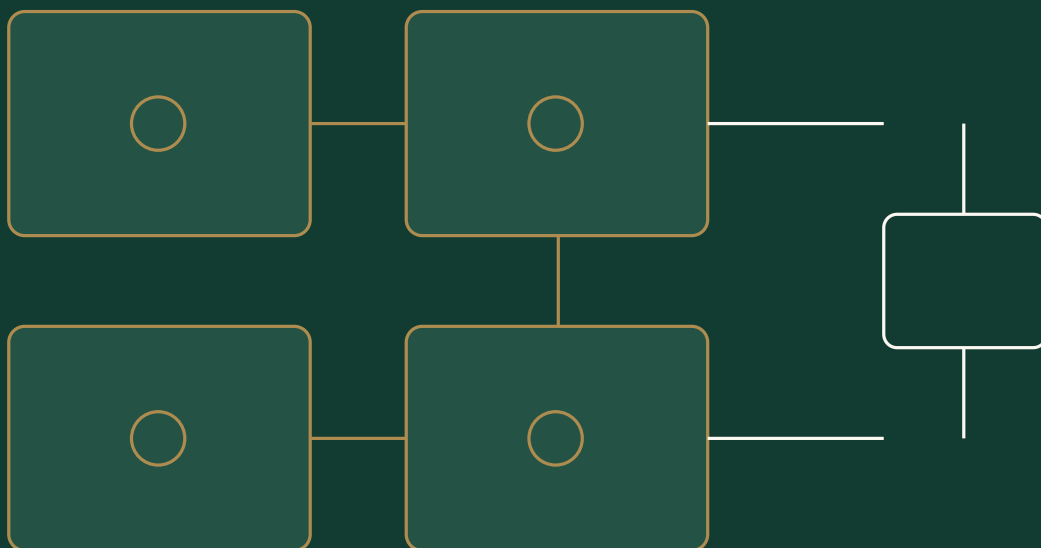


WŁASNE WTYCZKI WORDPRESS Z AI

Projektowanie i programowanie własnych funkcji



PROJEKT TWORZY CZŁOWIEK.
AI WSPIERA PROCES.

10 ROZDZIAŁÓW / JEDEN SYSTEM DEMONSTRACYJNY

Spis treści

Wprowadzenie i mapa projektu	3
01 Od potrzeby do kontraktu wtyczki	4
Najpierw opis zachowanie · Granice między trzema warstwami · Słownik, którego nie zmieniamy przypadkiem	
02 Struktura PHP i cykl WordPressa	7
Bootstrap i zamknięty autoloader · Actions i filters w odpowiednim momencie · Małe moduły, czytelne zależności	
03 Model treści: typ, taksonomia, standardowe pola	10
Typ treści z prawami własnej domeny · Temat jest klasyfikacją, okładka jest obrazem · Tworzymy dane, które obnażają błędy	
04 Metadane i interfejs redaktora	13
Rejestrujemy meta, opisujemy zakres · Od wyboru media do zapisu ID · Interfejs mówi, co naprawdę się wydarzyło	
05 Role, capability i granice dostępu	16
Nonce i capability sprawdzają różne rzeczy · Nadajemy najmniejszy potrzebny zakres praw · Testujemy odmowę, nie tylko wygodny sukces	
06 Bezpieczny zapis danych	19
Walidacja przed normalizacją · Guard clauses w zapisie materiału · Zabezpieczamy wyjście oraz komunikaty	
07 REST i publiczny katalog	22
Wykorzystujemy REST rdzenia do zapisu · Publiczny odczyt ma własny kontrakt · Przegląd API przez konkretne żądania	
08 Zapytania, wydajność i odporność	25
Zapytanie z najmniejszą potrzebną swobodą · Prepared SQL, gdy rzeczywiście jest potrzebne · Odpowiedź odporna na brak pliku	
09 Testy, wersjonowanie i utrzymanie	28
Test ma dowieść konkretnego zachowania · Macierz łączy sukces, odmowę i trwałość · Wersjonowanie i polityka usuwania	
10 Odbiór projektu i dalsza droga	31
Odbiór zaczyna się od dowodów · Codex przygotowuje przekazanie projektu · Następny krok: wiedza z publicznej biblioteki	
Źródła techniczne i dalsza lektura	34

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

PRACOWNIA / PRO 04

Własna funkcja, własna odpowiedzialność

Ten tom przeprowadza od konkretnej potrzeby biblioteki materiałów do kompletnej wtyczki, której zachowanie można przeczytać, sprawdzić i przekazać do odbioru.

WP DEMO LAB potrzebuje miejsca na publiczne poradniki i listy kontrolne. Materiał ma tytuł, opis, temat, okładkę, plik oraz numer wersji. Redaktor przygotowuje szkic, a uprawniony człowiek zatwierdza publikację. Zamiast wiązać tę funkcję z wyglądem strony, budujemy oddzielną wtyczkę wp-demo-library. Zmiana motywu nie powinna usuwać modelu materiałów ani praw do ich edycji.

Pracujemy wyłącznie na danych fikcyjnych. Przykłady Atelier Północ i Studio Fala nie opisują prawdziwych klientów. Biblioteka tej wersji jest publiczna także na poziomie plików. Szkic chroni rekord przed standardowym odczytem, ale nie zabezpiecza bezpośredniego adresu uploads. Prywatnego PDF nie przesyłamy do takiego magazynu. Tom 05 rozwija architekturę z osobnym przechowywaniem i kontrolą dostępu.

Czytaj stronę jako wyjaśnienie jednej decyzji. Pełne pliki znajdują się w EXAMPLES; krótkie fragmenty w książce nie zastępują kompletnej wtyczki. Prompty pomagają zlecać pracę Codexowi, ale odbiór wymaga porównania kodu z kontraktem i wykonania testów w WordPressie. Nie ma tutaj obietnicy, że wygenerowany kod jest gotowy dla dowolnego hostingu.

Założenia ćwiczenia to WordPress obsługujący wymagane API, PHP 8.2 lub nowszy i Node 20 lub nowszy. Do stagingu wybierz aktualne wspierane wydania, a konkretne wersje zapisz w dzienniku. W środowisku przygotowania sprawdzono składnię JavaScript oraz dane JSON; nie uruchomiono PHP, Dockera ani WordPressa. Dostarczony smoke jest instrukcją odtwarzalnego sprawdzenia, nie raportem z jego wykonania.

Najpierw opisz zachowanie

Dobry kontrakt wtyczki wskazuje użytkownika, dozwolone działanie, źródło danych i oczekiwany błąd. To daje Codexowi zadanie możliwe do sprawdzenia.

Punktem startu jest zdanie: administrator publikuje zatwierdzony materiał demonstracyjny i może przypisać mu publiczny plik. Nie zaczynamy od rozbudowanej listy funkcji. Wybieramy mały obszar odpowiedzialności, który potrafimy przetestować od dodania szkicu aż po odczyt anonimowy. Wersja dokumentu opisuje zmianę konkretnego materiału, a wersja wtyczki opisuje wydanie programu; tych numerów nie utożsamiamy.

Kontrakt zapisuje nazwy przed wygenerowaniem plików: slug wp-demo-library, typ wdl_material, baza REST wdl-materials oraz taksonomia wdl_topic. Tytuł i opis pozostają w standardowych polach WordPressa, a okładka jest obrazem wyróżniającym. Dwa pola dodatkowe mają typ integer. Nie dokładamy drugiego pola tytułu ani wpisywanego ręcznie adresu pobierania.

Zakres pierwszego wydania obejmuje panel, zapis, publiczny odczyt i bezpieczne zachowanie przy wyłączeniu. Nie obejmuje automatycznej publikacji, prywatnego magazynu ani pobierania dowolnego URL. Zapisz także zachowanie negatywne: użytkownik bez odpowiednich praw nie edytuje materiałów, błędna wersja nie zastępuje poprzedniej, a nieistniejący attachment nie staje się linkiem.

W praktyce tę kartę kontraktu dołączamy do każdej zmiany. Jeżeli Codex proponuje dodatkowy endpoint lub pole, pytamy, który warunek odbioru uzasadnia jego obecność. Dzięki temu łatwiej rozpoznać rozszerzenie zakresu i utrzymać czytelny projekt.

PROMPT DO CODEXA / Prompt 01 / karta kontraktu

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przeczytaj WP_DEMO_LAB_CONTRACT.md i README tomu 04. Przygotuj krótką kartę wymagań wp-demo-library: użytkownicy, pola, statusy, publiczność uploads, capability i dziesięć warunków odbioru. Użyj dokładnych nazw CPT, REST base i meta. Każdemu wymaganiu przypisz test pozytywny oraz negatywny. Nie zmieniaj plików, nie dodawaj funkcji, nie używaj danych klienta. Wskaż szczególnie różnicę między ochroną szkicu a dostępnością pliku.

KONTROLA: Sprawdź kompletność nazw i to, czy każdy warunek da się zaobserwować na stagingu.

PRACOWNIA / PRO 04

Granice między trzema warstwami

Rozdzielenie wyglądu, funkcji domenowej i magazynu plików pozwala wyjaśnić, gdzie działa dana reguła oraz co trzeba zmienić przy nowej potrzebie.

Motyw odpowiada za prezentację biblioteki. Wtyczka definiuje materiał i reguły jego zapisu. Media WordPressa przechowują publiczny załącznik. Ta granica jest ważniejsza niż liczba klas w kodzie. Jeżeli przeniesiesz rejestrację typu do functions.php motywu, zmiana wyglądu może ukryć materiały i utrudnić ich administrację, mimo że rekordy nadal istnieją w bazie.

Wtyczka wykorzystuje mechanizmy rdzenia: edytor, tytuł, treść, thumbnail, taksonomię i API metadanych. Nie buduje równoległego systemu dokumentów. Zachowanie takie jak publikacja pozostaje standardową operacją WordPressa, kontrolowaną przez właściwe capability. Osobne klasy skupiają kod odpowiedzialny za treść, pola, REST oraz cykl życia.

Magazyn plików ma odrębną właściwość: typowe uploads są obsługiwane przez serwer bez pytania o status rekordu materiału. Usunięcie linku z katalogu nie oznacza usunięcia kopii w cache, zakładkach czy wiadomościach. Zatwierdzenie publiczności musi nastąpić przed przesłaniem. Ten projekt nie obiecuje późniejszego zamienienia publicznego pliku w poufny.

Na warsztacie rozpisz odpowiedzialności na konkretnym przykładzie. Nowa okładka to operacja redakcyjna i prezentacyjna; dodatkowe pole wersji to kontrakt domenowy; prywatne pobieranie to zmiana architektury magazynu. Wybór właściwej warstwy ogranicza liczbę przypadkowych zależności i ułatwia analizę błędów po aktualizacji.

Przed kolejną funkcją wskaż, z której warstwy pochodzi jej wejście i gdzie kończy się odpowiedzialność wtyczki. Diagram jest narzędziem rozmowy z zespołem, a nie dowodem, że hosting realizuje zaprojektowane granice.

Od redakcji do publicznego pliku

01 Panel

Uprawniony człowiek opisuje materiał.

02 Wtyczka

Model treści, walidacja i prawa zapisu.

03 WordPress

Rekord, taksonomia, meta oraz attachment.

04 Odbiorca

Katalog i uploads przeznaczone do publikacji.

PRACOWNIA / PRO 04

Słownik, którego nie zmieniamy przypadkiem

Nazwy typu, taksonomii i pól są częścią umowy między panelem, API, motywem oraz przyszłym indeksem wiedzy. Zmiana wymaga planu kompatybilności.

Rozróżnij nazwę folderu od identyfikatora typu. wp-demo-library określa pakiet wtyczki, natomiast wdl_material wskazuje rekord treści. REST base wdl-materials tworzy ścieżkę API rdzenia. Nazwa wdl_topic wskazuje taksonomię i może wystąpić w zapytaniu, eksporcie oraz późniejszym indeksowaniu. Zapisz te identyfikatory w jednej karcie przed pracą nad interfejsem.

Meta zaczynające się od podkreślenia są polami technicznymi, ale sam podkreślnik nie zapewnia ochrony. _wdl_file_id przechowuje identyfikator attachment, nigdy dowolny adres. Wartość zero oznacza brak powiązania. _wdl_doc_version przyjmuje liczby od jednego do ustalonego maksimum. Osobna walidacja sprawdza, czy wskazany rekord faktycznie reprezentuje dozwolony plik.

Taksonomia porządkuje temat, nie realizuje uprawnień. Nazwanie terminu „wewnętrzne” nie tworzy prywatnej strefy. Podobnie etykieta „publiczny plik” pomaga redaktorowi, lecz nie zwalnia z kontroli procesu przesyłania. Warto opisać takie pułapki przy samych polach, aby kolejne osoby nie przypisywały nazwom właściwości, których kod nie zapewnia.

Przy zmianie schematu sprawdź wszystkie miejsca odczytu. Dodanie nowego pola może wymagać aktualizacji REST, fixture, testu, motywu i dokumentacji. Usuwanie lub zmiana nazwy istniejącego pola wymaga jeszcze migracji starych rekordów i decyzji, jak obsłużyć dotychczasowych konsumentów. Sama korekta etykiety w panelu zwykle nie wymaga takiej operacji.

Element	Uzgodniona wartość	Znaczenie
Pakiet	wp-demo-library	Oddzielna funkcja domenowa
Typ / REST	wdl_material / wdl-materials	Standardowy rekord i API
Temat	wdl_topic	Klasyfikacja materiałów
Plik	_wdl_file_id: integer	0 lub istniejący attachment
Wersja	_wdl_doc_version: integer	1-999999, numer dokumentu

Bootstrap i zamknięty autoloader

Główny plik wtyczki ma uruchomić dobrze zdefiniowane moduły, a nie wykonywać całą logikę aplikacji przy każdym żądaniu.

Nagłówek plugin.php pozwala WordPressowi rozpoznać nazwę, wersję i wymagania wtyczki. Kontrola ABSPATH blokuje bezpośrednie wykonanie pliku poza kontekstem rdzenia. Stała wskazująca główny plik przydaje się przy ścieżkach zasobów; numer wersji wspiera kontrolę cache skryptu. Nie umieszczamy tutaj danych materiałów ani sekretów.

Autoloader jest celowo prosty. Zamknięta mapa dopuszcza pięć konkretnych klas przestrzeni WP_Demo_Library i przypisuje im znane pliki includes. Nazwa klasy z żądania użytkownika nigdy nie służy do budowy ścieżki. To wystarcza dla małej wtyczki i pozwala prześledzić wszystkie zależności bez systemu budowania czy Composera.

Nie myl ładowania klasy z wykonywaniem jej zadania. Plugin::boot rejestruje hooks, a dopiero właściwy etap WordPressa uruchamia rejestrację treści, zapis lub odpowiedź REST. Dzięki temu formularz nie zapisuje danych podczas zwykłego wczytywania głównego pliku. Lifecycle uruchamia działania administracyjne wymagane przy aktywacji i deaktywacji.

Fragment przedstawia początek mapy, nie całość bootstrapu. Skopiuj pełny katalog EXAMPLES, aby otrzymać wszystkie klasy i hooks. Przy ocenie wygenerowanego kodu sprawdź szczególnie namespace, nazwy plików oraz brak drugiej kopii wtyczki. Błąd w jednej z tych nazw może przerwać uruchomienie jeszcze przed pojawieniem się panelu.

KOD W PRAKTYCE / PHP / Autoloader: jawna mapa klas

```
spl_autoload_register( static function ( $class ) {  
    $map = array(  
        Plugin::class => 'class-plugin.php',  
        Content::class => 'class-content.php',  
        Fields::class => 'class-fields.php',  
        Rest::class => 'class-rest.php',  
        Lifecycle::class => 'class-lifecycle.php',  
    );  
    if ( isset( $map[ $class ] ) ) {  
        require_once __DIR__ . '/includes/' . $map[ $class ];  
    }  
} );
```

PLIK: wp-demo-library/plugin.php

Fragment kompletnego pliku EXAMPLES; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Actions i filters w odpowiednim momencie

Hook jest punktem współpracy z WordPressem. Action wykonuje działanie, a filter zwraca wartość, którą dalszy proces wykorzysta.

W naszym projekcie init rejestruje typ, taksonomię oraz meta. `add_meta_boxes_wdl_material` tworzy interfejs pól na ekranie materiału. `save_post_wdl_material` reaguje na zapis konkretnego typu, a `rest_api_init` rejestruje publiczną trasę. Zdarzenia te mają różne konteksty; żaden handler nie powinien zakładać, że wszystkie dane formularza są zawsze obecne.

Filter `rest_pre_insert_wdl_material` kontroluje metadane przesłane przez API przed zapisem rekordu. Zwraca przygotowany obiekt albo `WP_Error`. Zapomniane `return` może zepsuć proces, nawet gdy walidacja wygląda poprawnie. Przeczytaj sygnaturę callbacku i liczbę przyjmowanych argumentów, zamiast kopiować przypadkowy przykład hooka z innego typu treści.

Każdy callback powinien szybko rozpoznać, czy dotyczy bieżącego żądania. Zasoby JavaScript ładujemy tylko na ekranach edycji `wdl_material`. Zapis pomijamy dla autosave i revision oraz przy braku nonce. Publiczny odczyt nie wywołuje handlera pól i nie zapisuje danych. To upraszcza diagnostykę oraz ogranicza działania uboczne.

Priorytet hooka nie zastępuje kontraktu między wtyczkami. Jeżeli inny moduł nadpisuje meta później, sama zmiana numeru priorytetu nie daje pewności spójności. Znajdź wszystkie źródła zapisu i uzgodnij wspólną walidację. W małym laboratorium pełna mapa pięciu modułów pozostaje możliwa do ręcznego sprawdzenia.

Przećwicz działanie przy dwóch drogach zapisu: tradycyjny formularz panelu oraz core REST. Ich zabezpieczenia uwierzytelniania różnią się, ale zakres dozwolonych wartości musi być taki sam. Diagram pokazuje momenty pracy, nie kolejność wszystkich wewnętrznych hooks rdzenia.

Cykl funkcji wtyczki

01 Wczytanie

Bootstrap dodaje callbacki.

02 Init

Rejestracja typu, tematów i meta.

03 Operacja

Panel albo REST uruchamia właściwą ścieżkę.

04 Walidacja

Dane i prawa są sprawdzane przed zapisem.

05 Odpowiedź

Użytkownik odczytuje wynik operacji.

PRACOWNIA / PRO 04

Małe moduły, czytelne zależności

Podział kodu jest wartościowy wtedy, gdy ułatwia zmianę i weryfikację zachowania. Pięć klas odpowiada pięciu konkretnym obszarom projektu.

Content opisuje model, Fields obsługuje pola i ich walidację, Rest udostępnia katalog, Lifecycle zarządza aktywacją, a Plugin wiąże hooks. Dzięki temu podczas poprawiania zakresu wersji dokumentu wiadomo, gdzie zacząć czytanie. Kod nie musi być rozbudowany, aby mieć wyraźną strukturę.

Wspólną walidację liczb oraz plików umieszczamy w Fields, ponieważ korzystają z niej panel, REST i odczyt. Nie kopiujemy reguły do trzech plików. W większym projekcie można wydzielić usługę domenową, lecz tutaj dodatkowa abstrakcja nie jest konieczna do zrozumienia działania. Istotne, aby każda ścieżka używała zgodnych zakresów i typów.

Nazwy metod mówią o czynności: register, save, read, activate. Statyczne metody są uproszczeniem edukacyjnym, a nie uniwersalnym zaleceniem projektowania aplikacji. Przy rozbudowie o zewnętrzny magazyn lub kolejkę zadań warto rozważyć wstrzykiwanie zależności i osobne adaptery. Najpierw jednak ustal, który problem taki podział rozwiązuje.

Refaktoryzacja powinna zachować obserwowalne zachowanie. Zmiana nazwy klasy nie może zmienić capability, ścieżki REST czy zakresu MIME. Zanim zaakceptujesz wynik Codexa, porównaj listę hooks, callbacków i pól przed zmianą oraz po niej. Uruchom te same przypadki negatywne, szczególnie zapis bez uprawnień i błędne ID pliku.

PROMPT DO CODEXA / Prompt 02 / czytelność PHP

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przeanalizuj pięć klas wp-demo-library i bootstrap. Przygotuj mapę odpowiedzialności oraz zależności. Wskaż miejsca powielania reguł i proponuj wyłącznie małe poprawki czytelności zachowujące API, nazwy CPT, capability i dane. Nie dodawaj frameworka ani Composera. Jeżeli zmienisz kod, pokaż konkretny diff i wyjaśnij wpływ każdej zmiany na panel, REST, aktywację i uninstall. Nie deklaruj testów PHP, których nie wykonałeś.

KONTROLA: Odbierz zmianę tylko wtedy, gdy zachowanie kontraktu oraz wszystkie ścieżki zapisu pozostają zgodne.

Typ treści z prawami własnej domeny

Custom Post Type pozwala wykorzystać standardową redakcję WordPressa, zachowując odrębne capability dla materiałów i ich publikacji.

wdl_material jest typem publicznym, ponieważ biblioteka prezentuje zatwierdzone materiały. show_in_rest udostępnia API rdzenia i umożliwia użycie edytora blokowego. rest_base ustala przewidywalną ścieżkę. Lista supports obejmuje tytuł, treść, okładkę, rewizje i custom-fields. Ostatni element jest potrzebny do poprawnej obsługi zarejestrowanych meta przez REST.

Własna para capability_type wskazuje liczbę pojedynczą i mnogą. map_meta_cap pozwala WordPressowi rozstrzygać edit_post dla konkretnego materiału na podstawie autora, statusu oraz praw użytkownika. Nie zastępuj tego sprawdzeniem samej nazwy roli. Administrator i redaktor mogą mieć zmienione zestawy praw na rzeczywistej witrynie.

Rejestracja typu nie nadaje automatycznie nowych capability. Osobna aktywacja przypisuje uzgodnione prawa wyłącznie administratorowi. Bez tego etap rejestracji może działać, a panel materiałów pozostanie niedostępny. Sprawdź oba elementy zamiast omijać problem przez globalne uprawnienie manage_options w każdej funkcji.

Fragment pokazuje część konfiguracji typu. Etykiety, archiwum, rewrite oraz taksonomia znajdują się w kompletnym pliku. Przy pierwszej instalacji odświeżenie reguł adresów wykonuje aktywacja; nie dodawaj flush_rewrite_rules do każdego init. To kosztowna operacja administracyjna, a nie standardowy krok odczytu strony.

KOD W PRAKTYCE / PHP / Typ treści i osobne capability

```
'show_in_rest' => true,
'rest_base' => 'wdl-materials',
'has_archive' => true,
'rewrite' => array( 'slug' => 'materialy-demo' ),
'capability_type' => array(
    'wdl_material', 'wdl_materials',
),
'map_meta_cap' => true,
'supports' => array(
    'title', 'editor', 'thumbnail',
    'revisions', 'custom-fields',
),
) );
```

PLIK: wp-demo-library/includes/class-content.php

Fragment kompletnego pliku EXAMPLES; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Temat jest klasyfikacją, okładka jest obrazem

Unikamy dodatkowych pól tam, gdzie WordPress ma gotową, zrozumiałą strukturę. Dzięki temu redaktor korzysta ze standardowych narzędzi.

wdl_topic grupuje materiały według tematu, na przykład poradniki lub listy kontrolne. Taksonomia jest hierarchiczna, więc później można utworzyć podtematy. W ćwiczeniu zaczynamy od płaskiej, małej listy. Zbyt głębokie drzewo utrudnia odnalezienie treści i nie daje dodatkowej ochrony danych.

Przypisywanie istniejącego tematu wymaga edit_wdl_materials. Zarządzanie terminami wymaga osobnego manage_wdl_topics. Ten podział pozwala później rozdzielić redakcję treści od porządkowania słownika. Nie przyznajemy prawa zmiany słownika przypadkowo tylko dlatego, że użytkownik może publikować materiał.

Tytuł, opis i okładka pozostają standardowymi polami. Featured image jest obrazem prezentacyjnym, a _wdl_file_id wskazuje dokument do pobrania. To dwa różne załączniki i różne cele. Fotografię PNG można wybrać jako okładkę, ale nie powinna przejść walidacji pola dokumentu, którego lista dopuszcza PDF, TXT i demo DOCX.

W panelu pokaż redaktorowi przykład kompletnego materiału: zrozumiały tytuł, opis celu dokumentu, jeden temat, fikcyjną okładkę i zatwierdzony plik. Dobrze opisane dane testowe ułatwiają odbiór API oraz przyszłych widoków. Nie trzeba od razu tworzyć wielu rekordów ani rozbudowanego importera.

Zadbaj również o semantykę opisu. Nie przechowuj w treści haseł, tajnych linków czy instrukcji wewnętrznych, skoro materiał może zostać opublikowany. Taksonomia nie zmienia tej zasady. Odczyt publiczny traktuje treść opublikowanego materiału jako publiczną informację i zwraca jej tekstową postać.

Dane	Mechanizm	Przykład
Nazwa	post_title	Poradnik demonstracyjny
Opis	post_content	Cel, zakres i sposób użycia
Okładka	featured image	Obraz reprezentujący materiał
Klasyfikacja	wdl_topic	poradniki
Dokument	_wdl_file_id	ID publicznego TXT
Wersja treści	_wdl_doc_version	1, później 2

PRACOWNIA / PRO 04

Tworzymy dane, które obnażają błędy

Dobre fixture pokazują różne statusy i ścieżki odczytu. Są małe, fikcyjne i mają jasno opisane skutki ponownego uruchomienia.

W katalogu fixtures znajdują się dwa rekordy JSON i odpowiadające im TXT. Atelier Północ jest opublikowanym poradnikiem, a Studio Fala szkicem listy kontrolnej. Różnica statusów pozwala sprawdzić, czy własny publiczny endpoint nie zwraca całej bazy materiałów. Oba pliki są publicznymi danymi demonstracyjnymi, więc ich przesłanie nie wymaga ochrony poufności.

Skrypt create-fixtures.php korzysta z WP-CLI i standardowego media_handle_sideload. Tworzy attachment, rekord i powiązania. Jest narzędziem seedowania w kontrolowanym środowisku, a nie publicznym importerem. Wymaga aktywnej wtyczki i istniejącego administratora. Ponowne uruchomienie tworzy kolejne rekordy; README informuje o tej właściwości.

Nie nazywaj skryptu idempotentnym, jeżeli nie rozpoznaje wcześniejszego importu. Przy prawdziwej potrzebie powtarzania dodaj stabilny identyfikator fixture, jawne sprawdzenie istniejących danych i strategię aktualizacji. Dla małego ćwiczenia łatwiejszy jest czysty staging oraz świadome resetowanie danych.

Porównaj trzy widoki: lista w panelu administratora, publiczny katalog oraz bezpośredni URL pliku szkicu. Panel pokazuje szkic, katalog go pomija, ale URL uploads nadal działa. Ten eksperyment pomaga zrozumieć granicę systemu znacznie lepiej niż sam tekst ostrzegawczy.

PROMPT DO CODEXA / Prompt 03 / fixture i statusy

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przeczytaj fixtures/materials.json oraz tests/create-fixtures.php. Sprawdź zgodność danych z modelem wdl_material i wskaż skutki powtórnego uruchomienia. Przygotuj instrukcję utworzenia dwóch fikcyjnych materiałów oraz porównania panelu, własnego GET i bezpośredniego URL TXT szkicu. Nie pobieraj obcych plików. Nie twórz klienta ani jego danych. Zapisz oczekiwane wyniki i odróżnij je od pomiarów faktycznie wykonanych w WordPressie.

KONTROLA: W instrukcji musi być jasno widoczne, że status draft nie chroni przesłanego TXT.

Rejestrujemy meta, opisujemy zakres

Rejestracja pola ustala typ, pojedynczą wartość, schemat REST oraz kontrolę edycji. Walidacja domenowa nadal sprawdza znaczenie identyfikatora.

Dwa pola meta mają różne zakresy. ID może być zerem, aby redaktor mógł usunąć powiązanie bez usuwania pliku. Numer dokumentu zaczyna się od jednego. Ustalone maksimum zabezpiecza przed absurdalnie dużymi wartościami i nadaje kontraktowi jednoznaczną postać. Ograniczenia zapisujemy także w schemacie REST.

`sanitize_callback` jest ostatnią normalizacją liczby w obrębie zakresu. Nie traktujemy go jako pełnej walidacji wejścia: wcześniejsza walidacja panelu i REST odrzuca tablice, ułamki, wartości ujemne oraz błędne attachment. Samo rzutowanie na integer mogłoby zamienić niepoprawny tekst na zero, ukrywając błąd użytkownika.

`auth_callback` sprawdza `edit_post` dla konkretnego materiału. Zapis core REST wymaga dodatkowo standardowych praw kontrolera i poprawnego uwierzytelnienia. Nonce przeglądarkowy nie zastępuje capability. Zaufany kod serwerowy korzystający bezpośrednio z `update_post_meta` musi sam respektować te zasady; metadata API nie uruchamia całego formularza.

Nie przenosimy opisu dokumentu do meta, ponieważ standardowa treść już spełnia tę funkcję. Mniejszy schemat oznacza mniej ścieżek zapisu do kontroli. Fragment pokazuje uprawnienie pola; pełny schemat oraz sanitizer znajdują się w `Content`. Przy rozbudowie sprawdź również wsparcie `custom-fields` na poziomie CPT i zgodność danych istniejących rekordów.

KOD W PRAKTYCE / PHP / Autoryzacja metadanych dla rekordu

```
'sanitize_callback' => static function ( $value )
    use ( $range ) {
        return max( $range[0], min(
            $range[1], (int) $value
        ) );
    },
'auth_callback' => static function (
    $allowed, $meta_key, $post_id
) {
    return current_user_can(
        'edit_post', $post_id
    );
},
) );
```

PLIK: `wp-demo-library/includes/class-content.php`

Fragment kompletnego pliku `EXAMPLES`; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Od wyboru media do zapisu ID

Media picker ułatwia pracę redaktorowi, ale o poprawności powiązania decyduje serwer. JavaScript nigdy nie jest jedyną kontrolą danych.

Po otwarciu edycji materiału wtyczka ładuje `wp_enqueue_media` i skrypt `admin.js`. Zasoby pojawiają się wyłącznie na odpowiednich ekranach, więc inne części panelu nie otrzymują dodatkowej logiki. Picker ogranicza widoczny wybór do uzgodnionych MIME, a po wyborze przepisuje ID załącznika do pola.

Wybrana nazwa pliku jest prezentowana przez `textContent`. Nie wstawiamy HTML pochodzącego z nazwy attachment. Pole z ID nadal pozostaje czytelne, aby redaktor i tester mogli zobaczyć, co zostanie zapisane. Przycisk usunięcia ustawia zero; nie usuwa pliku z biblioteki ani serwera.

Walidacja klienta sprawdza całkowitość i zakres ID oraz wersji. Pokazuje komunikat przy polu i wykorzystuje natywne `validity`. W edytorze blokowym część zapisów może przebiegać inną ścieżką niż klasyczny submit, dlatego interfejs jest pomocą, a serwer pozostaje rozstrzygający. Odbiór obejmuje oba edytory i test z wyłączonym JavaScriptem.

Serwer sprawdza, czy ID istnieje, jest attachment, wskazuje lokalny istniejący plik i ma zgodne rozszerzenie oraz MIME. Dodatkowo użytkownik musi móc edytować ten attachment. Taki wybór jest celowo restrykcyjny; administrator ma odpowiednie prawa, natomiast redaktor wymaga świadomej konfiguracji i testu własności mediów.

Nie poszerzaj MIME do dowolnego dokumentu ani wszystkich typów plików. PDF, TXT i demonstracyjny DOCX wystarczają do ćwiczenia. Ta kontrola nie analizuje treści załącznika i nie wykrywa malware. Używaj wyłącznie znanych, fikcyjnych plików zatwierdzonych do publicznego udostępnienia.

Powiązanie materiału z plikiem

01 Wybór

Media picker pokazuje dopuszczone typy.

02 ID

Formularz przesyła liczbę, nigdy obcy URL.

03 Serwer

Attachment, plik, MIME, zakres i capability.

04 Meta

Poprawne ID wiąże publiczny dokument z rekordem.

PRACOWNIA / PRO 04

Interfejs mówi, co naprawdę się wydarzyło

Komunikat powinien rozróżniać zmianę powiązania, usunięcie pliku i odrzucenie wartości. To pomaga uniknąć fałszywego poczucia ochrony.

Przycisk „Usuń powiązanie” mówi o relacji, nie o fizycznym kasowaniu pliku. Po kliknięciu JS prezentuje informację, że plik pozostaje. Wersja formularza pozwala świadomie wprowadzić liczbę, a metabox przypomina o publiczności uploads. Te komunikaty są częścią bezpieczeństwa procesu, choć nie stanowią kontroli dostępu.

Błąd zapisu pól jest zapisywany jako krótki transient użytkownika i wyświetlany w panelu z `esc_html`. Wtyczka zachowuje stare wartości meta. Standardowy zapis tytułu lub treści może się przy tym udać, ponieważ hook `save_post` reaguje po operacji zapisu rekordu. Nie obiecujemy transakcji obejmującej wszystkie pola i cały WordPress.

W odbiorze sprawdź, czy redaktor potrafi odpowiedzieć na trzy pytania: jaki plik jest powiązany, jaka wersja jest zapisana i czy plik może być publiczny. Po błędzie przeładuj edytor i odczytaj rzeczywiste meta. Komunikat klienta może opisywać wartość wpisaną w formularzu, podczas gdy serwer zachował poprzednią.

Codex może pomóc poprawić etykiety i dostępność, ale nie powinien ukrywać krytycznych informacji dla estetyki. Zachowaj label, role status oraz role alert. Sprawdź klawiaturę i czytelność komunikatu bez polegania na samym kolorze. Każda zmiana interfejsu wymaga powtórzenia ścieżki wyboru, usunięcia powiązania i odrzuconego zapisu.

PROMPT DO CODEXA / Prompt 04 / metabox bez niejasności

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Sprawdź render metaboxu i `assets/admin.js`. Oceń etykiety, powiązania label, obsługę klawiatury oraz komunikaty wyboru i usunięcia. Zaproponuj drobne ulepszenia po polsku bez zmiany kontraktu pól. Zachowaj jawny tekst o publicznym uploads. Nie zamieniaj ID na dowolny URL. Każdą poprawkę powiąż z testem w edytorze blokowym, klasycznym oraz przy wyłączonym JS. Oddziel kontrolę klienta od walidacji PHP.

KONTROLA: Po usunięciu powiązania użytkownik musi wiedzieć, że plik nadal istnieje i może pozostać publiczny.

Nonce i capability sprawdzają różne rzeczy

Nonce wiąże formularz z kontekstem żądania. Capability rozstrzyga, czy bieżący użytkownik może wykonać operację na konkretnym rekordzie.

W metaboxie tworzymy nonce związany z ID materiału. Dzięki temu token innego wpisu nie powinien zatwierdzać tej operacji. Po stronie zapisu weryfikujemy typ wartości, usuwamy slashy WordPresa i sprawdzamy nonce. Brak tokenu oznacza pominięcie własnych pól, co jest prawidłowe dla wielu innych dróg zapisu rekordu.

Osobna kontrola `current_user_can` z `edit_post` i ID materiału ustala prawo użytkownika. Nie sprawdzamy wyłącznie tego, czy ktoś jest zalogowany. Nie porównujemy też nazwy roli do administratora w każdym handlerze. Rdzeń mapuje capability rekordu zgodnie z jego właścicielem i statusem.

Nonces WordPresa nie są jednorazowymi biletami i nie chronią przed każdym powtórzeniem żądania. Dlatego działanie powinno być przewidywalne przy wielokrotnym zapisie tych samych pól. Użytkownik z poprawnym nonce, ale bez praw, nie otrzymuje dostępu. Sama obecność sekretne parametru w formularzu nie stanowi autoryzacji.

Fragment pokazuje sprawdzanie tokenu po wcześniejszych warunkach typu, autosave, revision i capability. Pełnego handlera nie zastępuje kilka linii. Przy przeglądzie Codexa zawsze czytaj otaczający kontekst, ponieważ poprawne `wp_verify_nonce` nie zabezpieczy kodu, który wcześniej zapisał wartości albo później ignoruje wynik kontroli.

KOD W PRAKTYCE / PHP / Nonce związany z ID materiału

```
if ( ! isset( $_POST['wdl_nonce'] )
    || ! is_string( $_POST['wdl_nonce'] ) ) { return; }
$nonce = sanitize_text_field(
    wp_unslash( $_POST['wdl_nonce'] )
);
if ( ! wp_verify_nonce( $nonce, 'wdl_save_' . $post_id ) ) {
    return;
}
```

PLIK: wp-demo-library/includes/class-fields.php

Fragment kompletnego pliku EXAMPLES; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Nadajemy najmniejszy potrzebny zakres praw

Pierwsza aktywacja przyznaje prawa materiałów tylko administratorowi. Redaktor otrzymuje je dopiero po jawnej decyzji oraz testach własności treści i mediów.

Lifecycle posiada pełną listę primitive capabilities materiałów oraz `manage_wdl_topics`. Administrator otrzymuje tylko brakujące prawa, a wtyczka zapisuje listę faktycznie nadanych capability. Dzięki temu `uninstall` może odróżnić własne nadania od praw istniejących wcześniej. Dane materiałów nie są przy tym kasowane.

Editor nie otrzymuje automatycznie nowych uprawnień. W README opisano opcjonalne nadanie praw do edycji i publikacji materiałów. Prawa edycji cudzych rekordów i zarządzania tematami wymagają osobnej decyzji. Przypisanie jednego szerokiego uprawnienia wszystkim użytkownikom byłoby trudne do uzasadnienia i utrudniłoby kontrolę późniejszych zmian.

Ważna jest również własność załącznika. Nasza walidacja wymaga `edit_post` dla wybranego `attachment`. Redaktor mający prawo edycji materiału może nadal nie mieć prawa do mediów innego autora. Test powinien obejmować własny plik oraz cudzy, zamiast zakładać, że widoczność w `media pickerze` oznacza zgodę na powiązanie.

Rola jest zestawem praw, który organizacja może zmieniać. Zapisz użytkownika testowego, aktualne capability i autora każdego rekordu. Bez tych informacji wynik testu odmowy bywa nieczytelny. Na `stagingu` przygotuj administratora, opcjonalnego redaktora oraz `subscriber`; nie korzystaj z konta klienta.

Odebranie prawa po publikacji nie usuwa treści ani pliku. To zmiana możliwości kolejnych operacji. Jeśli materiał ma zostać wycofany publicznie, potrzebny jest osobny proces redakcyjny oraz analiza `cache` i wcześniej rozpowszechnionych URL.

Operacja	Kontrola	Domyślny dostęp
Edycja rekordu	<code>edit_post(ID)</code>	Administrator
Publikacja	<code>publish_wdl_materials</code>	Administrator
Zmiana słownika	<code>manage_wdl_topics</code>	Administrator
Wybór attachment	<code>edit_post(file_ID)</code>	Uprawniony do tego medium
Publiczny katalog	Jawny GET <code>publish</code>	Każdy odbiorca

PRACOWNIA / PRO 04

Testujemy odmowę, nie tylko wygodny sukces

Bez prób negatywnych trudno stwierdzić, czy kod realizuje granicę dostępu. Poprawne konto administratora zwykle maskuje błędy w konfiguracji.

Zacznij od użytkownika bez capability materiałów. Nie powinien móc edytować rekordu ani jego meta, nawet jeżeli otrzyma poprawny nonce w kontrolowanym teście. Następnie sprawdź redaktora z ograniczonym zestawem praw oraz własny i cudzy materiał. Zapisz odpowiedź API i potwierdź stan w bazie po odmowie.

Test nonce obejmuje brak, zły token, token innego rekordu oraz prawidłowy token. Test capability obejmuje anonimowego użytkownika, subscriber i administratora. To niezależne osie. Przypadek „zły token administratora” nie sprawdza, czy użytkownik bez praw jest odrzucany.

W core REST wewnętrzny dispatcher smoke sprawdza prawa użytkownika ustawionego w procesie. Nie odtwarza jednak transportu HTTP ani uwierzytelnienia cookie. Osobno wykonaj żądania z panelu z X-WP-Nonce i próbę bez tokenu. W przypadku application password kontrola tokenu cookie ma inne znaczenie; nie mieszaj tych sposobów uwierzytelniania w jednym opisie testu.

Każdy test odmowy powinien sprawdzać więcej niż kod odpowiedzi. Odczytaj stare meta i porównaj, czy nie zmieniły się pomimo komunikatu błędu. Sprawdź także brak niezamierzonej publikacji. Dziennik zawiera dane fikcyjne, identyfikatory testowe i wynik, nigdy cookie ani hasło aplikacyjne.

PROMPT DO CODEXA / Prompt 05 / macierz uprawnień

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Na podstawie Lifecycle, Content i Fields przygotuj macierz testową: administrator, editor bez nowych praw, editor z jawnie nadanymi prawami oraz subscriber. Uwzględnij własny i cudzy materiał, własny i cudzy attachment, publikację, tematy oraz nonce. Dla każdego przypadku podaj oczekiwany status, dozwoloną zmianę i sposób porównania meta przed oraz po. Nie nadawaj praw automatycznie i nie zapisuj tokenów w raporcie. Oddziel dispatcher REST od testu HTTP.

KONTROLA: W macierzy poprawny nonce nigdy nie jest wystarczającą przyczyną dostępu.

Walidacja przed normalizacją

Najpierw rozstrzygamy, czy wartość należy do dozwolonej dziedziny. Dopiero poprawną wartość normalizujemy i zapisujemy w ustalonym formacie.

Pole integer ma przyjmować liczbę całkowitą albo jej dziesiętny zapis z formularza. Nie akceptujemy tablicy, adresu URL, wartości ujemnej, ułamka ani tekstu „123abc”. Fields::integer sprawdza typ, cyfry oraz zakres przy użyciu FILTER_VALIDATE_INT. Ta kolejność ogranicza przypadkową zamianę niepoprawnych danych w poprawnie wyglądające zero.

Walidacja pliku rozstrzyga inne pytanie: czy identyfikator wskazuje istniejący attachment dopuszczonego typu. Sprawdza lokalny plik i zgodność rozszerzenia z MIME rekordu. Zero jest jawnie dozwolone jako brak powiązania. Nie zgadujemy pliku na podstawie tytułu ani nie pobieramy treści spod wejściowego URL.

Sanitization nie jest synonimem bezpieczeństwa. sanitize_text_field jest odpowiednie do tokenu tekstowego czy prostego pola, ale nie naprawi brakujących praw. absint może zamienić wartość ujemną na dodatnią, czego tutaj nie chcemy. Zakres oraz znaczenie danych powinny wynikać z kontraktu, a nie z wygody funkcji pomocniczej.

Escaping następuje podczas wyświetlania w konkretnym kontekście. ID w atrybucie korzysta z esc_attr, komunikat z esc_html, a URL linku z odpowiedniej funkcji adresowej. W JSON nie wkładamy encji HTML do zwykłego tekstu. Klient katalogu wyświetla tekst przez textContent; inny renderer musi dobrać własną ochronę.

Reguły zapisujemy w jednym miejscu i wykorzystujemy w dwóch ścieżkach wejścia. Nie ma sensu rygorystyczny formularz, jeśli REST nadal akceptuje błędne ID. Testy obejmują obie drogi i sprawdzają zachowanie przy wartości niedostępnego attachment.

Trzy niezależne operacje

01 Walidacja

Typ, zakres i znaczenie wartości.

02 Normalizacja

Ustalony integer do zapisania.

03 Zapis

Uprawnione powiązanie w meta.

04 Wyświetlenie

Escaping dobrany do HTML lub JSON.

PRACOWNIA / PRO 04

Guard clauses w zapisie materiału

Krótką listą warunków wyjścia ogranicza ryzyko, że zapis autosave albo niepowiązane żądanie zmieni pola biblioteki.

`save_post_wdl_material` może zostać wywołany w więcej niż jednym kontekście. Najpierw potwierdzamy `post_type`, pomijamy autosave i revision oraz sprawdzamy `edit_post` dla ID. Następnie wymagamy nonce własnego metaboxu. Brak danych formularza nie jest dowodem ataku; inny mechanizm rdzenia może zapisywać ten sam rekord.

Po `wp_unslash` walidujemy obie wartości, zanim wywołamy `update_post_meta`. Dzięki temu nieprawidłowa wersja nie prowadzi do wcześniejszego zapisania nowego ID pliku w tej ścieżce. To uporządkowanie operacji, nie pełna transakcja bazy. Błąd infrastruktury podczas dwóch zapisów nadal wymaga diagnostyki.

Sprawdzenie `edit_post` dla attachment jest odrębne od prawa edycji materiału. Zabezpiecza wybór medium, nad którym użytkownik nie ma odpowiedniej kontroli. Do zera, oznaczającego brak powiązania, nie próbujemy mapować capability pliku. Rekord oraz załącznik mają różne identyfikatory i trzeba wyraźnie rozróżniać je w testach.

Nieprawidłowe wartości powodują komunikat i zachowanie dotychczasowych meta. Standardowy post może zostać wcześniej zapisany. Jeśli przyszły produkt wymaga atomowego zatwierdzania wszystkich pól, potrzebuje osobnego projektu operacji oraz kontroli błędów; dodanie pojedynczego return do `save_post` nie zapewni takiej gwarancji. Fragment pokazuje pierwszą bramkę handlera, a pełny plik wyjaśnia dalsze kroki.

KOD W PRAKTYCE / PHP / Pomiń nieodpowiedni kontekst zapisu

```
public static function save( $post_id, $post, $update ) {
    if ( 'wdl_material' !== $post->post_type
        || wp_is_post_autosave( $post_id )
        || wp_is_post_revision( $post_id )
        || ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE )
        || ! current_user_can( 'edit_post', $post_id ) ) {
        return;
    }
}
```

PLIK: `wp-demo-library/includes/class-fields.php`

Fragment kompletnego pliku EXAMPLES; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Zabezpieczamy wyjście oraz komunikaty

Poprawny zapis nie gwarantuje bezpiecznego HTML. Dane trzeba prezentować zgodnie z kontekstem, również gdy pochodzą z pozornie zaufanego panelu.

Tytuł materiału i nazwa pliku mogą zawierać znaki specjalne. W naszym endpointcie opis oraz tytuł są tekstem pozbawionym tagów. W panelu wartości input przechodzą przez `esc_attr`, a komunikat przez `esc_html`. JavaScript wyświetla nazwę pliku przez `textContent`, więc nie interpretuje jej jako fragmentu dokumentu HTML.

Adres załącznika pochodzi z `wp_get_attachment_url`, a nie z dowolnego parametru użytkownika. `esc_url_raw` porządkuje wartość przeznaczoną do JSON. Jeżeli później stworzysz link HTML po stronie serwera, zastosuj `esc_url` dla atrybutu `href`. Nie kopiuj encji z jednego kontekstu do innego bez zrozumienia, gdzie zostaną odczytane.

Testuj fikcyjne ciągi zawierające cudzysłów, znak mniejszości i ampersand. Zwykłe wyświetlenie tych znaków nie powinno uruchamiać skryptu ani uszkadzać struktury pola. Nie umieszczaj aktywnych payloadów w publicznej produkcji; staging oraz niewrażliwe dane wystarczają do sprawdzenia renderera.

Komunikat błędu nie powinien ujawniać pełnej ścieżki serwera, sekretu czy danych innego użytkownika. W metaboxie podajemy krótką wskazówkę o ID, MIME i wersji. Dokładniejsze szczegóły diagnostyczne można zapisać w kontrolowanym dzienniku developerskim, ale bez treści dokumentu i tokenów uwierzytelniania.

PROMPT DO CODEXA / Prompt 06 / przegląd wejścia i wyjścia

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przejdź przez `render`, `save`, `rest_validate` oraz `assets/admin.js`. Zrób listę wszystkich wejść użytkownika i miejsc wyświetlania danych. Dla każdego wskaż walidację, normalizację, `capability`, `nonce` oraz escaping odpowiedni do kontekstu. Przygotuj testy znaków specjalnych w tytule i nazwie pliku. Zaproponuj tylko konkretne poprawki z odniesieniem do pliku. Nie twierdź, że skanowanie MIME jest skanowaniem malware ani że `sanitize` nadaje uprawnienie.

KONTROLA: Odbiór wymaga braku dowolnego URL wejściowego i zgodności zabezpieczeń panelu oraz REST.

Wykorzystujemy REST rdzenia do zapisu

Core REST już obsługuje tworzenie i aktualizowanie treści. Dodatkowy własny endpoint jest tutaj potrzebny jedynie do małego, publicznego katalogu.

Zarejestrowany CPT z `show_in_rest` otrzymuje standardową trasę `wdl-materials`. Korzystamy z kontrolera WordPressa do zapisu tytułu, treści, statusu, taksonomii i meta. Schemat opisuje liczbę, a `rest_pre_insert_wdl_material` sprawdza znaczenie przesłanych identyfikatorów. Nie budujemy drugiej wersji procesu publikacji.

Zalogowana przeglądarka korzystająca z cookies przekazuje `X-WP-Nonce` utworzony dla `wp_rest`. Serwerowa integracja może użyć indywidualnego `application password` przez HTTPS, jeśli projekt tego wymaga. Obie drogi zachowują ograniczenia `capability` konta. Nie umieszczamy hasła ani cookie w przykładzie JavaScript i nie zapisujemy ich w repo.

Własna trasa GET ma jawny `permission_callback` zwracający zgodę, ponieważ jej celem jest publiczny odczyt już zatwierdzonych danych. Ta jawność nie powinna być kopiowana do operacji zapisu. Endpoint z `__return_true` oraz POST wymagałby osobnego uzasadnienia, którego ten projekt nie ma.

Fragment pokazuje rejestrację publicznego odczytu i pierwszą definicję paginacji. Callback `read` zawsze ustala `publish` i wyklucza rekordy z hasłem. Żądanie nie może podmienić statusu na `draft`. Zapis z przykładowym JSON wykonuj tylko po podstawieniu rzeczywistego ID własnego publicznego `attachment` i sprawdzeniu bieżącego użytkownika.

KOD W PRAKTYCE / PHP / Jawnie publiczna trasa odczytu

```
public static function register() {
    register_rest_route( 'wp-demo-lab/v1', '/materials', array(
        'methods' => 'GET',
        'callback' => array( self::class, 'read' ),
        // Public files: anonymous read is intentional.
        'permission_callback' => '__return_true',
        'args' => array(
            'page' => array(
                'type' => 'integer', 'minimum' => 1,
                'maximum' => 10000, 'default' => 1,
            ),
            'per_page' => array(
                'type' => 'integer', 'minimum' => 1,
                'maximum' => 20, 'default' => 10,
            ),
        ),
    ),
```

PLIK: `wp-demo-library/includes/class-rest.php`

Fragment kompletnego pliku `EXAMPLES`; test WordPress/PHP niewykonany.

PRACOWNIA / PRO 04

Publiczny odczyt ma własny kontrakt

Katalog zwraca mały, przewidywalny zestaw pól. Nie ujawnia szkiców, prywatnych rekordów ani arbitralnie wybranych statusów z żądania.

Callback tworzy WP_Query z post_type wdl_material oraz post_status publish. Ustawia has_password na false, dzięki czemu wpis chroniony hasłem nie trafi do uproszczonego tekstowego katalogu. Ograniczenie to ma znaczenie, ponieważ samo publish nie opisuje wszystkich możliwych warunków dostępności treści w WordPressie.

Parametr topic jest slugiem wdl_topic. Schemat ogranicza długość i znaki, a sanitization porządkuje wartość przed zbudowaniem tax_query. Nie interpolujemy parametru do SQL. Pusta wartość oznacza brak filtrowania tematu. Parametr per_page ma maksymalnie dwadzieścia, a page ograniczoną dodatnią liczbę.

Odpowiedź zawiera ID, tytuł, tekstowy opis, wersję, attachmentUrl i coverUrl. Gdy powiązany plik zniknie lub przestanie być poprawny, attachmentUrl jest null. To świadoma postać braku danych, którą klient powinien obsłużyć komunikatem lub ukryciem przycisku. Nie tworzymy fałszywego linku z samego numeru ID.

URL jest celowo publiczny. Zmiana rekordu na draft usunie go z tego katalogu, ale nie wycofa pliku z uploads. Również publiczna kopia w zewnętrznym cache może nadal istnieć. Przed udostępnieniem materiału zatwierdź treść i publiczność pliku, a po wycofaniu zaplanuj odpowiednie działania operacyjne.

Nagłówki X-WP-Total i X-WP-TotalPages pomagają klientowi sterować nawigacją. Nie zakładaj, że ostatnia strona zawsze zawiera pełną liczbę elementów. Test pustego tematu, nieistniejącego slugu i strony poza wynikami powinien dać przewidywalną, poprawną odpowiedź.

Publiczny GET materiałów

01 Żądanie

Topic, page oraz per_page w schemacie.

02 Zapytanie

Tylko publish i brak hasła.

03 Rekordy

Tekst, wersja i poprawny attachment.

04 JSON

Jawnie publiczne pola oraz paginacja.

PRACOWNIA / PRO 04

Przegląd API przez konkretne żądania

API oceniamy na podstawie rzeczywistych parametrów, odpowiedzi i zmian danych. Samo istnienie trasy w indeksie nie potwierdza bezpieczeństwa.

Na stagingu wykonaj anonimowy GET własnej trasy i sprawdź brak szkicu. Następnie prześlij niepoprawny limit, stronę zero oraz slug z niedozwolonymi znakami. Schemat powinien odrzucić żądania spoza kontraktu. Nieposiadający uprawnień klient nie powinien zaktualizować meta przez core REST.

Dla uprawnionego użytkownika przetestuj rzeczywisty attachment TXT oraz ID zwykłego posta. Pierwsze powinno zostać zapisane, drugie odrzucone. Sprawdź stary numer wersji i powiązanie po błędzie. Jeżeli formularz oraz API zachowują się inaczej, znajdź różnicę walidacji przed dopisaniem nowego wyjątku.

Nie zapisuj pełnych nagłówków uwierzytelnienia do raportu. Wystarczy sposób auth, rola lub zestaw capability, ścieżka, kod odpowiedzi i porównanie danych. Ekran terminala z cookie czy application password nie jest odpowiednim materiałem do publikacji w podręczniku.

Własny endpoint nie przyjmuje URL do pobrania ani identyfikatora prywatnego magazynu. Jeśli późniejsza integracja potrzebuje importu, projektuj ją jako oddzielną operację z jasną autoryzacją i listą dozwolonych źródeł. Nie rozszerzaj publicznego GET przypadkowym parametrem fetch, który zmieni go w serwer pobierający dowolne adresy.

PROMPT DO CODEXA / Prompt 07 / kontrakt REST

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przygotuj serię żądań na `https://example.test`: publiczny GET z tematem i paginacją, limit 21, page 0, niepoprawny topic oraz core REST zapis meta. Wskaż, które wymagają auth i jakie capability muszą istnieć. Używaj symbolicznych ID do podstawienia, bez haseł i cookie w plikach. Porównaj wynik z REST schema oraz `rest_validate`. Wyjaśnij null `attachmentUrl` i zachowanie `publish`, `draft`, `private` oraz wpisu z hasłem.

KONTROLA: Nie traktuj `__return_true` jako wzorca dla zapisu i nie deklaruj wykonania żądań bez raportu.

Zapytanie z najmniejszą potrzebną swobodą

Ograniczamy parametry wejściowe do danych, które odbiorca faktycznie potrzebuje. Stałe warunki bezpieczeństwa pozostają po stronie serwera.

WP_Query przyjmuje tablicę argumentów zamiast ręcznie zbudowanego SQL. Ustalamy typ oraz status, limitujemy stronę i rozmiar odpowiedzi. Kolejność według daty oraz ID ułatwia przewidywanie wyników, chociaż publikacje pomiędzy kolejnymi zadaniami mogą przesuwac granice stron. Prosta paginacja nie daje migawki całego katalogu.

Filtr tematu korzysta z tax_query i nazwy wdl_topic. Nie pozwalamy klientowi wybierać dowolnej taksonomii, klucza meta, kolumny sortowania ani warunku status. To ogranicza powierzchnię błędów i pozwala zrozumieć koszt zapytania. Własne rozszerzenia dodawaj dopiero po zdefiniowaniu potrzeb oraz testów.

Całkowita liczba wyników wymaga pracy bazy. Dla małej biblioteki jest uzasadniona, ponieważ służy nawigacji. W większym katalogu zmierz opóźnienie na reprezentatywnych danych i zdecyduj, czy potrzebne są dokładne sumy. Nie obiecuj konkretnego czasu odpowiedzi na podstawie samej liczby linii kodu.

Cache publicznej odpowiedzi musi uwzględniać topic i paginację oraz unieważnienie przy zmianie publikacji. Wycofanie materiału może pozostawić stary wynik w CDN. Ten przykład nie implementuje rozbudowanego cache; jego dodanie wymaga osobnego testu publikacji, zmiany wersji i wycofania.

Jeśli strona pokazuje katalog zalogowanemu użytkownikowi z innymi danymi, nie łącz jej cache z publiczną odpowiedzią. Najpierw rozróżnij faktycznie publiczny odczyt od operacji prywatnych, które rozwija dalszy tom.

Parametr	Granica	Przyczyna
post_type	Stałe wdl_material	Wąski zakres danych
post_status	Stałe publish	Brak szkiców w katalogu
has_password	false	Brak obejścia ochrony treści
per_page	1-20	Ograniczona odpowiedź
page	1-10000	Przewidywalny schemat
topic	Slug, maksimum 80 znaków	Jedna klasyfikacja

PRACOWNIA / PRO 04

Prepared SQL, gdy rzeczywiście jest potrzebne

Standardowe API wystarcza do tego katalogu. Znajomość przygotowanych zapytań jest potrzebna wtedy, gdy nowa funkcja wymaga własnej tabeli lub specjalnego odczytu.

Wtyczka nie tworzy tabeli i nie wykonuje własnego SQL. Nie dodajemy demonstracyjnego zapytania tylko po to, aby zwiększyć liczbę mechanizmów. Posty, termy i meta mają gotowe API. Ta decyzja ułatwia kompatybilność oraz zmniejsza zakres testów migracyjnych pierwszego wydania.

Jeżeli przyszła funkcja potrzebuje własnej tabeli, wszystkie wartości z wejścia przekazuj przez `wpdb::prepare` i właściwe placeholdery. Typ wartości ma znaczenie: integer, tekst oraz identyfikator SQL to różne przypadki. Nie wkładaj danych żądania bezpośrednio do fragmentu `WHERE`, `ORDER BY` ani nazwy kolumny.

Prepared query nie rozstrzyga autoryzacji. Poprawnie przygotowane zapytanie może nadal odczytać rekordy, których użytkownik nie powinien widzieć. Dlatego stałe warunki publiczności oraz kontrola capability muszą pozostać częścią projektu. Podobnie `sanitize` nie zastępuje przygotowania wartości SQL.

Nazwy tabel ustalaj z kontrolowanego prefiksu i wewnętrznej definicji. Dopuszczalne kolumny sortowania wybieraj z zamkniętej mapy. Jeśli obsługujesz wyszukiwanie `LIKE`, zadбай o właściwe escapowanie wzorca przed przygotowaniem zapytania. Nie twórz dynamicznej składni na podstawie dowolnego parametru.

W tym tomie zadaniem Codexa jest przegląd i uzasadnienie potrzeby, nie implementacja zbędnego magazynu. Przyszła zmiana powinna zawierać schemat, właściciela migracji, indeksy, strategię rollback oraz test wykonany na reprezentatywnych danych. Dopiero wtedy można ocenić koszt i zgodność.

PROMPT DO CODEXA / Prompt 08 / zapytania bez interpolacji

Skopiuj do Codexa razem z kompletnym katalogiem `EXAMPLES`.

Przejrzyj `EXAMPLES` pod kątem odczytu bazy. Wypisz wszystkie `WP_Query`, operacje meta i termów oraz ich źródła parametrów. Potwierdź, czy własny SQL jest potrzebny do aktualnego kontraktu. Jeżeli proponujesz przyszłą tabelę, najpierw opisz uzasadnienie i pokaż zasady `wpdb::prepare`, zamkniętej mapy sortowania i kontroli dostępu. Nie dodawaj interpolowanego SQL ani przypadkowej tabeli do działającego projektu.

KONTROLA: Prepared query ma chronić składnię; testy muszą osobno sprawdzać publiczność i uprawnienia.

PRACOWNIA / PRO 04

Odpowiedź odporna na brak pliku

Powiązany attachment może zniknąć po zapisaniu materiału. Odczyt ponownie sprawdza plik i daje jednoznaczną reprezentację braku danych.

Publiczny katalog nie ufa bezwarunkowo staremu ID. Dla każdego materiału sprawdza attachment przez wspólną funkcję `file_valid` i dopiero potem pobiera URL. Usunięty fizyczny plik, błędny MIME lub niepoprawny typ rekordu powodują null. Dzięki temu odpowiedź nie tworzy linku sugerującego istnienie poprawnego dokumentu.

Treść opisu jest zwracana jako tekst. To świadoma redukcja formatu, przydatna dla prostych list i przyszłego indeksu. Katalog nie wykonuje dowolnych shortcode z opisu i nie zamienia treści materiału w dodatkowy kod wyjściowy. Jeśli produkt potrzebuje HTML, zaprojektuj osobny kontrakt renderowania i testy bezpieczeństwa.

Wersja ma co najmniej jeden. Stary rekord bez meta nie powinien łamać prostego konsumenta. Taka wartość domyślna nie zastępuje jednak świadomej migracji, jeśli zakres albo znaczenie pola zmieni się w następnych wydaniach. Odróżnij tolerancję odczytu od gwarancji jakości wszystkich zapisanych danych.

Obsługa null musi być widoczna w kliencie katalogu: informacja o braku pliku albo brak przycisku pobrania. Nie próbuj konstruować zastępczego URL z samego ID. Fragment pochodzi z pełnego callbacku, który dodatkowo ustala status i paginację. Sprawdzenie pliku nie jest kontrolą prywatności uploads ani skanowaniem jego zawartości.

KOD W PRAKTYCE / PHP / URL tylko poprawnego attachment

```
$id = (int) get_post_meta(
    $post->ID, '_wdl_file_id', true
);
$url = $id && Fields::file_valid( $id )
    ? wp_get_attachment_url( $id ) : false;
$items[] = array(
    'id' => $post->ID,
    'title' => wp_strip_all_tags( $post->post_title ),
    'description' => wp_strip_all_tags(
        $post->post_content
    ),
    'version' => max( 1, (int) get_post_meta(
        $post->ID, '_wdl_doc_version', true
    ) ),
    'attachmentUrl' => $url
        ? esc_url_raw( $url ) : null,
    'coverUrl' => get_the_post_thumbnail_url(
        $post->ID, 'medium'
    ) ?: null,
);
```

PLIK: wp-demo-library/includes/class-rest.php

Fragment kompletnego pliku EXAMPLES; test WordPress/PHP niewykonany.

Test ma dowieść konkretnego zachowania

Oddzielamy lokalne sprawdzenia składni od integracji WordPressa i testu użytkownika. Każdy raport wskazuje środowisko oraz rzeczywiście wykonany zakres.

`node --check` potwierdza poprawność składni `admin.js`. Odczyt JSON sprawdza strukturę `fixture` i zgodność nazw plików. Te wyniki są przydatne, ale nie dowodzą, że `media picker` działa w panelu ani że PHP zarejestruje właściwe meta. Każdy poziom testu ma własną granicę i powinien być tak opisany.

`tests/integration.php` uruchamiany przez WP-CLI tworzy fikcyjne materiały `publish`, `draft` i `private` oraz `attachment TXT`. Sprawdza wybrane wartości, zapis z błędnym `nonce`, zapis anonimowy, publiczny odczyt i REST meta. Dane własnego `smoke` są usuwane w `finally`. Skrypt wymaga osobnego stagingu i aktywnej wtyczki.

Wewnętrzny `rest_do_request` korzysta z użytkownika procesu, więc nie zastępuje żądań HTTP z prawdziwym `cookie` oraz `X-WP-Nonce`. Ręczny odbiór musi także objąć MIME PDF i DOCX, niedostępne `attachment`, wpis z hasłem, `autosave` oraz zachowanie edytora blokowego. Nie używaj etykiety „pełne testy” dla częściowej listy.

Brak PHP i Dockera w środowisku przygotowania został zapisany w `TEST_STATUS.md`. Nie deklarujemy `php -l`, aktywacji wtyczki ani testów E2E. Na stagingu uzupełnij dziennik o wersje narzędzi, datę, polecenie, wynik i ewentualny błąd. Jeśli test nie został wykonany, oznacz go jako oczekujący.

Przy błędzie nie usuwaj kontroli bezpieczeństwa, aby uzyskać zielony wynik. Najpierw rozpoznaj źródło: rolę, własność, schemat, ścieżka pliku lub konkretny kontekst zapisu. Poprawka powinna mieć powtarzalny przypadek, który zawiódł przed zmianą i przechodzi po niej.

PROMPT DO CODEXA / Prompt 09 / uruchomienie testów

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przygotuj plan testowania EXAMPLES na czystym stagingu przez `wp-env`. Przeczytaj `.wp-env.json` i oba skrypty `tests`. Oddziel `node --check`, kontrolę JSON, `smoke WP-CLI`, HTTP `auth` oraz ręczny panel. Wskaż skutki seedowania i sprzątania danych. Jeżeli środowisko nie ma PHP lub Dockera, zapisz ograniczenie i instrukcję, bez fikcyjnego wyniku. Przy rzeczywistym błędzie pokaż polecenie, błąd i minimalną poprawkę.

KONTROLA: Raport ma wskazywać wykonane czynności; instrukcja przyszłego testu nie jest jego wynikiem.

PRACOWNIA / PRO 04

Macierz łączy sukces, odmowę i trwałość

Odbiór wtyczki obejmuje zachowanie przy poprawnych danych, błędnym wejściu oraz zmianie środowiska. Jedna ścieżka administratora nie wystarcza.

Dla każdego testu zapisujemy warunek początkowy, działanie, oczekiwany wynik i stan po operacji. Na przykład niepoprawny numer wersji powinien pozostawić wcześniejsze meta, niezależnie od komunikatu panelu. Usunięcie attachment powinno zmienić wynik odczytu na null, ale nie wymazać tytułu materiału.

Próby uprawnień wykonujemy na osobnych kontach stagingowych. Korzystanie cały czas z administratora nie sprawdzi odmowy ani mapowania edit_post dla cudzych rekordów. Przy teście nonce upewnij się, że token pochodzi z właściwego użytkownika i ID, a przy odmowie nie ujawniaj go w raporcie.

Do fixture TXT dodaj ręcznie znany, fikcyjny PDF oraz demonstracyjny DOCX. Wyklucz PNG w polu dokumentu, nieistniejący ID oraz MIME niezgodny z rozszerzeniem. Nie zmieniaj globalnych ustawień uploadu, aby umożliwić przypadkowy niebezpieczny typ. Jeżeli hosting nie dopuszcza dozwolonego formatu, zapisz wynik i sprawdź konfigurację.

Przetestuj także deaktywację, ponowną aktywację i uninstall po kopii. Typ może zniknąć z panelu, ale dane powinny pozostać. Reinstalacja musi ponownie zarejestrować model. Zwróć uwagę na zachowane publiczne uploads i opcjonalne prawa redaktora nadane ręcznie.

Macierz w README jest pełniejsza niż tabela na tej stronie. Wybierz przypadki krytyczne jako bramkę wydania, a resztę przypisz do konkretnego wykonawcy. Żaden pusty wiersz nie powinien być interpretowany jako zaliczony test.

Obszar	Próba	Oczekiwany wynik
Auth	Anonimowy zapis REST	Odmowa, meta bez zmian
Nonce	Błędny lub innego wpisu	Brak aktualizacji pól
Dane	Post zamiast attachment	Odrzucenie ID
MIME	PNG jako dokument	Odrzucenie wyboru
Publiczność	draft / private / hasło	Brak we własnym GET
Trwałość	Uninstall po kopii	Rekordy i pliki pozostają

PRACOWNIA / PRO 04

Wersjonowanie i polityka usuwania

Aktualizacja kodu, zmiana danych i usunięcie wtyczki mają różne skutki. Każda operacja wymaga jawnej, sprawdzalnej procedury.

Pierwsze wydanie nie tworzy własnej tabeli. Znacznik `wdl_library_schema` równy jeden dokumentuje stan, ale nie udaje wykonanej migracji. Aktualizacja plików nie uruchamia ponownie activation hook. Jeżeli następna wersja potrzebuje zmiany danych albo capability, musi posiadać osobny kontrolowany krok, zamiast polegać na reaktywacji przez użytkownika.

Migrator powinien rozpoznawać aktualny schemat, wykonywać niewielkie partie i bezpiecznie obsługiwać ponowienie. Zapisuje postęp i błędy, unika pełnego skanowania na każdym init oraz ma test na kopii starych danych. Sama zmiana default meta nie aktualizuje automatycznie wszystkich historycznych rekordów.

Rollback kodu jest skuteczny tylko wtedy, gdy dane nadal są zgodne z wcześniejszą wersją. Po migracji potrzebny bywa również powrót do właściwej kopii bazy i plików. Przed wydaniem przećwicz odtworzenie, zapisz archiwum wcześniejszego kodu oraz ustal moment wstrzymania nowych zapisów.

Deaktywacja zachowuje dane i odświeża reguły adresów. Uninstall usuwa własne opcje oraz zapisane jako własne nadania administratora, ale zachowuje posty, termy, meta i uploads. Ręcznie przyznane prawa redaktora wymagają osobnego cofnięcia. Pełne kasowanie biblioteki to odrębna decyzja po eksporcie i kopii.

Jeśli inne moduły zaczęły korzystać z tych samych capability, uzgodnij usuwanie przed uninstall. Odpowiedzialność wtyczki nie kończy się na przycisku „Usuń”; trzeba wiedzieć, co pozostaje dostępne i jak przywrócić panel materiałów.

PROMPT DO CODEXA / Prompt 10 / plan wydania

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przygotuj checklistę aktualizacji wp-demo-library 1.0.0 na stagingu: kopia bazy i uploads, odtworzenie, archiwum kodu, testy, ręczna akceptacja i rollback. Wyjaśnij, dlaczego activation hook nie jest migratorem każdej aktualizacji. Opisz zachowanie deaktywacji i uninstall na podstawie plików. Jeśli proponujesz przyszłą migrację, podaj jej idempotencję, partie, dziennik i kompatybilność wsteczną. Nie usuwaj rekordów ani plików w ramach przeglądu.

KONTROLA: Procedura musi rozróżniać cofnięcie kodu od cofnięcia danych i wskazywać zachowane publiczne uploads.

Odbiór zaczyna się od dowodów

Wydanie przyjmujemy na podstawie zgodności kontraktu, raportu testów i sprawdzonego odtworzenia. Samo wygenerowanie kompletnego katalogu nie zamyka pracy.

Osoba odbierająca powinna otrzymać kompletne EXAMPLES, opis pól i uprawnień, instrukcję instalacji, macierz testów oraz status wykonania. Dzięki temu nie musi odgadywać, które fragmenty są demonstracją, a które realnym kodem. W książce każda próbka wskazuje pełny plik źródłowy i jawny status testowania.

Najpierw porównujemy nazwy i reguły z kontraktem serii. Następnie uruchamiamy staging, seedujemy dane fikcyjne i wykonujemy smoke oraz ręczne przypadki. Krytyczne odmowy obejmują brak capability, błędny nonce, niedozwolony MIME i prywatny status. Publiczność uploads sprawdzamy świadomie, zamiast traktować działający URL szkicu jako usterkę rekordu.

Kolejny krok to deaktywacja, ponowna aktywacja i odtworzenie kopii. Na czystym środowisku testujemy uninstall bez usuwania danych. Zanotuj, które capabilities nadała wtyczka, a które administrator przyznał później ręcznie. Takie informacje pozwalają uniknąć zaskoczenia podczas wycofywania modułu.

Akceptacja człowieka obejmuje również treść dokumentów i ich publiczny charakter. Żaden test składni nie potwierdza, że PDF można udostępnić. Nie automatyzujemy publikacji tylko dlatego, że plik przeszedł kontrolę typu. W ramach laboratorium publikujemy wyłącznie znane materiały fikcyjne.

Diagram opisuje ścieżkę wydania po zakończeniu implementacji. Każdy etap ma wynik do przeczytania: zgodność, test, kopię albo decyzję. Brak wyniku oznacza zadanie oczekujące, nie cichą zgodę. Wdrożenie na prawdziwą witrynę wymaga dostosowania projektu i własnego odbioru środowiska.

Bramki przyjęcia wydania

01 Kontrakt

Nazwy, pola, prawa i publiczność.

02 Staging

Rzeczywiste wersje oraz fikcyjne dane.

03 Testy

Sukcesy, odmowy, panel i HTTP.

04 Odtworzenie

Przećwiczona kopia i rollback.

05 Akceptacja

Człowiek zatwierdza wydanie i materiały.

PRACOWNIA / PRO 04

Codex przygotowuje przekazanie projektu

Końcowy raport powinien pomagać następnej osobie wykonać pracę, a nie budować przekonanie, że wszystko zostało przetestowane bez dowodu.

Poproś Codexa o raport oparty na plikach i wynikach poleceń. Ma wymienić model treści, ścieżki zapisu, granice publicznego odczytu oraz politykę uninstall. Każde stwierdzenie o teście powinno posiadać polecenie, środowisko i wynik. Jeżeli narzędzie jest niedostępne, potrzebna jest instrukcja do wykonania i widoczny status oczekujący.

W naszym pakiecie lokalne sprawdzenie JavaScript i JSON jest rzeczywistym wynikiem. PHP, aktywacja WordPressa, smoke i testy edytora pozostają do uruchomienia. Nie zamieniamy tego ograniczenia w drobną uwagę ukrytą na końcu. Osoba odbierająca powinna wiedzieć przed instalacją, co już sprawdzono, a co wymaga jej środowiska.

Przekazanie obejmuje również procedurę obsługi błędów. Kiedy użytkownik zgłasza niezapisany plik, sprawdzamy ID, typ rekordu, istnienie lokalnego pliku, MIME, capability materiału oraz attachment. Nie zaczynamy od dodania szerokich uprawnień lub dopuszczenia wszystkich typów uploadu.

W raporcie nie powinno być prawdziwego hosta klienta, hasła aplikacyjnego, nonce, cookie ani treści poufnego dokumentu. Fikcyjne nazwy i identyfikatory stagingowe wystarczą do odtworzenia błędu. Tytuł „przetestowana wtyczka” jest uzasadniony dopiero po wykonaniu wymaganej macierzy w opisanym środowisku.

PROMPT DO CODEXA / Prompt 11 / raport przekazania

Skopiuj do Codexa razem z kompletnym katalogiem EXAMPLES.

Przygotuj raport odbiorowy wp-demo-library wyłącznie na podstawie EXAMPLES i dostępnego dziennika. Podaj pliki, model danych, publiczny endpoint, kontrolę panelu i REST, politykę danych po uninstall oraz wymagania stagingu. Utwórz trzy jawne grupy: testy wykonane, testy niewykonane i warunki przed wdrożeniem. Przy wynikach podaj polecenia. Nie deklaruj php -l ani aktywacji, jeśli brak dowodu. Dodaj procedurę diagnozy błędnego attachment bez poszerzania uprawnień.

KONTROLA: Raport musi być zrozumiały dla osoby, która nie uczestniczyła w generowaniu kodu.

PRACOWNIA / PRO 04

Następny krok: wiedza z publicznej biblioteki

Zaprojektowana wtyczka stanowi małe źródło uporządkowanych materiałów. Kolejny tom wykorzystuje ten model, ale rozdziela publiczne treści od prywatnego magazynu.

Masz już czytelny typ materiału, tematy, wersję dokumentu i ID załącznika. Taki kontrakt pozwala innemu modułowi odczytać opublikowane materiały oraz zdecydować, co trzeba ponownie przetworzyć po zmianie wersji. Nie trzeba znać układu formularza redaktora, aby rozumieć dane.

Przyszły indeks wiedzy powinien śledzić identyfikator i wersję dokumentu oraz wycofywać stare fragmenty po usunięciu lub zastąpieniu materiału. Sama zmiana nazwy pliku nie jest wystarczającą strategią synchronizacji. Wersja musi odpowiadać zatwierdzonej zmianie treści, a proces powinien wykrywać brak pliku i niejednoznaczne wyniki aktualizacji.

Publiczne materiały mogą być źródłem odpowiedzi, lecz prywatne dokumenty wymagają osobnego ACL i magazynu poza publicznym uploads. Endpoint ask z następnego tomu korzysta z uwierzytelnienia i capability oraz serwerowej integracji. Nie dokładaj sekretu modelu do admin.js i nie przenoś kontroli grup do parametru przeglądarki.

Przed rozbudową zachowaj stabilność tej małej biblioteki. Dopisz test do każdego nowego pola lub sposobu zapisu, uaktualnij fixture i README oraz sprawdź wycofanie publikacji. Rozszerzenie powinno korzystać z opisanego kontraktu, zamiast omijać go bezpośrednim, niekontrolowanym zapisem meta.

Ostatnim zadaniem warsztatu jest samodzielne przeprowadzenie odbioru na stagingu. Utwórz dwa fikcyjne materiały, sprawdź sukces i odmowę, przeczytaj publiczny katalog, a następnie wyłącz i przywróć wtyczkę. Zapisz dowody oraz nierozwiązane kwestie. Dopiero taki komplet pokazuje, że rozumiesz własną funkcję, jej dane, ograniczenia oraz rzeczywiste granice działania.

DOKUMENTACJA / 13.10.2026

Źródła i dalsze sprawdzanie

Oficjalne źródła wykorzystane do minimalnego sprawdzenia API. Odczytaj odpowiednią stronę ponownie przed dostosowaniem przykładu do swojego wydania WordPressa.

Plugin Handbook

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

Nonces

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

Roles and Capabilities

<https://developer.wordpress.org/plugins/users/roles-and-capabilities/>

register_post_type()

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

register_post_meta()

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

register_meta()

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

Adding Custom Endpoints

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

Data Validation

<https://developer.wordpress.org/plugins/security/data-validation/>

wp_get_attachment_url()

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

wpdb::prepare()

<https://developer.wordpress.org/reference/classes/wpdb/prepare/>

@wordpress/env

<https://developer.wordpress.org/block-editor/reference-guides/packages/packages-env/>

media_handle_sideload()

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