

Contents

Primus Backend	1
Architektura	1
Kluczowe komponenty:	2
Realizacja Wymagań Specyfikacji	2
1. Definiowanie Magazynu (api/v1/endpoints/rack_CRUD.py)	2
2. Definiowanie Asortymentu (api/v1/endpoints/ product_definition_CRUD.py)	2
3. Przyjmowanie Asortymentu (Inbound)	2
4. Zdejmowanie Asortymentu (Outbound - FIFO)	2
5. Monitorowanie i Alerty (IoT & Tasks)	3
6. Wizualizacja	3
7. AI i Voice (api/v1/endpoints/ai.py, api/v1/endpoints/voice.py)	3
8. Raportowanie (api/v1/endpoints/reports.py)	3
9. Backupy i Bezpieczeństwo (api/v1/endpoints/backups.py)	3
10. Użytkownicy i Uprawnienia	3
Stos Technologiczny	4
Uruchomienie	4
Docker (Zalecane)	4
Lokalnie (Development)	4
Testy Wydajnościowe	5
Uruchomienie testów	5
Scenariusze testowe	5

Primus Backend

Backend systemu **Primus Inter Pares 2026** – serce inteligentnego systemu zarządzania magazynem (WMS). Aplikacja realizuje logikę biznesową, zarządza stanem magazynowym zgodnie z zasadą FIFO, integruje usługi AI/IoT oraz zapewnia bezpieczeństwo danych.

Pełna dokumentacja projektu: Primus Docs

Dokumentacja API (Swagger): Github pages (lub dostępna lokalnie po uruchomieniu w trybie local /docs)

Architektura

System został zaprojektowany jako **Modularny Monolit** z elementami architektury sterowanej zdarzeniami (Event-Driven) do obsługi IoT i zadań w tle.

Kluczowe komponenty:

- **API Layer (FastAPI):** Obsługa żądań HTTP, walidacja danych (Pydantic), autoryzacja (OAuth2).
 - **Service Layer:** Izolowana logika biznesowa (np. AllocationService, StockService, ReportService).
 - **Worker (Celery):** Asynchroniczne przetwarzanie zadań (generowanie raportów PDF, backupy, trening AI).
 - **IoT Integration (MQTT/Redis):** Odbiór i analiza danych z sensorów w czasie rzeczywistym.
-

Realizacja Wymagań Specyfikacji

Backend realizuje wszystkie punkty wyszczególnione w regulaminie zawodów:

1. Definiowanie Magazynu (`api/v1/endpoints/rack_CRUD.py`)

- **Model Regału:** Obsługa regałów o wymiarach MxN (elementy).
- **Parametry:** Przechowywanie limitów temperatury (min/max), wagi (kg) oraz wymiarów (mm) dla każdego regału.
- **Import CSV:** Masowe dodawanie regałów z plików CSV.

2. Definiowanie Asortymentu (`api/v1/endpoints/product_definition_CRUD.py`)

- **Karta Produktu:** Nazwa, kod kreskowy/QR, zdjęcie, wymogi temperaturowe, waga, wymiary, termin ważności (dni).
- **Flagi Specjalne:** Obsługa produktów niebezpiecznych (ADR) i kruchych.
- **Import CSV:** Szybkie wprowadzanie katalogu produktów.

3. Przyjmowanie Asortymentu (Inbound)

- **Inteligentna Alokacja (AllocationService):**
 - Algorytm automatycznie wyszukuje najlepsze miejsce składowania.
 - **Kryteria:** Zgodność temperatur, wymiary slotu vs produktu, limit udźwigu regału.
 - **Walidacja:** Blokada umieszczenia produktu, jeśli żaden regał nie spełnia wymogów (zgodnie z specyfikacją).
- **Rejestracja:** Zapis czasu przyjęcia i operatora.

4. Zdejmowanie Asortymentu (Outbound - FIFO)

- **Algorytm FIFO (StockService):** System wymusza wydanie najstarszej partii danego asortymentu (sortowanie po `entry_date`).
- **Blokada:** Nie pozwala na wydanie nowszego towaru, jeśli starszy jest dostępny (chyba że w trybie awaryjnym administratora).

5. Monitorowanie i Alerty (IoT & Tasks)

- **Terminy Ważności:**

- *Warning*: Automatyczne powiadomienia o zbliżającym się końcu ważności (AlertService).
- *Expired*: Blokada wydania i alert dla produktów przeterminowanych.

- **Monitoring Środowiskowy (IoT):**

- Integracja z czujnikami temperatury i wagi (przez MQTT).
- Wykrywanie anomalii: Przekroczenie zakresu temperatur dla danego re- gału lub niezgodność wagi (kradzież).

6. Wizualizacja

- Backend dostarcza strukturę danych (JSON) reprezentującą siatkę magazynu (MxN) wraz ze stanem zajętości slotów, co pozwala frontendowi na ren- derowanie interaktywnej mapy.

7. AI i Voice (api/v1/endpoints/ai.py, api/v1/endpoints/voice.py)

- **Rozpoznawanie Obrazu (YOLO)**: Identyfikacja produktów na podstawie zdjęć (np. uszkodzony kod kreskowy). Backend obsługuje trenowanie modelu na nowych danych (Continuous Learning).
- **Asystent Głosowy (LLM)**: Przetwarzanie języka naturalnego (NLP) do sterowania systemem (np. "Zdejmij Mleko", "Przyjmij Mleko", "Wygeneruj raport").

8. Raportowanie (api/v1/endpoints/reports.py)

- Generowanie plików **PDF** w tle (Celery) (ReportService):
 - Raport kończących się dat ważności.
 - Raport przekroczeń temperatur (historia incydentów).
 - Pełna inwentaryzacja (Audit Log).

9. Backupy i Bezpieczeństwo (api/v1/endpoints/backups.py)

- **Szyfrowane Backupy**: Zrzuty bazy danych są szyfrowane algorytmem **AES-128 (Fernet)** przed zapisem (BackupService).
- **Harmonogram**: Automatyczne tworzenie kopii (np. codziennie w nocy).
- **Odzyskiwanie**: Możliwość przywrócenia stanu magazynu z wybranego pliku backupu.

10. Użytkownicy i Uprawnienia

- **Role:**

- **Administrator**: Pełny dostęp, zarządzanie magazynem i użytkownikami.
- **Magazynier**: Operacje przyjęcia/wydania, podgląd stanu.

- **Bezpieczeństwo:** Hasła hashowane (bcrypt), uwierzytelnianie **OAuth2 + JWT**.
- **2FA:** Obsługa dwuskładnikowego uwierzytelniania (TOTP - Google Authenticator).

Stos Technologiczny

Kategoria	Technologia	Zastosowanie
Język	Python 3.11	Główny język aplikacji
Framework	FastAPI	Wydajne, asynchroniczne API
Baza Danych	PostgreSQL 15	Przechowywanie danych relacyjnych
ORM	SQLAlchemy 2.0 (Async)	Mapowanie obiektowo-relacyjne
Migracje	Alembic	Zarządzanie schematem bazy danych
Zadania w tle	Celery + Redis	Raporty, backupy, AI
AI/ML	PyTorch + Ultralytics	Detekcja obiektów (YOLOv8)
IoT	Paho MQTT	Komunikacja z sensorami

Uruchomienie

Docker (Zalecane)

Cała infrastruktura jest definiowana w repozytorium primus-infra. Aby uruchomić backend w kontenerze:

```
docker compose up -d backend
```

Lokalnie (Development)

Wymagane: Python 3.11+, Poetry

1. Instalacja zależności:

```
poetry install
```

2. Uruchomienie serwera deweloperskiego:

```
poetry run uvicorn app.main:app --reload
```

API dostępne pod adresem: <http://localhost:8000>

Testy Wydajnościowe

Projekt zawiera scenariusze testowe oparte o narzędzie **Locust**, pozwalające na symulację dużego obciążenia zgodnie z wymaganiami konkursowymi.

Uruchomienie testów

```
poetry run locust -f scripts/locustfile.py
```

Panel sterowania dostępny pod: <http://localhost:8089> Or directly inspect the locustfile.py.

Scenariusze testowe

1. **Import Masowy (CSV):** Test wydajności wczytywania 5000+ definicji produktów.
2. **Obciążenie Alokacji:** Symulacja równoległych przyjęć towaru przez wielu magazynierów.
3. **Raportowanie:** Generowanie ciężkich raportów PDF przy dużym obciążeniu bazy.
4. **AI Queue:** Kolejowanie zadań rozpoznawania obrazu.