Utrzymanie systemu

Cel dokumentacji
Dokument określa procedury, standardy i zasady dotyczące utrzymania, serwisowania, rozbudowy i monitorowania systemu. Niniejszy dokument opisuje procedury utrzymania aplikacji webowej służącej do zarządzania procesami magazynowymi i transportowymi. Aplikacja umożliwia efektywne zarządzanie przepływem towarów w magazynach, ich przyjęciem, wydaniem oraz śledzeniem transportu do różnych lokalizacji. W dokumentacji znajdują się instrukcje dotyczące utrzymania systemu, zarządzania danymi, aktualizacji oraz rozwiązywania najczęstszych problemów.

Zarządzanie Wersjami i Cykl Rozwoju (Rozbudowa Systemu)
Proces rozwoju CargoSmart opiera się na metodyce ciągłej integracji/ciągłego wdrażania (CI/CD) oraz wersjonowaniu semantycznym.

Wersjonowanie Semantyczne (SemVer)
System CargoSmart używa formatu wersji X.Y.Z:
X (Major): Duże zmiany, które zrywają z kompatybilnością wsteczną (np. przebudowa modułu transportu, wymagająca zmian w integracjach zewnętrznych).
Y (Minor): Nowe funkcjonalności, większe pakiety poprawek, które są kompatybilne wstecznie (np. dodanie nowego raportu, optymalizacja algorytmu trasowania).
Z (Patch): Nieznaczące zmiany, drobne błędy, literówki (np. poprawa etykiety w interfejsie użytkownika).
Wersje 0.y.z: W fazie początkowego rozwoju (przed 1.0.0), wszelkie zmiany są dopuszczalne.

Procedura Podnoszenia Wersji (Release Procedure)
Nowe wersje są wydawane cyklicznie co 4 tygodnie (stały okres, wzorowany na cyklach wydań popularnych aplikacji, np. VS Code co miesiąc), z wyjątkiem krytycznych poprawek błędów, które mogą być wydane natychmiast.

Release Notes (Informacje o wydaniu)
Każde wydanie musi zawierać notatki release notes dokumentujące:
Zmienione funkcjonalności.
Naprawione błędy.
Zerwanie kompatybilności wstecznej (jeśli dotyczy, tylko w wersjach Major).
Zmiany w zależnościach zewnętrznych (np. minimalna wymagana wersja przeglądarki Chrome/Edge, używana wersja biblioteki Python/Java).

Kompatybilność Wsteczna
Zasada: Dążymy do utrzymania kompatybilności wstecznej w wersjach Minor (Y) i Patch (Z), aby minimalizować koszt aktualizacji po stronie klienta (brak konieczności angażowania zespołu klienta, sprzętu czy rekonfiguracji integracji).
Wyjątki: Świadome zerwanie kompatybilności (wersja Major X) jest dopuszczalne, gdy utrzymanie jej blokuje rozwój jakościowy, skalowalność lub powoduje nadmierny wzrost kosztów utrzymania obecnej architektury.

Serwisowanie Systemu (Obsługa Błędów i Zgłoszeń)
Zdefiniowano kategorie błędów oraz procedury serwisowe zgodnie z umowami SLA.

Kategorie Błędów i Czas Reakcji (SLA)			
Kategoria Błędu	**Opis**	**Priorytet**	**Docelowy Czas Usunięcia**
Krytyczny	Aplikacja niedostępna / blokada procesów magazynowych	P1	4 godziny (24/7)
Wysoki	Częściowa niedostępność / poważne błędy w funkcjonalności	P2	1 dzień roboczy
Średni	Błędy wpływające na wygodę, ale z obejściem (workaround)	P3	Następne wydanie Minor (4 tyg)
Niski	Literówki, błędy UI/UX	P4	Według harmonogramu rozwoju

Procedura Zgłaszania i Obsługi Błędu
Zgłoszenie błędu: Można zgłaszać błędy poprzez:
Formularz zgłoszeniowy wewnątrz aplikacji (moduł "Pomoc").
Obsługa: Zgłoszenia trafiają do systemu zarządzania projektami, gdzie są kategoryzowane i przypisywane do zespołu.
Procedura aktualizacji po usunięciu błędu:
Informacja o poprawce trafia do release notes.
W przypadku błędu krytycznego (P1/P2), zgłaszający jest informowany bezpośrednio przez dedykowanego konsultanta o wdrożeniu poprawki.

Skalowanie Infrastruktury
Plan skalowania ma na celu sprostanie rosnącym potrzebom biznesowym (wzrost liczby użytkowników, złożoności danych).

Strategie Skalowania
Skalowanie Horyzontalne (więcej serwerów/instancji): Preferowana metoda skalowania warstwy aplikacyjnej (stateless application servers).
Skalowanie Wertykalne (więcej RAMu/CPU): Stosowane w pierwszej kolejności dla dedykowanych serwerów baz danych, zanim przejdziemy do bardziej złożonych rozwiązań (replikacja, sharding).

Optymalizacja Składowych Systemu
Cache: Implementacja cache'owania (Redis/Memcached) na poziomie frontendu, backendu i zapytań do bazy danych w celu redukcji obciążenia.
Baza Danych: Ciągła optymalizacja zapytań SQL, rozważenie zmiany źródła lub rodzaju danych (np. migracja części danych analitycznych do hurtowni danych).
Architektura: Utrzymanie warstwy logiki aplikacji w sposób modułowy, ułatwiający przyszłą mikroserwisową architekturę, co ułatwi niezależne skalowanie poszczególnych funkcji (np. moduł śledzenia GPS, moduł fakturowania).

Metodologia

Projekt będzie realizowany zgodnie z podejściem Agile, co pozwoli na elastyczne dostosowywanie funkcjonalności systemu do zmieniających się wymagań użytkowników i potrzeb rynku. Proces tworzenia aplikacji będzie składał się z kilku iteracji, które obejmują:

Analizę wymagań – zebranie szczegółowych informacji na temat potrzeb użytkowników oraz oczekiwań względem systemu.

Projektowanie architektury systemu – zaprojektowanie struktury bazy danych, interfejsu użytkownika oraz integracji z innymi systemami.

Implementację i testowanie – kodowanie poszczególnych modułów systemu oraz przeprowadzanie testów funkcjonalnych i wydajnościowych.

Wdrożenie i szkolenie – implementacja aplikacji w środowisku produkcyjnym oraz przeprowadzenie szkoleń dla użytkowników końcowych.

Utrzymanie i aktualizacje – zapewnienie wsparcia po wdrożeniu oraz rozwój aplikacji o dodatkowe funkcjonalności.

Podsumowanie

Utrzymanie systemu zarządzania procesami magazynowymi i transportowymi wymaga dbałości o wysoką dostępność, bezpieczeństwo oraz wydajność aplikacji. Kluczowe procedury obejmują regularne aktualizacje, monitoring aplikacji i infrastruktury, zarządzanie danymi oraz szybkie reagowanie na awarie. Dzięki skutecznemu utrzymaniu systemu użytkownicy mogą cieszyć się nieprzerwaną, efektywną obsługą procesów magazynowych i transportowych.