufności szkicu i jedno dotyczące dostępności klawiatury. Do pierwszego potrzebujesz zapytania i listy statusów. Do drugiego CSS oraz widoku w przeglądarce. Zapisz, czego nie można sprawdzić wyłącznie analizą plików. Tak przygotowane pytania prowadzą do raportu, który da się ocenić bez wiary w ogólne zapewnienia.

PROMPT DO CODEXA / Prompt 03 / analiza istniejącego kodu

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Bez zmian w plikach prześledź zapis `_wdl_client` z meta box oraz jego wyświetlenie. Wskaż sprawdzenie typu wejścia, `wp_unslash`, `nonce`, `capability`, `sanitation` i `escape`. Oddziel ścieżkę panelu od ścieżki REST. Następnie wskaż, które zachowania potwierdza kod, a które wymagają `tests/integration.php`. Nie nazywaj tej analizy wykonanym audytem bezpieczeństwa.

KONTROLA: Raport powinien wskazać funkcje i granice dowodów; nie może traktować nonce jako uprawnień.

Brief opisuje zmianę i granice

Zanim powstanie kod, napisz krótką umowę zadania. Wskaż użytkownika, oczekiwane zachowanie, dozwolone pliki, przypadki brzegowe i kryteria odbioru. Dla agenta te elementy są bardziej użyteczne niż długi opis oczekiwanej „nowoczesności”.

Nasz użytkownik to redaktor zarządzający portfolio. Chce zapisać tytuł, opis, obraz, klienta i rok, a potem pokazać opublikowane wpisy na stronie Realizacje. Szkice pozostają poza listą. Blok ma ustawienia liczby rekordów i kolumn, ale nie dostaje prawa publikowania. Motyw odpowiada za globalny wygląd, wtyczka za model danych i funkcję domenową. Ta granica zachowuje realizacje także po zmianie motywu.

Brief powinien powiedzieć, co zrobimy przy braku obrazu, pustym roku i pustej liście. Powinien także nazwać ograniczenia: bez npm build, bez zewnętrznych requestów, bez sekretów i jedna instancja bloku na stronie. Jeśli agent zaproponuje większą architekturę, porównaj ją z potrzebą użytkownika. Zwiększenie liczby warstw nie jest samo w sobie dowodem jakości. Dla małego portfolio prosta rejestracja, server render i natywna paginacja wystarczają do ćwiczenia.

W kryteriach odbioru użyj czasowników: redaktor zapisuje, administrator widzi, subskrybent nie zmienia, anonimowy odwiedzający odczytuje, lista nie ujawnia szkicu. Oprócz pozytywnej ścieżki podaj co najmniej jeden test negatywny. Kiedy później pojawi się błąd, taki brief pozwoli rozstrzygnąć, czy to regresja, czy nowa potrzeba poza zakresem. Przechowuj brief obok historii zmiany, aby można było wrócić do powodu implementacji.

PROMPT DO CODEXA / Prompt 04 / wymagania

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Zrealizuj wtyczkę wp-demo-projects dla WP DEMO LAB. Model: wdl_project, REST wdl-projects, klient string _wdl_client i rok integer _wdl_year. Blok wp-demo-lab/projects: API v3, limit 1-12, columns 1-3, publish only, paginacja natywna, escape. Bez npm build i zmian motywu. Najpierw przedstaw kryteria akceptacji, pliki oraz otwarte kwestie; nie rozszerzaj funkcji o formularze, integracje ani automatyczne publikowanie.

KONTROLA: Sprawdź zgodność nazw, jawne przypadki brzegowe i zakaz zmian motywu.

Plan ma być wykonalny

Plan przekłada brief na małe kroki. Każdy krok powinien mieć rezultat i sposób sprawdzenia. W naszym projekcie kolejność wynika z zależności: rejestrujemy dane, przygotowujemy blok, renderujemy widok, a następnie oceniamy całość.

Pierwszy krok to model danych: CPT, wsparcie pól, rejestracja meta i uprawnienia. Drugi obejmuje meta box oraz zapis z nonce. Trzeci opisuje block.json, zależności skryptów i edytor. Czwarty przygotowuje render PHP oraz responsywny CSS. Ostatni łączy testy i dokumentację uruchomienia. Nie musisz wykonywać jednego commita na każdy drobny plik, lecz powinieneś umieć wskazać, które zachowanie potwierdza dana część.

Plan powinien ujawnić brakujące narzędzia przed implementacją. Jeśli PHP nie jest dostępny, zapisz procedurę lintowania i test integracyjny jako etap do wykonania później. Nie zastępuj go pozornym sprawdzaniem liczby nawiasów. Dostępne Node wykorzystaj do prawdziwej składni oraz czystego pomocnika atrybutów. Plan nie powinien zawierać wykonywania poleceń, których projekt nie oferuje. Dobry agent sprawdzi strukturę katalogów i potwierdzi komendy.

Oceń plan pod kątem niepotrzebnego zakresu. Wdrożenie REST klienta JavaScript nie jest potrzebne, gdy frontend renderuje PHP. Biblioteka komponentów nie jest konieczna dla dwóch suwaków. Osobny panel ustawień nie pomaga użytkownikowi wstawiającemu jedną listę na stronie. Wybory techniczne mają wspierać zachowanie, nie mnożyć elementy ćwiczenia. Jeżeli plan wymaga migracji istniejących treści, powinien dokładnie wyjaśnić przyczynę oraz drogę powrotu.

Ćwiczenie: dopisz do każdego kroku planu jedno kryterium „gotowe, gdy”. Dla modelu będzie to poprawna rejestracja nazw oraz pól. Dla frontendu: brak szkiców i czytelna paginacja. Dla dokumentacji: osoba z pustą instalacją potrafi utworzyć trzy wpisy i osadzić blok. Ten ostatni punkt często ujawnia brak pliku, niejasną ścieżkę lub nieprawdziwe założenie o instalacji.

Plan małego projektu

01 Dane

CPT, meta i kontrola praw

02 Edycja

Meta box oraz ograniczone atrybuty

03 Render

Publish, HTML i CSS

04 Weryfikacja

Node teraz; WordPress na stagingu

05 Przekazanie

README, status testów i odbiór

Od planu do implementacji

Po akceptacji briefu agent powinien przejść do pracy bez ponownego pytania o każdą rutynową decyzję. Sprawdza bieżący stan plików, wdraża plan i przedstawia rzeczywisty diff. Zmiana zakresu wymaga wyjaśnienia, ale nie zatrzymujemy pracy przy każdym detalu.

Zadanie implementacyjne powinno zachować wcześniejszy kontrakt i wskazać dozwoloną swobodę. Agent może dobrać nazwy lokalnych funkcji, uporządkować pliki i napisać test pomocnika. Nie może przy okazji zmienić REST base, rozszerzyć publikowania ani dodać mechanizmu wymagającego sekretu w przeglądarce. Sprawdź jego wynik względem przyjętych kryteriów, a nie wyłącznie względem listy wygenerowanych plików. Kompletność oznacza działający przepływ oraz instrukcję jego odtworzenia.

Przy dłuższej pracy proś o krótkie aktualizacje pokazujące ustalenia i najbliższy krok. „Dodałem pięć plików” mówi mniej niż „model zawiera oba pola; teraz sprawdzam, czy render ogranicza status i atrybuty”. Aktualizacja nie jest dowodem testu, ale pozwala wcześniej zauważyć odejście od zakresu. Jeśli agent odkryje brak narzędzia, powinien kontynuować dostępne kontrole i jasno oznaczyć pozostałe.

Po implementacji wymagaj raportu o zachowaniu, sposobie weryfikacji i ograniczeniach. Raport nie powinien twierdzić, że WordPress został uruchomiony, jeśli wykonano jedynie test JavaScript. Nie akceptuj sformułowania „wszystko przetestowane” bez listy testów. Zwięzły raport może zawierać kilka pozycji, ale każda musi odpowiadać konkretnej operacji i wynikowi.

PROMPT DO CODEXA / Prompt 05 / implementacja

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Wykonaj przyjęty plan w folderze wp-demo-projects oraz tests. Dostarcz wszystkie potrzebne pliki, bez npm build. Zachowaj kontrakt nazw. Zadbaj o custom-fields, nonce, edit_post dla konkretnego rekordu, publish only i escape. Uruchom dostępne testy Node. Przy braku PHP przygotuj tests/integration.php i instrukcję, oznacz je jako nieuruchomione. Zakończ diffem zakresu, wynikiem komend oraz opisem ograniczeń.

KONTROLA: Oceń kompletność plików, deklarację zależności skryptu i zgodność raportu z rzeczywistym środowiskiem.

CPT należy do wtyczki

Realizacje są funkcją domenową, dlatego rejestrujemy je we wtyczce. Zmiana motywu nie powinna usuwać dostępu do modelu danych. Typ `wdl_project` wspiera standardowe pola WordPressa oraz meta udostępniane przez REST.

Funkcja `wdl_projects_init` działa na hooku `init`. W tym miejscu powstaje typ wpisu, meta oraz rejestracja bloku. Ustalona nazwa `wdl_project` pozostaje krótka i jednoznaczna. Publiczny typ pozwala odwiedzić szczegóły opublikowanej realizacji. `has_archive=false` oznacza, że nie tworzymy osobnego archiwum typu: listę redaktor osadza na zwykłej stronie Realizacje. To zapobiega przypadkowemu zdublowaniu dwóch głównych widoków portfolio.

Wsparcie `title`, `editor` i `thumbnail` daje standardową obsługę treści. `revisions` obejmuje standardowe rewizje wpisu; nie obiecujemy w tym przykładzie wersjonowania własnych meta, którego nie konfigurowaliśmy osobno. `custom-fields` jest istotne dla zarejestrowanych pól meta używanych w REST. Nie traktuj go jako zbędnego dodatku tylko dlatego, że formularz pokazujemy w osobnym meta box. Zmiana listy `supports` może zerwać funkcję w miejscu odległym od widocznego panelu.

`capability_type=post` i `map_meta_cap=true` wykorzystują standardowy model uprawnień wpisów. Wtyczka nie dodaje własnej roli redaktora realizacji. Jeśli projekt wymaga oddzielnych praw publikowania portfolio, trzeba zaplanować zmianę modelu `capabilities` oraz testy migracji ról. Nie dopisuj ich przypadkowo do ćwiczenia. Po aktywacji odświeżamy reguły odnośników raz, zamiast wykonywać kosztowny flush podczas każdego żądania.

KOD W PRAKTYCE / PHP / Fragment: typ i podstawowy kontrakt

```
'name' => __( 'Realizacje', 'wp-demo-projects' ),
'singular_name' => __(
    'Realizacja', 'wp-demo-projects' ),
'add_new_item' => __(
    'Dodaj realizację', 'wp-demo-projects' ),
),
'public' => true,
'show_in_rest' => true,
'rest_base' => 'wdl-projects',
'has_archive' => false,
'rewrite' => array( 'slug' => 'realizacja' ),
'menu_icon' => 'dashicons-portfolio',
'supports' => array(
    'title', 'editor', 'thumbnail',
    'revisions', 'custom-fields',
),
'capability_type' => 'post',
'map_meta_cap' => true,
) );
register_post_meta( 'wdl_project', '_wdl_client', array(
    'type' => 'string',
    'single' => true,
    'show_in_rest' => true,
    'sanitize_callback' => 'sanitize_text_field',
```

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

Fragment pełnego pliku w `EXAMPLES`; PHP/WordPress nieuruchomione. Sprawdzenie Node dotyczy wyłącznie składni JavaScript.

Meta: typ, zapis i publiczny odczyt

Pole klienta jest tekstem, a rok liczbą całkowitą. Rejestracja meta opisuje typ oraz zachowanie REST, ale formularz panelu nadal musi kontrolować własny zapis. Zawsze pytaj, skąd przyszło wejście i kto może je zmienić.

`register_post_meta` wiąże oba pola z `wdl_project`. Dla klienta używamy `sanitize_text_field`. Dla roku pomocnik przyjmuje zakres 1900–2100; wartość 0 oznacza brak roku i nie pojawia się w karcie. Schemat REST dopuszcza 0, ponieważ brak zapisanej meta integer ma taką wartość domyślną. Wyjaśniamy tę regułę w README. Gdy biznes wymaga innego znaczenia braku danych, zmień kontrakt świadomie i przetestuj dotychczasowe wpisy.

`auth_callback` sprawdza `edit_post` dla konkretnego `post_id`. Ta decyzja nie zależy od samego tekstu roli użytkownika: model capabilities WordPressa rozstrzyga prawa do wskazanego rekordu. Callback nie jest funkcją ukrywającą meta przed wszystkimi czytelnikami REST. Opublikowane dane klienta i roku są przewidziane do publicznej prezentacji. Nie przechowuj w nich adresu prywatnego, numeru umowy ani danych, których nie wolno pokazać na stronie.

Sanityzacja i walidacja pełnią inne zadania. Formularz roku pozwala zapisać cztery cyfry we właściwym zakresie; niepoprawne wejście nie nadpisuje dotychczasowej wartości. REST ma schemat liczbowy i sanitation. Bezpośredni `update_post_meta` może uruchomić callback sanitation, lecz nie powinien być używany jako zamiennik sprawdzenia praw we własnym endpointcie. W przykładzie nie tworzymy takiego endpointu. Każda nowa ścieżka zapisu musi otrzymać własną kontrolę autoryzacji.

Ćwiczenie: zapisz poprawny rok, potem wyślij 9999 z meta box. Oczekuj zachowania starej wartości. Następnie usuń rok pustym polem i sprawdź brak zera w HTML. Testy pokazują, jak model reaguje na celowe błędy, zamiast ograniczać się do jednego poprawnego wpisu. Przy dodawaniu nowych pól opisz równie jasno ich typ, brak wartości i publiczny charakter.

Nonce oraz capability są niezależne

Meta box wysyła nonce, a funkcja zapisu sprawdza uprawnienia do konkretnego wpisu. Potrzebujemy obu warunków. Prawidłowy nonce nie daje subskrybentowi prawa do edycji realizacji, a capability nie zwalnia z kontroli żądania formularza.

Funkcja zapisu zaczyna się od krótkich warunków zakończenia: właściwy typ wpisu, brak autosave i rewizji, capability edit_post oraz obecność nonce o prawidłowym typie. Potem weryfikuje nonce akcji wdl_projects_save. Dopiero po tych kontrolach czyta pola formularza. To ułatwia przegląd: można zobaczyć granicę autoryzacji przed operacją zmieniającą dane. Warunki nie powinny być rozproszone między kilkoma niepowiązanymi funkcjami bez jasnego powodu.

Dane z \$_POST mogą być tablicą, mimo że formularz normalnie wysyła tekst. Dla klienta po sanitation stosujemy wp_slash przed update_post_meta: ta funkcja ponownie usuwa slashes. W przeciwnym razie literalny backslash mógłby zniknąć. Dlatego sprawdzamy is_string przed wp_unslash i sanitation. Nie opieraj bezpieczeństwa na type=number w HTML: można ominąć formularz. Do wartości wypisywanych w input używamy esc_attr, natomiast tekst klienta na karcie wymaga esc_html. Każdy kontekst wyjścia ma własną funkcję. Ochrona zapisu i ochrona wyświetlania rozwiązują różne problemy.

Procedura integracyjna obejmuje fałszywy nonce, prawidłowy zapis administratora oraz zapis subskrybenta z jego poprawnym nonce. Ten ostatni przypadek dowodzi, dlaczego oba mechanizmy są potrzebne. W pakiecie procedura jest przygotowana do WordPressa, lecz nieuruchomiona. Nie uznaj jej samych asercji za wynik. Zapisz dokładną komendę, środowisko i rezultat dopiero po rzeczywistym wykonaniu.

PROMPT DO CODEXA / Prompt 06 / przegląd bezpieczeństwa

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Przejrzyj wp-demo-projects.php i render.php w granicach tego przykładu. Skup się na nonce, capability konkretnego post_id, typach \$_POST, sanitation meta, publish only oraz escape HTML i URL. Dla każdego problemu podaj plik, ścieżkę wejścia, skutek i minimalną naprawę. Nie zgłaszaj hipotetycznych problemów bez powiązania z kodem. Oddziel analizę od testów WordPress, których nie możesz uruchomić.

KONTROLA: Sprawdź, czy raport nie myli nonce z uprawnieniem i czy uwzględnia publiczne meta.

block.json spina obie strony

Plik `block.json` jest punktem wspólnego opisu bloku. Zawiera nazwę, wersję API, atrybuty oraz powiązanie skryptu, stylu i renderowania. Dynamiczny blok zapisuje konfigurację, a publiczny HTML powstaje na serwerze.

API version 3 stanowi część wymagań tego przykładu. Nazwa `wp-demo-lab/projects` musi być identyczna po stronie PHP i `registerBlockType` w JavaScript. `limit` oraz `columns` mają typ `integer` i domyślne wartości 6 oraz 3. `supports.html=false` ogranicza możliwość edytowania własnego HTML instancji w interfejsie. Nie zastępuje to walidacji serwera; treść wpisu można zmienić inną ścieżką. `render.php` ponownie ogranicza wartości wejściowe.

`editorScript` wskazuje zarejestrowany uchwyt `wp-demo-projects-editor`, a nie plik wymagający budowania. PHP rejestruje ten skrypt wraz z zależnościami. `style` odwołuje się do lokalnego CSS, a `render` do pliku PHP. Ta struktura pozwala uruchomić ćwiczenie przez skopiowanie kompletnego folderu. Nie musisz pobierać pakietów npm ani kompilować JSX. Wszystkie zmiany w przykładzie powinny zachować tę zasadę, chyba że nowy brief jawnie wprowadzi pipeline budowania.

Dynamiczność oznacza, że po zmianie opublikowanych rekordów lista nie wymaga ponownego zapisania strony. `save` zwraca `null`, więc frontend nie otrzymuje zakodowanej listy kart zapisanej w treści. W edytorze pokazujemy jasno opisaną kartę ustawień. Nie udajemy podglądu danych, których nie pobieramy. Ostateczny wygląd i paginację należy sprawdzić na podglądzie strony. To prosty kompromis odpowiedni dla samodzielnego ćwiczenia.

KOD W PRAKTYCE / JSON / Fragment metadanych bloku

```
{
  "$schema": "https://schemas.wp.org/trunk/block.json",
  "apiVersion": 3,
  "name": "wp-demo-lab/projects",
  "version": "1.0.0",
  "title": "Realizacje WP DEMO LAB",
  "category": "widgets",
  "icon": "portfolio",
  "description": "Opublikowane realizacje z paginacją.",
  "textdomain": "wp-demo-projects",
  "attributes": {
    "limit": {
      "type": "integer",
      "default": 6
    },
    "columns": {
      "type": "integer",
      "default": 3
    }
  },
  "supports": {
    "html": false
  }
}
```

PLIK: `wp-demo-projects/blocks/projects/block.json`

Fragment pełnego pliku w EXAMPLES; PHP/WordPress nieuruchomione. Sprawdzenie Node dotyczy wyłącznie składni JavaScript.

JavaScript bez npm build

Skrypt edytora korzysta z globalnego obiektu `wp` dostarczanego przez WordPress. Elementy tworzymy przez `createElement`, a ustawienia pokazujemy w `InspectorControls`. W PHP deklarujemy zależności, dzięki czemu kolejność ładowania nie jest przypadkowa.

W projekcie używamy `wp.blocks`, `wp.element`, `wp.blockEditor`, `wp.components` oraz `wp.i18n`. Odpowiadają im uchwyty skryptów WordPressa wpisane do `wp_register_script`. Dodatkowo edytor zależy od `wp-demo-projects-options`. Jeśli pominiesz zależność, błąd może ujawnić się wyłącznie w konkretnym wariancie ładowania strony. Działająca kontrola `node --check` nie wykryje nieobecnego `window.wp` ani złej kolejności załadowania `assetów` w przeglądarce.

Helper `options.js` zawiera czyste funkcje ograniczania wartości. Używa CommonJS dla testu Node i globalnej nazwy `WDLProjectOptions` w przeglądarce. Nie importuje pakietów WordPressa i nie pobiera sieci. Dzięki temu można sprawdzić zachowanie bez uruchamiania edytora. Nie twórz sztucznego środowiska udającego cały WordPress tylko po to, by otrzymać zielony test. Sprawdzenie małego kontraktu danych daje prawdziwy, choć ograniczony dowód.

Komponent `edit` odczytuje znormalizowane wartości i zapisuje ustawienia przez `setAttributes`. Każdy `RangeControl` ma `min` oraz `max`, ale `onChange` również ogranicza liczbę. `useBlockProps` zapewnia właściwy wrapper edytora. Ten wrapper należy odróżnić od `get_block_wrapper_attributes` używanego w PHP. Obie strony uczestniczą w kontrakcie bloku, lecz działają w innych środowiskach i wymagają osobnego odbioru.

Ćwiczenie: w kopii skryptu usuń zależność `options` i zaobserwuj, jaki błąd pojawia się w edytorze. Przywróć ją przed odbiorem. Zapisz różnicę między błędem składni JavaScript, błędem ładowania `assetu` i błędem działania komponentu. Dopiero taka obserwacja pozwala sformułować precyzyjny `bugfix`. Nie dodawaj bundlera jako pierwszej reakcji na pojedynczy brak zależności.

Atrybuty wymagają dwóch granic

Suwak pomaga redaktorowi wybrać prawidłową wartość, ale nie stanowi granicy zaufania serwera. Ten sam kontrakt ograniczamy w edytorze i render.php. Dla limitu jest to 1–12, dla kolumn 1–3.

Pomocnik bounded przyjmuje wyłącznie skończone liczby JavaScript, obcina część ułamkową i ogranicza przedział. Nie akceptuje tekstu „4” jako wartości liczbowej w edytorze. Przy błędnym typie przywraca domyślną liczbę. Takie zachowanie jest jawne i testowalne. PHP dodatkowo obsługuje możliwe wejście z ręcznie zmienionej treści: ogranicza wartości numeryczne i stosuje defaults dla pozostałych. Nie zlecamy przeglądarce ostatecznej decyzji o kosztownym zapytaniu.

Po co ograniczać limit, skoro redaktor zwykle wybierze sześć kart? Bez granicy można zapisać tysiące rekordów w jednej instancji i zwiększyć koszt generowania strony. columns z ogromną wartością zepsułoby wygląd, nawet bez poważniejszej podatności. Atrybuty odpowiadają za konfigurację małego komponentu; nie powinny dawać dowolnego dostępu do argumentów WP_Query. Utrzymuj listę obsługiwanych opcji krótką i przewidywalną.

Test Node obejmuje defaults, krańce przedziałów, typy nieliczbowe, NaN, Infinity, ułamki oraz brak mutacji wejścia. Każdy przypadek odpowiada realnej decyzji helpera. Nie potwierdza, że PHP implementuje identyczne szczegóły, ani że edytor wyświetla suwaki. Serwer i UI wymagają własnych kontroli. Przy dopisywaniu trzeciego atrybutu opisz typ, granice, fallback i sposób oceny po obu stronach.

KOD W PRAKTYCE / JAVASCRIPT / Fragment: czysta normalizacja

```
function bounded(value, min, max, fallback) {
  if (typeof value !== 'number' ||
      !Number.isFinite(value)) { return fallback; }
  return Math.min(max, Math.max(min, Math.trunc(value)));
}
function normalize(attributes) {
  const source = attributes || {};
  return {
    limit: bounded(source.limit, 1, 12, 6),
    columns: bounded(source.columns, 1, 3, 3)
  };
}
return { bounded, normalize };
```

PLIK: wp-demo-projects/blocks/projects/options.js

Fragment pełnego pliku w EXAMPLES; PHP/WordPress nieuruchomione. Sprawdzenie Node dotyczy wyłącznie składni JavaScript.

Zapytanie publiczne tylko po publish

Widok listy jest publiczny także wtedy, gdy ogląda go zalogowany administrator. Dlatego WP_Query otrzymuje jawnie `post_status=publish`. Nie pozwalamy kontekstowi użytkownika rozszerzyć listy o szkice albo prywatne rekordy. Wykluczamy także realizacje chronione hasłem.

`render.php` odczytuje `limit` i `columns`, po czym ogranicza ich wartości. Parametr `wdl_projects_page` określa stronę wyników; jego typ i zakres także kontrolujemy. Następnie wykonujemy WP_Query z właściwym typem, statusem i `posts_per_page`. Warunek `has_password=false` chroni opis wpisu z hasłem, pobierany bezpośrednio z `post_content`. Kolejność `date` oraz `ID` daje przewidywalny wynik także wtedy, gdy dwa wpisy mają ten sam czas. Nowe publikacje mogą zmienić pozycje między żadaniami. Dla demonstracyjnego portfolio taki model wystarcza.

Iterujemy po `$query->posts` bez `the_post` i `setup_postdata`. Dzięki temu nie zmieniamy globalnego wpisu, od którego może zależeć pozostała część szablonu. Tytuł, permalink i obraz odczytujemy dla konkretnego ID. Opis skracamy po usunięciu shortcode i tagów; nie wykonujemy pełnego `the_content` w małej karcie. Każdy zabieg ma cel: karta pozostaje krótkim podsumowaniem, a nie osadzonym wnętrzem całej realizacji.

Ćwiczenie: utwórz szkic o wyrazistym tytule, a następnie oglądaj listę jako administrator i anonimowy odwiedzający. Nie powinien pojawić się w żadnym z tych widoków. Dodaj opublikowaną realizację z hasłem: karta ma ją pomijać. Dodaj klienta A & B i sprawdź, czy znak ampersand jest poprawnie wyświetlony bez interpretowania treści jako HTML. Oddziel test publicznej listy od uprawnionego odczytu szkicu w panelu edycji.

KOD W PRAKTYCE / PHP / Fragment: publiczne zapytanie

```
'has_password' => false,
'posts_per_page' => $limit,
'paged' => $page,
'orderby' => array( 'date' => 'DESC', 'ID' => 'DESC' ),
'ignore_sticky_posts' => true,
) );
$wrapper = get_block_wrapper_attributes( array(
    'class' => 'wdl-projects wdl-projects--cols-' . $columns,
) );
?>
<section <?php echo $wrapper; ?>>
<?php if ( $query->posts ) : ?>
```

PLIK: `wp-demo-projects/blocks/projects/render.php`

Fragment pełnego pliku w EXAMPLES; PHP/WordPress nieuruchomione. Sprawdzenie Node dotyczy wyłącznie składni JavaScript.

HTML i CSS mają własny odbiór

Poprawna rejestracja bloku nie gwarantuje czytelного widoku. Karty muszą działać na małym ekranie, przy długich nazwach i bez obrazu. Dostępność oceniamy w konkretnym kontekście strony, a nie po samym wyglądzie izolowanego fragmentu.

Każda realizacja jest article z nagłówkiem, linkiem, metadanymi i krótkim opisem. Nagłówek karty ma poziom h3, dlatego na stronie nadrzędnej warto umieścić sekcję z h2 nad listą. Jeżeli kompozycja witryny wymaga innej hierarchii, zmień ją świadomie w komponencie i sprawdź czytnikiem ekranu. Numer nagłówka nie jest parametrem dekoracji; opisuje strukturę treści. Paginacja ma osobny nav i zrozumiałą etykietę.

Tytuł i klient otrzymują `esc_html`, link `esc_url`, a wartości formularza `esc_attr`. HTML obrazu generuje standardowa funkcja WordPressa. Wrapper bloku tworzy `get_block_wrapper_attributes`, zamiast ręcznie sklejać atrybuty obsługiwane przez system bloków. Nie używaj jednej funkcji `escape` do wszystkich miejsc. Kontekst wyjścia rozstrzyga, czy potrzebujesz tekstu, wartości atrybutu, adresu czy ograniczonego HTML.

CSS stosuje prefiks `wdl-projects` dla gridu, kart, obrazu i nawigacji. Mobilnie widok ma jedną kolumnę. Od 40rem warianty dwu- i trzykolumnowe pokazują dwie, a od 64rem wariant trzykolumnowy pokazuje trzy. Wariant jednokolumnowy pozostaje jedną kolumną. `minmax(0,1fr)`, `min-width:0` i `overflow-wrap` pomagają przy długich nazwach. Obraz ma proporcję oraz `object-fit`, ale jego semantyczny `alt` nadal wybiera redaktor w bibliotece mediów.

Wygląd dziedziczy kolor i nie wymusza całej palety motywu. Obrys oraz fokus używają `currentColor`, więc konkretny motyw musi zapewnić odpowiedni kontrast. Sprawdź widok z aktywnym motywem blokowym, a później z motywem docelowym. Oglądaj szerokości 320, 768 i 1280 pikseli, powiększenie tekstu oraz klawiaturę. Brak poziomego przewijania jest ważnym, lecz niewystarczającym kryterium dostępności.

Ćwiczenie: usuń obraz z jednej realizacji, wydłuż tytuł drugiej i pozostaw pusty rok trzeciej. Przejdź tabulatorem przez linki kart oraz paginację. Fokus musi być widoczny, a opis celu linku zrozumiały. Zapisz zaobserwowane usterki razem z nazwą motywu i szerokością okna. Agent potrzebuje tych danych, aby naprawić konkretny konflikt CSS zamiast zgadywać nowe style.

Paginacja jest częścią kontraktu

Lista wykorzystuje natywne `paginate_links` i parametr `wdl_projects_page`. To prosty model dla jednej instancji bloku na stronie. Linki powinny zachować kontekst adresu, a nie wymagać JavaScript do przejścia na kolejne wyniki.

WP_Query otrzymuje `paged` i liczy `max_num_pages`. Bez tego nie zbudujesz poprawnej nawigacji po pełnym zbiorze. Linki powstają z bieżącego adresu z zastąpionym parametrem strony. `paginate_links` tworzy elementy listy oraz oznacza bieżący numer; wynik przepuszczamy przez `wp_kses_post`. Nawigacja pojawia się tylko przy więcej niż jednej stronie. Nie ustawiamy `no_found_rows=true`, ponieważ wtedy informacja potrzebna do paginacji byłaby niedostępna.

Przy trzech rekordach i limicie dwa sprawdzisz zarówno pierwszą, jak i drugą stronę. Przejście wstecz powinno wrócić do pierwszego zestawu. Parametr o bardzo dużej wartości prowadzi do pustego stanu, a nie do nieograniczonego limitu rekordów. Serwer ogranicza numer strony do 10000. Nie nazywamy tego zabezpieczeniem przed wszystkimi kosztownymi zapytaniami: duże portfolio może wymagać innych strategii wydajności. Przykład dotyczy małej listy demonstracyjnej.

Dwie instancje bloku współdzieliłyby ten sam parametr. Dlatego README wyraźnie wymaga jednej listy na stronie. Jeśli pojawi się rzeczywista potrzeba kilku niezależnych list, nowy brief powinien określić identyfikator instancji, adresy i zachowanie nawigacji. Nie dodawaj losowego ID podczas renderowania, gdy musi być stabilne pomiędzy żądaniami. Najpierw zaprojektuj kontrakt, potem implementuj rozszerzenie.

Ćwiczenie: utwórz trzy wpisy z `fixtures.json` i ustaw limit dwa. Zapisz tytuły obecne na obu stronach. Następnie dodaj szkielet i potwierdź, że liczba publicznych stron nie wzrosła. Otwórz link drugiej strony w nowej karcie bez zalogowania. Ta kontrola pokazuje, czy nawigacja działa jako zwykły adres HTTP. Nie ograniczaj odbioru do kliknięć wykonywanych wyłącznie w panelu administratora.

Budowa strony wyników

01 Adres

`wdl_projects_page` z kontrolą typu

02 Atrybuty

Limit 1–12, kolumny 1–3

03 Zapytanie

Wyłącznie opublikowane realizacje

04 Karty

Tekst i adresy z `escape`

05 Nawigacja

Natywne linki oraz pusty stan

Czytaj diff jako historię zachowania

Diff pokazuje, co naprawdę zmienił agent. Rozpocznij przegląd od zakresu plików, potem przejdź przez przepływ danych i warunki bezpieczeństwa. Zmiana może wyglądać estetycznie, a jednocześnie usunąć kontrolę statusu albo zależność skryptu.

Porównaj listę zmienionych plików z planem. Nowy plik `render.php` jest oczekiwany, modyfikacja motywu w tym zadaniu wymaga wyjaśnienia. Sprawdź, czy agent nie dodał bundlera, zewnętrznych usług lub konfiguracji zawierającej sekret. Obejrzyj także dokumentację: pominięta instrukcja aktywacji lub nieprawdziwy raport testów jest wadą przekazania projektu, nawet gdy sam kod wydaje się kompletny. Użytkownik potrzebuje możliwości odtworzenia wyniku.

Następnie odczytaj warunki zapisu i argumenty zapytania. Czy `capabilities` nadal dotyczy konkretnego `post_id`? Czy `nonce` jest sprawdzany przed aktualizacją? Czy status pozostaje `publish`? Czy atrybuty mogą wygenerować dowolną klasę lub kosztowny limit? Czy `escape` odpowiada miejscu wyświetlenia? Odpowiedź na te pytania wymaga prześledzenia kilku linii, ale nie wymaga ogólnego przepisywania całego pliku. Przegląd powinien wskazać rzeczywisty skutek zmiany.

`DIFF_EXAMPLE.patch` ilustruje hipotetyczną naprawę: dodanie jawnego `post_status`. Aktualny `render.php` zawiera już ten warunek, więc plik nie jest poprawką do aplikowania. Taka etykieta zapobiega pomyleniu materiału dydaktycznego ze stanem repozytorium. W prawdziwym review pokaż pozycję problemu, jego skutek oraz minimalną zmianę. Nie zgłaszaj ponownie usterki, która nie istnieje w aktualnej wersji.

Ćwiczenie: poproś drugą osobę o wskazanie trzech najważniejszych warunków w diffie bez czytania całej rozmowy. Jeżeli nie potrafi znaleźć kontraktu, opisu zachowania i statusu testów, popraw opis zmiany. Dobry commit oraz dokumentacja powinny być zrozumiałe dla opiekuna, który dołączy do projektu po miesiącu. Agent nie będzie zawsze dostępny, aby tłumaczyć kontekst każdego pliku.

Debug zaczyna się od odtworzenia

Przed naprawą zbierz objaw, warunki i minimalną procedurę odtworzenia. Brak bloku w edytorze, błąd PHP przy aktywacji i niepoprawna druga strona listy to różne problemy. Jedno ogólne „nie działa” nie rozstrzyga, gdzie szukać.

Dla błędu edytora zapisz wersję WordPressa, nazwę motywu, informację o aktywacji wtyczki oraz komunikat konsoli. Sprawdź, czy editor.js i options.js zostały załadowane, zanim zmienisz komponent. Dla problemu publicznego widoku zapisz adres, ustawienia bloku, statusy danych i sposób zalogowania. Dla błędu odnośników sprawdź reguły permalinków oraz to, czy szczegóły realizacji są opublikowane. Najkrótsza poprawka zależy od źródła, nie od intensywności objawu.

Agent powinien sformułować hipotezę i wskazać kontrolę, która ją rozstrzygnie. Jeśli wszystkie karty znikają po ustawieniu strony drugiej, porównaj `posts_per_page`, `paged` i publiczną liczbę rekordów. Jeżeli klient zawiera niepożądany HTML, sprawdź sanitization zapisu i escape wyjścia oddzielnie. Nie stosuj masowej zmiany kilku mechanizmów, zanim potwierdzisz przyczynę. Takie zmiany mogą ukryć objaw i jednocześnie stworzyć nową regresję.

W tym pakiecie nie mamy wyników runtime PHP. W razie zgłoszenia błędu agent może przygotować procedurę oraz przejrzeć kontrakt, lecz nie powinien wynajdywać komunikatu logu. Prawdziwy komunikat trzeba otrzymać z twojego stagingu. Przed udostępnieniem usuń sekrety oraz dane klientów. Zachowaj treść techniczną, która pomaga zlokalizować przyczynę: plik, numer linii, typ wyjątku i etap działania.

PROMPT DO CODEXA / Prompt 07 / bugfix

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Objaw: na stagingu druga strona listy nie pokazuje oczekiwanych rekordów. Dane: trzy publish, limit 2, jedna instancja bloku. Przeczytaj `render.php` i instrukcję paginacji. Najpierw opisz hipotezy oraz minimalne kontrole adresu i `WP_Query`. Nie zmieniaj kodu bez wskazania przyczyny. Po potwierdzeniu napraw tylko właściwe miejsce, dodaj kontrolę regresji i zapisz wykonane testy; nie deklaruj wyników WordPressa bez uruchomienia.

KONTROLA: Oczekuj rozdzielienia hipotezy od obserwacji oraz minimalnego zakresu naprawy.

Review ma kończyć się decyzją

Przegląd kodu powinien prowadzić do jasnej decyzji: można odebrać, trzeba naprawić konkretny problem albo brakuje konkretnego dowodu. Długa lista luźnych sugestii utrudnia odróżnienie blokady od ulepszenia na przyszłość.

Oddziel problemy uniemożliwiające odbiór od pomysłów rozwojowych. Ujawnienie szkicu na publicznej liście jest blokadą. Dodanie filtrowania po branży jest nową funkcją, jeśli nie wynikało z briefu. Brak wykonania testu PHP jest ograniczeniem dowodów; należy je uzupełnić przed wdrożeniem, a nie przedstawiać jako potwierdzoną wadę składni. Takie kategorie pomagają utrzymać pracę w ustalonym zakresie i ocenić, co agent naprawdę powinien poprawić.

Opis problemu powinien zawierać warunek wejściowy i rezultat. „Brakuje walidacji” jest mniej użyteczne niż „columns=99 trafia do nazwy klasy bez ograniczenia”. W obecnym przykładzie serwer tę wartość ogranicza. Weryfikuj aktualny kod, zamiast powtarzać ostrzeżenie z wcześniejszej wersji. Gdy sugestia dotyczy konwencji zespołu, wskaż źródło reguły. Nie przedstawiaj własnej preferencji formatowania jako podatności bezpieczeństwa.

Po poprawce porównaj nowy diff ze zgłoszonym problemem. Uruchom test, który rozstrzygał regresję, oraz kontrole odpowiednie dla dotkniętej warstwy. Nie powtarzaj szerokiego audytu bez powodu, ale także nie kończ po samym usunięciu komunikatu. Jeśli naprawa zmienia kontrakt, uaktualnij README, testy i kryteria odbioru. Review ocenia wynik, nie samą liczbę iteracji.

PROMPT DO CODEXA / Prompt 08 / przegląd diffu

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Przejrzyj aktualny diff względem briefu wp-demo-projects. Sprawdź zakres plików, nazwy CPT/REST/bloku, granice zapisu i publish-only oraz zależności skryptów. Zgłoś tylko problemy z konkretnym skutkiem. Każdy opis: warunek, plik, zachowanie i minimalna naprawa. Oddziel blokady, brakujące testy środowiskowe i sugestie poza zakresem. Potwierdź, czy README umożliwia samodzielny start.

KONTROLA: Nie akceptuj ogólnych ostrzeżeń bez ścieżki problemu ani fikcyjnych wyników runtime.

Test odpowiada na jedno pytanie

Test jest użyteczny, gdy potrafisz powiedzieć, jaką decyzję potwierdza. Kontrola składni, test pomocnika i test integracyjny dostarczają różnych dowodów. Nie ustawiaj ich w jednej kategorii „działa”, bo wtedy łatwo przeoczyć brak runtime aplikacji.

`node --check` sprawdza, czy plik JavaScript da się sparsować. Nie otwiera WordPressa i nie tworzy komponentu. `node --test` uruchamia asercje `options.test.js`: dane domyślne, granice, błędne typy, ułamki oraz brak mutacji. To prawdziwy test czystej funkcji używanej przez edytor. Jego zakres jest celowo mały. Jeżeli ktoś usuwa helper z projektu, powinien także ocenić, czy test nadal sprawdza kod używany w aplikacji.

Procedura `integration.php` wymaga uruchomionego WordPressa i WP-CLI. Sprawdza rejestrację modelu, nonce, capability, sanitation, rok, publiczny render oraz anonimowy REST. Tworzy tymczasowe rekordy i konto, a w finally przywraca użytkownika oraz usuwa dane. To nie jest test uruchamiany samym php bez bootstrapu WordPressa. Instrukcja musi wskazywać prawidłowy katalog instalacji, użytkownika administratora i ścieżkę pliku. Używaj pustego stagingu, nie produkcji.

Ręczny odbiór uzupełnia asercje o zapis w edytorze, linki paginacji, fokus i zachowanie z motywem. Nie wszystkie te obserwacje warto od razu automatyzować. W małym demonstracyjnym komponencie odtwarzalna procedura często daje wystarczający materiał do pierwszego odbioru. Gdy funkcja staje się krytyczna lub jest często zmieniana, zwiększ automatyzację właśnie tych zachowań, które rzeczywiście regredują.

Ćwiczenie: przypisz do każdej komendy pytanie i ograniczenie. Dla Node: czy składnia i helper są poprawne, bez dowodu WordPressa. Dla `integration.php`: czy ścieżki danych działają w konkretnej instalacji, bez dowodu wszystkich motywów. Dla przeglądarki: czy widok jest użyteczny w obserwowanym kontekście. Tak przygotowana mapa testów pozwala uniknąć zarówno nadmiernej pewności, jak i bezcelowego powtarzania kontroli.

Kontrola	Dowód	Granica
<code>node --check</code>	Składnia JS	Nie uruchamia edytora
<code>node --test</code>	Normalizacja atrybutów	Nie uruchamia PHP
<code>integration.php</code>	Kontrakt na stagingu	Wymaga WordPressa
Przeglądarka	Widok i interakcje	Wymaga obserwacji

Przypadki negatywne chronią kontrakt

Poprawny zapis trzech realizacji pokazuje tylko jedną ścieżkę. Dodaj próby, które powinny zostać odrzucone albo bezpiecznie ograniczone. Najważniejsze przypadki wynikają z praw użytkownika, statusu rekordu, typu wejścia i granic konfiguracji.

Dla meta box sprawdź brak nonce, fałszywy nonce i poprawny nonce użytkownika bez `edit_post`. Ten ostatni przypadek często wykrywa pomylenie ochrony żądania z autoryzacją. Dla roku spróbuj niepoprawnego zakresu i pustego pola. Oczekiwanie musi być zapisane przed testem: 9999 nie nadpisuje roku, a pusty input usuwa wartość. Jeśli dopiero po wyniku wybierzesz „właściwe” zachowanie, test nie ocenia wcześniejszej umowy.

Dla publicznej listy utwórz draft i private z unikalnymi tytułami. Oglądaj listę także jako administrator, ponieważ jego prawa nie powinny rozszerzać publicznego zapytania. Dla anonimowego REST użyj listy ID obejmującej rekord opublikowany i dwa niepubliczne. Oczekuj tylko opublikowanego. Przy testowaniu treści wybierz znaki wymagające escape, na przykład ampersand w nazwie klienta. Nie potrzeba agresywnego testowania obcej witryny, aby sprawdzić własny kontrakt wyjścia.

Dla helpera zdefiniowaliśmy NaN, Infinity, tekst, null i obiekt. W JavaScript typ number obejmuje więcej wartości niż zwykłe liczby całkowite, dlatego `Number.isFinite` ma znaczenie. Test nie musi kopiować implementacji; powinien wymieniać oczekiwane zachowanie użytkowe dla konkretnych wejść. Nie testuj liczby linii pliku jako dowodu bezpieczeństwa. Taka kontrola może pilnować formatu wydawniczego, ale nie wykrywa błędów uprawnień.

Ćwiczenie: wybierz jeden test negatywny, usuń w kopii odpowiedni warunek ochronny i potwierdź, że test wykrywa regresję. Następnie przywróć kod. Wykonuj to wyłącznie na własnym środowisku demonstracyjnym. Dla nieuruchomionego PHP przygotuj tę procedurę do stagingu, bez deklarowania wyniku. Różnica między napisanym testem a wykonaną kontrolą pozostaje ważna także wtedy, gdy asercje wyglądają wiarygodnie.

Raport testów bez dopowiadania

Raport ma umożliwić sprawdzenie wyniku przez inną osobę. Zapisz komendę, datę, wersję narzędzia, status oraz zakres dowodu. Jeżeli test nie został uruchomiony, wpisz to wprost i wyjaśnij, czego potrzebuje do wykonania.

TEST_STATUS.md jest częścią przekazania projektu. W tym pakiecie oddziela wykonane kontrole Node oraz JSON od nieuruchomionych PHP, aktywacji i integracji WordPress. Nie wpisujemy wyniku PASS dla procedury, którą tylko napisaliśmy. Nie mówimy też o pełnym E2E na podstawie ręcznego przejrzania plików. Użytkownik powinien móc skopiować komendy i uzupełnić własny dziennik po uruchomieniu stagingu.

Raport sukcesu testu powinien wskazać ograniczenia, gdy mają wpływ na decyzję o wdrożeniu. „Cztery testy helpera przeszły; integracja WordPress pozostaje do wykonania” jest krótkie i precyzyjne. Nie potrzebuje długiego katalogu hipotetycznych zagrożeń. Jeżeli test zawiódł, zapisz komunikat i przyczynę rozpoznaną albo jeszcze nieznaną. Nie poprawiaj samego opisu wyniku, aby ukryć problem. Zmiana kodu i ponowne wykonanie kontroli muszą być widoczne w historii pracy.

Dziennik stagingu warto uzupełnić o motyw, konfigurację permalinków, rolę użytkownika i zestaw danych. Te informacje często tłumaczą różnicę między dwoma instalacjami. Zachowaj jedynie dane techniczne potrzebne do odtworzenia; nie wklejaj haseł, tokenów ani pełnej konfiguracji produkcyjnej. Dowód nie powinien tworzyć nowego problemu z dostępem do sekretów.

PROMPT DO CODEXA / Prompt 09 / testy i raport

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Uruchom `node --check` dla `editor.js` i `options.js` oraz `node --test tests/options.test.js`. Sprawdź poprawność JSON. Zapisz komendy, wersję, datę i wyniki w `TEST_STATUS.md`. Jeśli nie masz PHP i WordPressa, oznacz ich kontrole jako nieuruchomione i wskaż procedurę README. Nie zastępuj integracji testem tekstowym i nie deklaruje E2E. Zakończ listą dowodów potrzebnych przed wdrożeniem.

KONTROLA: Porównaj raport z rzeczywistymi wynikami komend i upewnij się, że nie zawiera fikcyjnej aktywacji.

Samodzielny start i odbiór

Odbiór sprawdza, czy osoba nieuczestnicząca w implementacji potrafi odtworzyć funkcję. Dlatego README zaczyna się od minimalnej instalacji WordPressa i motywu blokowego. Poprzedni tom może wzbogacić wygląd, lecz nie jest wymagany do uruchomienia wtyczki.

Skopiuj właściwy folder do wp-content/plugins i aktywuj jedną kopię. Następnie utwórz trzy realizacje według fixtures.json. Plik ten nie jest eksportem importera WordPress: służy jako jawne źródło danych do ręcznego ćwiczenia. Tytuł i treść przepisujesz do standardowego edytora, klienta i rok do sekcji Dane realizacji. Obrazy są opcjonalne. Dzięki temu start nie zależy od dodatkowego importera, nieudokumentowanego narzędzia ani dostępu do zewnętrznej usługi.

Utwórz zwykłą stronę Realizacje ze slugiem realizacje i wstaw jedną instancję bloku. Ustaw limit dwa, aby zobaczyć drugą stronę listy. Edytor opisuje ustawienia i odsyła do podglądu; nie prezentuje fikcyjnej listy wyników. Otwórz publiczny widok bez logowania, przejdź do drugiej strony, odwiedź szczegóły rekordu i przetestuj pusty stan. Potem dodaj szkic i potwierdź, że lista go nie ujawnia.

Odbiór redaktora obejmuje poprawny zapis pól oraz zachowanie pustego roku. Odbiór odwiedzającego obejmuje czytelne karty, linki i nawigację. Odbiór programisty obejmuje kontrakt danych, uprawnienia, zakres diffu i wyniki testów. Każda perspektywa ma własne kryteria. Nie wystarczy, że administrator potrafi zobaczyć rekord w panelu, gdy anonimowy użytkownik dostaje błędną stronę. Nie wystarczy też atrakcyjny frontend, jeśli subskrybent może zmieniać cudze dane.

Ćwiczenie: przekaż README drugiej osobie i poproś o start na pustym stagingu. Zanotuj miejsca, w których potrzebowała dodatkowej informacji. Popraw instrukcję tak, aby nazwy, ścieżki i oczekiwane wyniki były jednoznaczne. To test dokumentacji i kompletności pakietu. Wykonaj go przed uznaniem materiału za gotowy do przekazania, a nie dopiero wtedy, gdy odbiorca zgłosi brakujący plik.

Deploy jest osobną decyzją

Zakończenie implementacji nie oznacza automatycznej publikacji na produkcji. Przed wdrożeniem przygotuj gotowy do oceny zakres, wyniki stagingu i możliwość powrotu. Człowiek zatwierdza konkretną zmianę, której skutki zostały już opisane.

Przygotuj kopię plików oraz bazy i sprawdź procedurę odtworzenia. Samo istnienie archiwum nie dowodzi, że potrafisz przywrócić działającą stronę. Wtyczka tego tomu nie wykonuje własnych migracji ani nie usuwa danych przy dezaktywacji. Mimo to publikacja nowych rekordów i zmiana permalinków wpływają na bazę, więc rollback kodu nie cofa wszystkich działań redakcyjnych. Ustal, które czynności są częścią wdrożenia i kto za nie odpowiada.

Na stagingu wykonaj lint PHP, procedurę integracyjną oraz odbiór przeglądarki. Porównaj docelową wersję z diffem i upewnij się, że nie dołączono plików testowych do nieodpowiedniego miejsca publicznego. Skopiuj jedynie wtyczkę do katalogu plugins; materiały dokumentacyjne i testy mogą pozostać poza webroot. Wprowadzenie nowego systemu deploymentu nie należy do tego ćwiczenia. Użyj istniejącej, zatwierdzonej procedury organizacji.

Po wdrożeniu wykonaj krótki smoke test: strona otwiera się anonimowo, lista zawiera tylko publish, paginacja działa, zapis pól nadal jest możliwy dla uprawnionego użytkownika. Przy problemie najpierw ogranicz wpływ i przywróć znaną wersję zgodnie z planem. Nie próbuj kolejnych nieprzetestowanych napraw bez końca na działającej stronie. Zapisz obserwację i odtwórz ją na stagingu, aby kolejna poprawka miała dowód.

PROMPT DO CODEXA / Prompt 10 / zakres i wdrożenie

Wklej po udostępnieniu wskazanych plików. Dostosuj ścieżki do własnego repozytorium.

Przygotuj reviewowalny pakiet wdrożenia wp-demo-projects. Podaj finalny zakres plików, wynik testów lokalnych, brakujące kontrole stagingu, kroki aktywacji oraz rollback kodu i ograniczenia dotyczące bazy. Nie wdrażaj ani nie publikuj. Wskaż, jakie konkretne dowody musi otrzymać człowiek przed akceptacją. Nie rozszerzaj zadania o inne wtyczki lub motyw.

KONTROLA: Plan powinien oddzielać implementację, staging, decyzję człowieka i smoke test.

Co dalej: rozwijaj jedną umowę

Po odbiorze masz mały, czytelny komponent: model danych, dwa pola, dynamiczny blok i zestaw procedur. Następne zadanie powinno rozwijać konkretną potrzebę użytkownika, zachowując istniejący kontrakt lub świadomie opisując jego zmianę.

Możesz rozważyć filtr po kategorii, niezależne listy na jednej stronie albo rzeczywisty podgląd danych w edytorze. Każda opcja zwiększa zakres: filtr wymaga nowego modelu i parametrów, kilka list wymaga stabilnej identyfikacji paginacji, a podgląd edytora wymaga obsługi requestu oraz stanów błędu. Nie dodawaj ich wyłącznie dlatego, że agent potrafi szybko wygenerować kod. Najpierw zapisz problem użytkownika i warunek akceptacji nowego zachowania.

Przy pracy z większą witryną korzystaj z tego samego rytmu: orientacja, brief, plan, implementacja, diff i dowód. Instrukcje AGENTS.md porządkują stałe reguły, natomiast prompt opisuje bieżące zadanie. Pamiętaj, że środowisko może zmienić się między sesjami. Sprawdź dostępne narzędzia oraz status repozytorium zamiast zakładać, że poprzedni wynik nadal obowiązuje. Dobre raportowanie umożliwia kontynuację przez innego programistę lub inną sesję agenta.

Kolejny tom serii rozszerzy projekt o kontrolowany przepływ treści. To nowa funkcja z innym kontraktem uprawnień; wtyczka realizacji nie powinna przypadkowo przejąć jej odpowiedzialności. Utrzymuj funkcje domenowe osobno od motywu i od adapterów usług. Publiczne portfolio pozostaje prostym odczytem opublikowanych treści. Taka granica pomaga rozumieć system, testować poszczególne części i ograniczać skutki zmian.

Ćwiczenie końcowe: napisz krótką notatkę przekazania dla przyszłego opiekuna. Wymień zachowanie, nazwy kontraktu, dozwolone dane, wykonane testy oraz te pozostające do wykonania. Dodaj jedną uzasadnioną propozycję rozwoju i jej kryterium odbioru. Jeżeli nie możesz wskazać użytkownika, któremu pomoże, odłóż ją. Najcenniejszym rezultatem tego tomu jest zdolność zamiany pracy agenta w małą, zrozumiałą i sprawdzalną zmianę aplikacji.

Źródła i dalsza weryfikacja

Źródła oficjalne sprawdzono podczas przygotowania tomu, 13 października 2026. Dokumentacja objaśnia API i zasady narzędzi; nie zastępuje wyników testów konkretnej instalacji. Przed wdrożeniem sprawdź zgodność z używanymi wersjami oraz uzupełnij dziennik stagingu.

`update_post_meta()`

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

`paginate_links()`

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

Custom instructions with AGENTS.md

<https://learn.chatgpt.com/docs/agent-configuration/agents-md>

Agent approvals & security

<https://learn.chatgpt.com/docs/agent-approvals-security>

Codex Security

<https://learn.chatgpt.com/docs/security>

`register_post_type()`

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

`register_post_meta()`

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

Metadata in block.json

<https://developer.wordpress.org/block-editor/reference-guides/block-api/block-metadata/>

Nonces

<https://developer.wordpress.org/plugins/security/nonces/>

`WP_Query`

https://developer.wordpress.org/reference/classes/wp_query/

`get_block_wrapper_attributes()`

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

Character Escaping: zapis stringa i zachowanie backslasha. Natywna nawigacja po stronach wyników `WP_Query`. Instrukcje projektu, wykrywanie plików i reguły review. Uprawnienia środowiska, zatwierdzanie i ograniczenia dostępu. Oficjalny punkt wejścia; nie jest raportem audytu przykładu. Rejestracja typu, supports, REST i capabilities. Typ, sanitation i autoryzacja pól meta. API version, atrybuty oraz assety i render. Nonce chroni żądanie, nie nadaje uprawnień. Status, limit, paged i liczba stron wyników. Wrapper dynamicznego bloku i obsługiwane atrybuty.

Po otwarciu źródła sprawdź sekcję dotyczącą użytego argumentu albo hooka. Jeżeli dokumentacja opisuje nowszą funkcję, porównaj ją z minimalną wersją z README. Nie rozszerzaj przykładu bez testu kompatybilności. Dokumentacja WordPressa jest podstawą kontraktu CPT, meta oraz dynamicznego bloku. OpenAI Docs opisują instrukcje i uprawnienia pracy agenta. Przykłady zastosowania, ćwiczenia i decyzje projektowe w tym tomie są opracowaniem LAB PERCEPCJI dla fikcyjnego WP DEMO LAB. Nie przedstawiają opinii dostawcy o gotowej wtyczce.

Katalog EXAMPLES jest właściwym punktem startu dla kodu. README wyjaśnia aktywację i utworzenie danych; AGENTS.md określa zakres; TEST_STATUS.md rozróżnia wykonane kontrole od procedur do uruchomienia. Zachowaj te pliki razem z wtyczką w swoim repozytorium. Przy rozbudowie zapisz także nowe źródła, datę weryfikacji oraz zmianę kryteriów odbioru.