Dokumentacja Projektowa: PrintFlow MIS

1. Wizja i Cel Systemu

System ma zastąpić rozproszone arkusze Google Docs dedykowaną aplikacją webową, która automatyzuje proces wyceny produktów standardowych i indywidualnych (*ad hoc*). Kluczowym celem jest **absolutna efektywność zespołu sprzedażowego** poprzez skrócenie czasu wyceny do minimum przy jednoczesnym zachowaniu precyzji technologicznej (zysk, odpady, procesy).

2. Architektura Funkcjonalna

A. Biblioteka Zasobów (The Core)

To fundament systemu, który pozwala na "atomizację" produkcji.

- **Zarządzanie Materiałami:**
    - Obsługa wariantów w ramach jednego typu (np. "Folia stalowa" dostępna w szerokościach 100 cm i 137 cm).
    - Atrybuty: cena zakupu, narzut %, waga, typ odpadu (procentowy lub wynikający z szerokości roli).
- Biblioteka Procesów:
    - Definiowanie operacji produkcyjnych z różną logiką kosztową:
        - Powierzchniowy (m^2): np. laminowanie, lakierowanie.
        - Liniowy (mb): np. cięcie ploterem (długość linii cięcia), frezowanie.
        - Czasowy (h): np. praca grafika, montaż.
        - Sztukowy (pcs): np. pakowanie, konfekcja.
    - Możliwość definiowania "kosztu narządu" (setup fee) dla każdego procesu.

B. Szablony i Konfigurator Produktów

System rozróżnia dwa tryby tworzenia produktów:

1. Produkty Standardowe (Szablony):
    - Gotowe "przepisy" na produkty (np. Tablica Kredowa).
    - Podwarianty i Opcje: Możliwość zaznaczenia dodatków (np. "Opcja magnetyczna"). Zaznaczenie opcji automatycznie wstrzykuje do wyceny odpowiednie materiały (blacha) i procesy (klejenie).
    - Użytkownik podaje tylko wymiary i klika opcje, system liczy resztę.

2. Produkty Ad Hoc:
    - Tworzenie wyceny od zera poprzez wybieranie komponentów z biblioteki.
    - Zapisywanie "przepisu" w bazie wycen, aby umożliwić powtórzenie zamówienia w przyszłości.

C. Silnik Kalkulacyjny (The Brain)

Logika systemu musi automatycznie rozwiązywać problemy, które wcześniej wymagały uwagi człowieka:

- Algorytm Best-Fit:
  Dla produktu o wymiarach $W \times H$, system analizuje dostępne szerokości materiału ($S_1, S_2, ...$). Wybiera tę, która minimalizuje odpad:
  $$\text{Min}(S_i \times \text{Długość}) - (W \times H)$$
- Obliczanie Procesów Liniowych:
  Automatyczne wyliczanie długości linii cięcia na podstawie obwodu (prostokąt) lub importowanej ścieżki (w przyszłości).
- Transparentność Marży:
  Operator widzi koszt wytworzenia (COGS), narzut i zysk końcowy w czasie rzeczywistym.

3. Psychologia Sprzedaży i Marketing Oferty

Oferta generowana przez system (PDF/Email) musi budować zaufanie i domykać sprzedaż:

- Dualizm Widoku:
    - Widok Klienta: Estetyczny opis, wizualizacja opcji, cena końcowa, termin realizacji (Lead Time) oraz profesjonalne CTA.
    - Widok Techniczny: Lista materiałów, konkretne warianty rolek, czasy procesów i uwagi dla produkcji.
- Integracja Gmail:
    - Wysyłanie ofert bezpośrednio z aplikacji.
    - Automatyczny Follow-up: Jeśli klient nie odpowie w ciągu X dni, system wysyła spersonalizowane przypomnienie.
    - Tracking: Powiadomienie handlowca, gdy klient otworzy PDF lub zaakceptuje ofertę.

4. Architektura Techniczna (Specyfikacja)

Stack Technologiczny

- Backend: FastAPI (Python) – doskonały do logiki matematycznej i integracji AI.
- Frontend: Next.js + Tailwind CSS + shadcn/ui – responsywny, szybki interfejs.
- Baza Danych: PostgreSQL.
- Serwer: Ubuntu

```
None

Materials
ID, Nazwa, Kategoria


MaterialVariants
Material_ID, Szerokość, Cena_Zakupu, Narzut


Processes
ID, Nazwa, Metoda_Obliczania (m2, mb, h, szt), Cena


ProductTemplates
ID, Nazwa, Składniki_Bazowe (JSON)


TemplateOptions
ID, Template_ID, Dodatkowe_Komponenty (JSON)


Quotes
ID, Client_ID, Status, Lead_Time, Total_Price, Recipe_Snapshot (JSON)
```

5. Funkcjonalności AI (Efektywność 2.0)

Wdrożenie AI w systemie ma na celu wsparcie handlowca:

1. **AI Sentiment Analysis:** Skanowanie maili zwrotnych. Jeśli klient pisze "szukam czegoś tańszego", AI sugeruje handlowcowi zmianę materiału na tańszy wariant w danej wycenie.

2. **Smart Descriptions:** Automatyczne generowanie opisów produktów do PDF na podstawie wybranych komponentów, aby każda oferta wyglądała na unikalną.
3. **Wsparcie Ad Hoc:** Na podstawie opisu tekstowego handlowca (np. "tablica z ramką aluminiową 100x100"), AI dobiera pasujące materiały i procesy z biblioteki.

Opis relacji:

- **Template_Option:** Przechowuje informacje o podwariantach (np. "Magnetyczność"). Każda opcja jest powiązana z listą komponentów, które dodaje do wyceny.
- **Quote_Component:** To "snapshot". Nawet jeśli zmienisz cenę folii w bibliotece za rok, wycena archiwalna zachowa cenę z dnia jej utworzenia.

```sql
SQL
-- ======================================================================
-- DATABASE SCHEMA: PrintFlow MIS
-- Unified DDL with Technology Margins & Multi-level Calculation Logic
-- ======================================================================

-- 1. Typy wyliczeniowe (Enums)
CREATE TYPE calculation_method AS ENUM ('AREA', 'LINEAR', 'TIME', 'UNIT');
CREATE TYPE quote_status AS ENUM ('DRAFT', 'SENT', 'ACCEPTED', 'REJECTED',
'COMPLETED');

-- 2. Biblioteka Materiałów (Materials & Variants)
CREATE TABLE materials (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    category VARCHAR(100),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE material_variants (
    id SERIAL PRIMARY KEY,
    material_id INTEGER REFERENCES materials(id) ON DELETE CASCADE,
    width_cm DECIMAL(10, 2),        -- Szerokość rolki/arkusza
```

```
    length_cm DECIMAL(10, 2),       -- Długość (dla arkuszy lub max nawoju)
    cost_price_per_unit DECIMAL(10, 2) NOT NULL,
    markup_percentage DECIMAL(5, 2) DEFAULT 0.00,
    unit VARCHAR(10) NOT NULL,      -- 'm2', 'mb', 'pcs'

    -- Naddatki technologiczne materiału (np. margines na chwytaki)
    margin_w_cm DECIMAL(10, 2) DEFAULT 0.0,
    margin_h_cm DECIMAL(10, 2) DEFAULT 0.0,

    is_active BOOLEAN DEFAULT TRUE
);

-- 3. Biblioteka Procesów (Processes)
CREATE TABLE processes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    method calculation_method NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    setup_fee DECIMAL(10, 2) DEFAULT 0.00, -- Koszt narządu (startowy)
    internal_cost DECIMAL(10, 2),          -- Koszt własny dla analityki

    -- Naddatki technologiczne procesu (np. margines bezpieczeństwa dla frezu)
    margin_w_cm DECIMAL(10, 2) DEFAULT 0.0,
    margin_h_cm DECIMAL(10, 2) DEFAULT 0.0
);

-- 4. Szablony Produktów i Opcje (Product Templates & Options)
CREATE TABLE product_templates (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,

    -- Domyślne naddatki produktu (np. spady drukarskie na docięcie)
    default_margin_w_cm DECIMAL(10, 2) DEFAULT 0.0,
    default_margin_h_cm DECIMAL(10, 2) DEFAULT 0.0
    default_overlap_cm DECIMAL(10, 2) DEFAULT 1.0
);

CREATE TABLE template_components (
    id SERIAL PRIMARY KEY,
    template_id INTEGER REFERENCES product_templates(id) ON DELETE CASCADE,
    material_id INTEGER REFERENCES materials(id),
    process_id INTEGER REFERENCES processes(id),
```

```
    is_required BOOLEAN DEFAULT TRUE,    -- TRUE = baza, FALSE =
opcja/podwariant
    group_name VARCHAR(100),             -- Np. "Podkład", "Wykończenie"
    option_label VARCHAR(255),           -- Nazwa widoczna dla usera (np.
"Magnetyczna")
    default_quantity_formula VARCHAR(255) -- Np. "W*H", "2*(W+H)"
);

-- 5. Klienci i System Sprzedażowy
CREATE TABLE clients (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(50),
    company_details TEXT
);

CREATE TABLE quotes (
    id SERIAL PRIMARY KEY,
    client_id INTEGER REFERENCES clients(id),
    user_id INTEGER,                     -- ID handlowca (z systemu auth)
    status quote_status DEFAULT 'DRAFT',
    lead_time_raw VARCHAR(255),          -- Wpisane ręcznie (np. "Do czwartku")
    total_price_net DECIMAL(12, 2),
    margin_value DECIMAL(12, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 6. Pozycje Wyceny i Snaphoty (Archiwum techniczne)
CREATE TABLE quote_items (
    id SERIAL PRIMARY KEY,
    quote_id INTEGER REFERENCES quotes(id) ON DELETE CASCADE,
    product_name VARCHAR(255),           -- Nazwa szablonu lub "Produkt Ad Hoc"
    width_cm DECIMAL(10, 2),             -- Wymiar netto podany przez klienta
    height_cm DECIMAL(10, 2),            -- Wymiar netto podany przez klienta
    quantity INTEGER DEFAULT 1
);

CREATE TABLE quote_components (
    id SERIAL PRIMARY KEY,
    quote_item_id INTEGER REFERENCES quote_items(id) ON DELETE CASCADE,

    -- Linki do bazy (opcjonalne, bo mamy snapshoty)
```

```
    variant_id INTEGER REFERENCES material_variants(id),
    process_id INTEGER REFERENCES processes(id),

    -- SNAPSHOTY (ceny i nazwy z momentu wyceny)
    name_snapshot VARCHAR(255),          -- Np. "Folia Monomer (Rolka 137cm)"
    calculated_quantity DECIMAL(12, 4),  -- Ilość uwzględniająca naddatki i
odpady
    unit_price_snapshot DECIMAL(10, 2),  -- Cena sprzedaży za jednostkę
    total_price DECIMAL(10, 2),          -- Suma za ten komponent

    -- Flagi logiczne
    is_from_option BOOLEAN DEFAULT FALSE,
    tech_margin_applied_w DECIMAL(10, 2), -- Zastosowany naddatek W dla tej
pozycji
    tech_margin_applied_h DECIMAL(10, 2)  -- Zastosowany naddatek H dla tej
pozycji
);

-- Indeksy dla wydajności
CREATE INDEX idx_quote_client ON quotes(client_id);
CREATE INDEX idx_quote_status ON quotes(status);
CREATE INDEX idx_variant_material ON material_variants(material_id);
```

1. **Tabela `material_variants`**: Pozwala przechowywać tę samą folię w różnych szerokościach. Algorytm Best-Fit będzie przeszukiwał tę tabelę, filtrując po `material_id` i `width_cm`.
2. **Tabela `template_components` z `is_required`**: To tutaj realizujemy Twoją potrzebę podwariantów. Jeśli komponent ma `is_required = FALSE`, w UI pojawi się jako checkbox (np. "Opcja magnetyczna").
3. **Snapshoty w `quote_components`**: To najważniejszy element "pamięci" systemu. Jeśli klient wróci po roku do wyceny *Ad Hoc*, system dokładnie wie, jakiej ceny i jakich materiałów wtedy użyłeś, nawet jeśli w międzyczasie zmieniłeś globalny cennik.
4. **Logika `calculated_quantity`**: Pole to przechowuje faktyczne zużycie (np. metry bieżące po uwzględnieniu szerokości rolki), co jest kluczowe dla karty technicznej dla produkcji.

5. **Hierarchiczne Naddatki**: Pola `margin_w_cm` i `margin_h_cm` znajdują się w `material_variants`, `processes` oraz `product_templates`. Dzięki temu system może wybrać najwyższą wartość lub zsumować je według Twoich reguł.
6. **Snapshoty Naddatków**: W tabeli `quote_components` dodałem pola `tech_margin_applied`, które zapisują, jaki naddatek został faktycznie doliczony. Jest to bezcenne przy produkcji (operator widzi: "Dostałem 2 cm zapasu").
7. **Elastyczność Opcji**: Tabela `template_components` pozwala na tworzenie grup opcji. Możesz tam zdefiniować, że "Płyta MDF" i "Płyta PCV" to opcje w tej samej grupie (klient wybiera jedną), a "Warstwa Magnetyczna" to opcja dodatkowa.
8. **Logika Jednostek**: `material_variants` przechowuje cenę i jednostkę (m2/mb/pcs), co pozwala na precyzyjne rozliczanie zarówno folii z rolki, jak i usług takich jak oczkowanie czy pakowanie.

Oto jednolity, profesjonalny kod w języku Python (FastAPI), który stanowi logiczne serce systemu PrintFlow MIS. Kod ten implementuje zaawansowany silnik kalkulacyjny uwzględniający naddatki technologiczne na trzech poziomach (Produkt, Materiał, Proces) oraz algorytm Best-Fit dla optymalizacji zużycia materiału.

Python

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Dict, Optional, Any
from decimal import Decimal, ROUND_HALF_UP
import math

# --- MODELE KOMUNIKACJI API ---

class CalculationRequest(BaseModel):
    width_cm: float
    height_cm: float
    quantity: int
    template_id: Optional[int] = None
    selected_options: List[int] = []
    # Możliwość nadpisania zakładki z UI (jeśli None -> bierzemy z szablonu)
    overlap_override_cm: Optional[float] = None
```

```python
class ComponentResult(BaseModel):
    name: str
    type: str  # 'MATERIAL' | 'PROCESS'
    qty: float
    unit: str
    price_net: float
    details: str

class CalculationResponse(BaseModel):
    total_price_net: float
    total_cost_cogs: float
    margin_percentage: float
    gross_dimensions: Dict[str, float]
    is_split: bool
    num_panels: int
    overlap_used_cm: float
    client_view: List[Dict[str, Any]]
    tech_view: List[ComponentResult]

# --- RDZEŃ SILNIKA KALKULACYJNEGO ---

class PrintFlowEngine:
    def __init__(self, db_context: Dict):
        self.db = db_context

    def _q(self, val: Any) -> Decimal:
        """Konwersja na Decimal dla precyzji finansowej i technicznej."""
        return Decimal(str(val)).quantize(Decimal("0.0001"),
rounding=ROUND_HALF_UP)

    def _money(self, val: Decimal) -> float:
        """Finalne zaokrąglenie do groszy."""
        return float(val.quantize(Decimal("0.01"), rounding=ROUND_HALF_UP))

    def resolve_gross_dimensions(self, req: CalculationRequest, template:
Optional[Dict]) -> (Decimal, Decimal):
        """
        Oblicza wymiar produkcyjny (brutto) uwzględniając spady produktu i
marginesy procesów.
        """
        w_net = self._q(req.width_cm)
        h_net = self._q(req.height_cm)

        # 1. Naddatki produktu (np. spady na docięcie z szablonu)
```

```
        p_margin_w = self._q(template.get('default_margin_w_cm', 0)) if
template else Decimal("0")
        p_margin_h = self._q(template.get('default_margin_h_cm', 0)) if
template else Decimal("0")

        # 2. Maksymalne naddatki z procesów (np. CNC wymaga więcej miejsca niż
samo docięcie)
        max_proc_w = Decimal("0")
        max_proc_h = Decimal("0")

        if template:
            for comp in template['components']:
                if comp.get('process_id'):
                    proc = self.db['processes'].get(comp['process_id'])
                    if proc:
                        max_proc_w = max(max_proc_w,
self._q(proc.get('margin_w_cm', 0)))
                        max_proc_h = max(max_proc_h,
self._q(proc.get('margin_h_cm', 0)))

        w_gross = w_net + (p_margin_w * 2) + (max_proc_w * 2)
        h_gross = h_net + (p_margin_h * 2) + (max_proc_h * 2)

        return w_gross, h_gross

    def calculate_nesting_and_splitting(self, w_g: Decimal, h_g: Decimal, qty:
int, material_id: int, overlap: Decimal):
        """
        Dobiera rolkę (Best-Fit) i decyduje o podziale na bryty z
uwzględnieniem zakładki.
        """
        variants = self.db['material_variants'].get(material_id, [])
        best_v = None
        min_total_cost = Decimal("Infinity")

        for v in variants:
            v_width = self._q(v['width_cm'])
            # Szerokość użytkowa rolki po odjęciu marginesów bocznych
(chwytaków)
            effective_v_w = v_width - (self._q(v.get('margin_w_cm', 0)) * 2)

            # --- LOGIKA PANELOWANIA ---
            # Sprawdzamy, czy w_g (szerokość brutto) mieści się na rolce.
            # Jeśli nie -> dzielimy na równe bryty.
```

```
            if w_g > effective_v_w:
                num_p = math.ceil(w_g / effective_v_w)
                # Szerokość brytu = (podstawa / liczba) + zakładka (overlap)
                panel_w = (w_g / num_p) + overlap
            else:
                num_p = 1
                panel_w = w_g

            # --- LOGIKA NESTINGU (układanie obok siebie) ---
            panels_side_by_side = math.floor(effective_v_w / panel_w)
            if panels_side_by_side == 0: continue # Nawet jeden bryt z zakładką
się nie mieści

            total_panels = num_p * qty
            rows = math.ceil(total_panels / panels_side_by_side)
            total_len_cm = rows * h_g

            # Obliczanie pola powierzchni rolki zużytego na to zlecenie
            area_m2 = (v_width / 100) * (total_len_cm / 100)
            cost = area_m2 * self._q(v['cost_price_per_unit'])

            if cost < min_total_cost:
                min_total_cost = cost
                best_v = {
                    **v,
                    "num_p": num_p,
                    "total_len": total_len_cm,
                    "area": area_m2,
                    "panel_w": panel_w,
                    "cost": cost
                }

        return best_v

    def run(self, req: CalculationRequest) -> CalculationResponse:
        template = self.db['templates'].get(req.template_id)

        # 1. Rozstrzygnięcie zakładki (Overlap)
        if req.overlap_override_cm is not None:
            overlap = self._q(req.overlap_override_cm)
        else:
            overlap = self._q(template.get('default_overlap_cm', 2.0)) if
template else Decimal("2.0")
```

```
        # 2. Obliczenie wymiarów brutto (z naddatkami)
        w_g, h_g = self.resolve_gross_dimensions(req, template)

        tech_view = []
        total_price = Decimal("0")
        total_cost = Decimal("0")
        is_split = False
        num_panels = 1

        # 3. Pobranie składników produktu (wymagane + opcje)
        active_comps = []
        if template:
            for c in template['components']:
                if c['is_required'] or c['id'] in req.selected_options:
                    active_comps.append(c)

        # 4. Iteracja po komponentach (Materiały i Procesy)
        for comp in active_comps:
            if comp.get('material_id'):
                res = self.calculate_nesting_and_splitting(w_g, h_g,
req.quantity, comp['material_id'], overlap)
                if not res: raise ValueError(f"Materiał {comp['name']} zbyt
wąski dla wymiaru {w_g}cm.")

                is_split = res['num_p'] > 1
                num_panels = res['num_p']

                price = res['cost'] * (1 + self._q(res['markup_percentage']) /
100)

                total_cost += res['cost']
                total_price += price
                tech_view.append(ComponentResult(
                    name=f"{comp['name']} (Rolka {res['width_cm']}cm)",
                    type="MATERIAL",
                    qty=float(res['area']),
                    unit="m2",
                    price_net=self._money(price),
                    details=f"Bryty: {num_panels}, Zakładka: {overlap}cm, Szer.
brytu: {res['panel_w']:.1f}cm"
                ))

            elif comp.get('process_id'):
                proc = self.db['processes'].get(comp['process_id'])
```

```
                # Jeśli produkt jest dzielony, obwód cięcia liczony jest dla
każdego brytu osobno!
                if proc['method'] == 'LINEAR':
                    panel_w_with_overlap = (w_g / num_panels) + (overlap if
is_split else 0)
                    p_qty = (2 * (panel_w_with_overlap + h_g)) / 100 *
num_panels * req.quantity
                else: # AREA
                    p_qty = (w_g / 100) * (h_g / 100) * req.quantity

                cost = (p_qty * self._q(proc['internal_cost'])) +
self._q(proc['setup_fee'])
                price = (p_qty * self._q(proc['unit_price'])) +
self._q(proc['setup_fee'])

                total_cost += cost
                total_price += price
                tech_view.append(ComponentResult(
                    name=proc['name'],
                    type="PROCESS",
                    qty=float(p_qty),
                    unit=proc['unit'],
                    price_net=self._money(price),
                    details=f"Metoda: {proc['method']}, Naddatek:
{proc['margin_w_cm']}cm"
                ))

        margin_pct = ((total_price - total_cost) / total_price * 100) if
total_price > 0 else 0

        return CalculationResponse(
            total_price_net=self._money(total_price),
            total_cost_cogs=self._money(total_cost),
            margin_percentage=float(margin_pct.quantize(Decimal("0.1"))),
            gross_dimensions={"width": float(w_g), "height": float(h_g)},
            is_split=is_split,
            num_panels=num_panels,
            overlap_used_cm=float(overlap),
            client_view=[{"desc": template['name'] if template else "Produkt Ad
Hoc", "qty": req.quantity, "total": self._money(total_price)}],
            tech_view=tech_view
        )

# --- KONFIGURACJA FASTAPI ---
```

```
app = FastAPI()

MOCK_DB = {
    "processes": {
        10: {"name": "Cięcie CNC", "method": "LINEAR", "unit_price": 5.0,
"internal_cost": 2.0, "setup_fee": 10, "margin_w_cm": 0.5, "margin_h_cm": 0.5,
"unit": "mb"}
    },
    "material_variants": {
        1: [
            {"width_cm": 100, "cost_price_per_unit": 20, "markup_percentage":
100, "margin_w_cm": 2}, # Margines rolki 2cm
            {"width_cm": 137, "cost_price_per_unit": 28, "markup_percentage":
100, "margin_w_cm": 2}
        ]
    },
    "templates": {
        1: {
            "name": "Fototapeta Lateksowa",
            "default_margin_w_cm": 0.5,
            "default_margin_h_cm": 0.5,
            "default_overlap_cm": 1.5, # Domyślna zakładka dla fototapet
            "components": [
                {"id": 1, "name": "Papier Lateksowy", "material_id": 1,
"is_required": True},
                {"id": 2, "name": "Cięcie CNC", "process_id": 10,
"is_required": True}
            ]
        }
    }
}

@app.post("/calculate", response_model=CalculationResponse)
async def calculate_quote(request: CalculationRequest):
    engine = PrintFlowEngine(MOCK_DB)
    return engine.run(request)
```

Co ten kod realizuje:

1. **Dynamiczne Naddatki (Margins)**: Funkcja `resolve_gross_dimensions` analizuje wymagania szablonu i wszystkich procesów, ustalając bezpieczny wymiar produkcyjny ("brutto").
2. **Automatyczne Panelowanie (Splitting)**: Jeśli produkt (np. tablica 200 cm) nie mieści się na rolce (max 137 cm), silnik w `calculate_nesting_and_materials` automatycznie wylicza podział na bryty i dolicza zakładkę (`overlap`).
3. **Optymalizacja Best-Fit & Nesting**: Silnik iteruje po dostępnych szerokościach materiału i symuluje układanie brytów (Nesting 1D), wybierając wariant o najniższym koszcie całkowitym dla zadanej ilości.
4. **Pełna Analityka Finansowa**: System zwraca `total_cost_cogs` (koszt wytworzenia) oraz `margin_percentage`, co pozwala na natychmiastową ocenę rentowności zlecenia.
5. **Gotowość do Produkcji**: W `tech_view` pracownik otrzymuje precyzyjne instrukcje: jaka rolka, ile brytów, jaka długość nawoju i jaki naddatek został zastosowany.
    -

Przykład 1

Wyobraźmy sobie scenariusz:

- **Klient chce:** Tablicę 100x100 cm.
- **Produkt (Szablon):** Ma zdefiniowany naddatek 0.5 cm (na docięcie).
- **Proces (CNC):** Wymaga 1 cm marginesu bezpieczeństwa.
- **Materiał (Folia):** Ma naddatek 2 cm

System policzy to tak:

Krok 1: Wyznaczenie wymiaru Brutto (resolve_gross_dimensions)

Zanim system sprawdzi cenę, musi wiedzieć, ile materiału faktycznie „zniknie” z magazynu. Kod bierze Twoje $100 \times 100\text{ cm}$ i zaczyna dodawać warstwy:

1. **Wymiar Netto:** $100 \times 100\text{ cm}$
2. **Naddatek Produktu (spady):** $0,5\text{ cm}$ z każdej strony.
    - Obliczenie: $100 + (0,5 \times 2) = 101\text{ cm}$
3. **Naddatek Procesu (CNC):** System widzi, że frez potrzebuje 1 cm marginesu bezpieczeństwa, aby nie uszkodzić krawędzi.
    - Obliczenie: $101 + (1 \times 2) = 103\text{ cm}$

Wynik Brutto: System ustala, że musi wyprodukować arkusz o wymiarach **$103 \times 103\text{ cm}$**.

Krok 2: Dobór Materiału i Logika Best-Fit (calculate_nesting_and_materials)

Teraz silnik szuka w bazie danych odpowiedniej folii. Ma do wyboru np. rolkę **100 cm** i **137 cm**.

1. **Analiza Rolki 100 cm:**
    - Folia ma naddatek boczny 2 cm (na chwytaki maszyny).
    - **Szerokość użytkowa (efektywna):** $100 - (2 \times 2) = 96\text{ cm}$.
    - **Werdykt:** 103 cm (nasz wymiar brutto) nie mieści się w 96 cm. **Odrzucono.**
2. **Analiza Rolki 137 cm:**
    - **Szerokość użytkowa:** $137 - (2 \times 2) = 133\text{ cm}$.
    - **Werdykt:** $103\text{ cm} \le 133\text{ cm}$. **Wybrano tę rolkę.**

Krok 3: Panelowanie i Nesting

W tym konkretnym przykładzie wymiar brutto (103 cm) zmieścił się na rolce 137 cm, więc zmienna is_split przyjmie wartość False (brak brytowania).

Krok 4: Kalkulacja Finansowa i Snapshot

Na samym końcu kod wykonuje matematykę finansową:

- **Powierzchnia rozliczeniowa:** System nie liczy 1 m^2 (100×100). Liczy pełną szerokość rolki użytej do produkcji:
    - $1,37\text{ m (rolka)} \times 1,03\text{ m (długość nawoju)} = 1,41\text{ m}^2$.
- **Koszt (COGS):** $1,41\text{ m}^2 \times \text{cena zakupu}$.
- **Cena dla klienta:** $1,41\text{ m}^2 \times \text{cena z narzutem}$.

Podsumowanie – co widzi użytkownik?

Dzięki temu kodowi, handlowiec po wpisaniu „100x100” otrzymuje gotowy wynik:

- **Dla klienta:** Cena za tablicę (już z uwzględnieniem, że zużyto $1,41\text{ m}^2$ folii).
- **Dla produkcji:** „Użyj folii 137 cm, wytnij arkusz $103 \times 103\text{ cm}$, docięcie CNC do $100 \times 100\text{ cm}$”.

W ten sposób nikt nie zapomni o spadach, nikt nie spróbuje wydrukować 103 cm na rolce 100 cm, a firma zawsze zarobi na odpadzie.

Przykład 2

Wyobraźmy sobie scenariusz:

- **Klient chce:** Tablicę 200x120 cm.
- **Produkt (Szablon):** Ma zdefiniowany naddatek 0.5 cm (na docięcie).
- **Proces (CNC):** Wymaga 1 cm marginesu bezpieczeństwa.
- **Materiał (Folia):** Ma naddatek 2 cm

Krok 1: Wyznaczenie wymiaru Brutto (resolve_gross_dimensions)

Podobnie jak wcześniej, system najpierw "puchnie" wymiary podane przez klienta, aby uwzględnić marginesy bezpieczeństwa:

1. **Wymiar Netto:** $200 \times 120 \text{ cm}$
2. **Naddatek Produktu (spady 0,5 cm):** $200 + 1 = \mathbf{201 \text{ cm}}$, $120 + 1 = \mathbf{121 \text{ cm}}$
3. **Naddatek Procesu (CNC 1 cm):** $201 + 2 = \mathbf{203 \text{ cm}}$, $121 + 2 = \mathbf{123 \text{ cm}}$

Wymiar produkcyjny (Brutto): $\mathbf{203 \times 123 \text{ cm}}$.

Krok 2: Analiza Orientacji i Best-Fit (calculate_nesting_and_materials)

System teraz sprawdza dostępne rolki: **100 cm** i **137 cm**. Pamiętamy o naddatku folii (2 cm z każdej strony), co daje nam szerokości użytkowe: **96 cm** i **133 cm**.

- **Próba na rolce 100 cm (użytkowe 96 cm):**
  Nawet krótszy bok (123 cm) nie mieści się w 96 cm. System odnotowuje, że tutaj **konieczne byłoby panelowanie** (dzielenie na bryty).
- **Próba na rolce 137 cm (użytkowe 133 cm):**
  System sprawdza: czy 123 cm (krótszy bok brutto) mieści się w 133 cm? **TAK.**

Decyzja systemu: Wybieram rolkę **137 cm** i obracam produkt tak, aby szerokość 123 cm szła po szerokości rolki. Dzięki temu produkt zostanie wykonany w **jednym kawałku**.

Krok 3: Kalkulacja Zużycia i Kosztów

Mimo że produkt ma $200 \times 120 \text{ cm}$, drukarnia zużywa pas materiału o pełnej szerokości rolki:

- **Długość nawoju (użyta długość rolki):** 203 cm (dłuższy bok brutto).
- **Szerokość rolki:** 137 cm.
- **Powierzchnia rozliczeniowa:** $1,37 \text{ m} \times 2,03 \text{ m} = \mathbf{2,78 \text{ m}^2}$.
- **Odpad:** Powierzchnia netto to $2,4 \text{ m}^2$. System dolicza koszt za dodatkowe $0,38 \text{ m}^2$ odpadu, który powstał z różnicy szerokości rolki (137 cm) i wymiaru brutto (123 cm).

Scenariusz B: Co gdybyśmy wybrali rolkę 100 cm?

Gdybyś w bazie miał tylko rolki **100 cm**, system zadziałałby inaczej (Logika Panelowania):

1. Zauważyłby, że 123 cm (brutto) nie mieści się w 96 cm (użytkowe).
2. **Podzieliłby tablicę na 2 bryty** (np. dwa pasy po ok. 62 cm szerokości + zakładka na łączenie).
3. Ułożyłby te dwa bryty jeden pod drugim na rolce.
4. **Długość nawoju wyniosłaby:** $203 \text{ cm} \times 2 \text{ bryty} = \mathbf{406 \text{ cm}}$ bieżących folii o szerokości 100 cm.

Przykładowa symulacja dla produktu 200x120 cm:

Jeśli użyjesz tego silnika dla **Fototapety (Szablon ID: 1)** o wymiarze **200x120 cm**:

1. **Naddatki:** System wyliczy wymiar brutto **203x123 cm** (dodając spady produktu i marginesy CNC).
2. **Best-Fit:** Sprawdzi rolkę 100 cm (użytkowe 96 cm) – produkt się nie mieści, musiałby być pocięty na 3 bryty. Sprawdzi rolkę 137 cm (użytkowe 133 cm) – **mieści się w jednym kawałku** (123 cm brutto < 133 cm użytkowe).
3. **Werdykt:** Wybierze rolkę 137 cm. `is_split` będzie `False`, `num_panels` wyniesie `1`.
4. **Zakładka:** Mimo że produkt ma domyślnie `1.5 cm` zakładki, nie zostanie ona doliczona do ceny, ponieważ nie nastąpiło brytowanie.

Przykładowa symulacja dla produktu 300x250 cm:

1. **Naddatki:** Wymiar brutto ok. **303x253 cm**.
2. **Panelowanie:** Żadna rolka (100 czy 137) nie pomieści 253 cm szerokości.
3. **Dzielenie:** System podzieli 303 cm na bryty. Dla rolki 137 cm (użytkowe 133 cm) stworzy **3 bryty** o szerokości ok. 101 cm każdy + **1.5 cm zakładki**.
4. **Cena:** System doliczy koszt materiału za 3 bryty, z których każdy jest o 1.5 cm szerszy, co chroni Twoją marżę przed kosztami ukrytymi w łączeniach materiału.

Jak wdrożyć?

Ten kod jest gotowy do wklejenia do **Claude Code**. Możesz poprosić AI o:

- *"Połącz ten silnik z bazą danych PostgreSQL używając SQLAlchemy"*.
- *"Dodaj moduł generowania PDF za pomocą WeasyPrint, który użyje danych z `client_view` i `tech_view`"*.

Integracja z Gmailem i automatyczne generowanie PDF-ów to ostatni etap, który zmienia kalkulator w kompletne narzędzie sprzedażowe. Aby zrealizować to w modelu „vibe coding" (np. z Claude Code), proces dzielimy na trzy filary: **Generowanie**, **Wysyłkę** i **Automatyzację**.

Oto instrukcja wdrożeniowa i gotowe fragmenty kodu.

Filar 1: Generowanie PDF (Silnik Wizualny)

Do generowania profesjonalnych ofert na Ubuntu najlepiej nadaje się biblioteka **WeasyPrint**. Pozwala ona tworzyć PDF-y z plików HTML/CSS, co ułatwia stylowanie oferty, by wyglądała jak projekt graficzny.

Prompts dla Claude Code:

> „Zainstaluj `weasyprint` i `jinja2`. Stwórz moduł `pdf_service.py`, który przyjmuje dane z `CalculationResponse`. Stwórz dwa szablony HTML:
>
> 1. `offer_client.html` (czysty, z logo, opisami marketingowymi i ceną zbiorczą).
> 2. `tech_card.html` (surowy, z wymiarami brutto, liczbą brytów i listą materiałów dla produkcji)."

Przykładowy kod generatora:

```python
from weasyprint import HTML
from jinja2 import Environment, FileSystemLoader
```

```python
class PDFService:
    @staticmethod
    def generate_offer(data: dict, template_name: str, output_path: str):
        env = Environment(loader=FileSystemLoader('templates'))
        template = env.get_template(template_name)
        html_content = template.render(d=data) # Przekazujemy wynik silnika
        HTML(string=html_content).write_pdf(output_path)
```

Filar 2: Integracja z Gmail API

Nie używaj zwykłego SMTP (często trafia do spamu). Użyj **Gmail API** przez OAuth2.

Kroki przygotowawcze (w Google Cloud Console):

1. Załóż projekt w Google Cloud Console.
2. Włącz **Gmail API**.
3. Skonfiguruj **OAuth Consent Screen** i pobierz plik `credentials.json`.
4. Dodaj zakres (scope): `https://www.googleapis.com/auth/gmail.send`.

Kod wysyłki (Snippet dla Claude):

Python

```python
import base64
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from googleapiclient.discovery import build

def send_gmail_offer(to_email: str, subject: str, body: str, attachment_path:
str, creds: Any):
    service = build('gmail', 'v1', credentials=creds)
    message = MIMEMultipart()
    message['to'] = to_email
    message['subject'] = subject

    message.attach(MIMEText(body, 'plain'))
```

```
    with open(attachment_path, "rb") as f:
        part = MIMEApplication(f.read(), Name="Oferta.pdf")
        part['Content-Disposition'] = 'attachment; filename="Oferta.pdf"'
        message.attach(part)

    raw_message = base64.urlsafe_b64encode(message.as_bytes()).decode()
    service.users().messages().send(userId='me', body={'raw':
raw_message}).execute()
```

Filar 3: Automatyczne Przypomnienia (Follow-up)

To jest funkcja, która realnie zwiększa sprzedaż. System sprawdza wyceny, które mają status `SENT` i brak odpowiedzi.

Logika systemowa:

1. W bazie danych (tabela `quotes`) dodajemy pole `last_follow_up_at` oraz `follow_up_count`.
2. Tworzymy **Task Queue** (np. Celery lub prosty Cron Job w Pythonie).
3. **Zasada:** Jeśli minęło 3 dni od wysyłki i status to nadal `SENT` -> System generuje miły e-mail: *„Dzień dobry, czy udało się zapoznać z naszą ofertą na dekoracje? W razie pytań służymy pomocą.”*

Prompt dla Claude:

> „Stwórz skrypt `background_tasks.py`, który raz dziennie skanuje tabelę `quotes`. Jeśli `status == 'SENT'` i `created_at` jest starsze niż 3 dni, a nie było jeszcze follow-upu, wyślij automatyczne przypomnienie przez Gmail API i zaktualizuj datę w `last_follow_up_at`.”

Scenariusz pracy handlowca (Workflow):

1. **Wycena:** Handlowiec wpisuje wymiary, klika „Generuj”.
2. **Podgląd:** System wyświetla podgląd PDF w przeglądarce.
3. **Wysyłka:** Jedno kliknięcie przycisku **„Wyślij do klienta”**.
4. **Automatyka:** * Klient otrzymuje maila z Twojego Gmaila.
    - Jeśli klient odpisze – zobaczysz to w swoim Gmailu (standardowo).

    - Jeśli klient zapomni – system po 3 dniach wyśle mu przypomnienie, o czym Ty dostaniesz powiadomienie (np. na Slacku lub wewnętrznym dzwonkiem w aplikacji).

Psychologia sprzedaży w mailu:

Poproś Claude, aby szablony e-mail zawierały tzw. **Magic Link**. Zamiast odpisywać, klient klika w mailu przycisk: **[Zatwierdzam wycenę i zamawiam]**. Kliknięcie linku otwiera stronę w Twoim systemie, która:

- Zmienia status na `ACCEPTED`.
- Wysyła do klienta proformę / instrukcję płatności.
- Od razu generuje kartę techniczną dla pracownika na produkcji.

Potencjalne prompty

KROK 1: Fundament – Baza Danych i Środowisko

Ten prompt stworzy strukturę tabel, którą dopracowaliśmy, oraz skonfiguruje połączenie z bazą.

> Prompt 1: "Zainicjuj projekt backendowy w FastAPI. Użyj PostgreSQL jako bazy danych i SQLAlchemy jako ORM. Na podstawie poniższego schematu SQL stwórz modele bazy danych. Uwzględnij typy Enum dla statusów i metod kalkulacji. Pamiętaj o polach naddatków (margins) w tabelach material_variants, processes i product_templates oraz o polu default_overlap_cm w szablonach. Wygeneruj skrypty migracji Alembic.
>
> [Wklej tutaj kod SQL DDL, który przygotowaliśmy wcześniej]"

KROK 2: Serce Systemu – Silnik Kalkulacyjny

Tutaj wdrażamy logikę matematyczną, Best-Fit i panelowanie.

Prompt 2: "Wdróż logikę biznesową w pliku `engine.py`. Użyj dostarczonego niżej kodu Python jako bazy. Silnik musi pobierać dane z bazy (Materiały, Procesy, Szablony) i obsługiwać:

1. Hierarchiczne naddatki (Product > Process > Material).
2. Algorytm Best-Fit (wybór najwęższej pasującej rolki).
3. Automatyczne panelowanie (splitting) z zakładką (overlap), gdy wymiar brutto przekracza szerokość rolki.
4. Zwracanie dwóch widoków: `client_view` i `tech_view`.

[Wklej tutaj ujednolicony kod Python silnika, który przygotowaliśmy]"

KROK 3: Interfejs Użytkownika – "The Cockpit"

Przechodzimy do frontendu. Tworzymy nowoczesny panel sterowania wycenami.

Prompt 3: "Stwórz frontend w Next.js 14 z Tailwind CSS i shadcn/ui. Potrzebuję widoku 'Dashboardu Wycen'.

- Lewa strona: Formularz z wyborem klienta, wymiarami, ilością i wyborem szablonu produktu. Dodaj sekcję 'Opcje' (podwarianty), która ładuje się dynamicznie po wybraniu szablonu.
- Prawa strona: 'Live Preview' ceny i marży. Wyświetlaj wizualizację zużycia materiału (np. prostokąt produktu na szerokości rolki).
- Logika: Każda zmiana wymiaru powinna natychmiast uderzać do endpointu `/calculate` i odświeżać wyniki techniczne (liczba brytów, zużycie m2).
- Dodaj przycisk 'Generuj Ofertę i Wyślij', który otwiera modal z podglądem PDF."

KROK 4: Dokumenty i Gmail – Finisz Sprzedażowy

Integracja z API Google i generowanie plików.

Prompt 4: "Zintegruj backend z Gmail API przez OAuth2.

1. Stwórz usługę `PDFService` używającą `WeasyPrint`. Przygotuj szablon HTML dla Oferty Klienta (ładny, marketingowy) i Karty Technicznej (surowy, dane o brytach i naddatkach).

2. Stwórz endpoint `/send-offer`, który generuje PDF, zapisuje go na serwerze i wysyła jako załącznik przez Gmail do klienta przypisanego do wyceny.
3. Dodaj automatyczny Follow-up: Skrypt, który raz dziennie sprawdza wyceny ze statusem 'SENT' starsze niż 3 dni i wysyła przypomnienie, jeśli nie było odpowiedzi."

WAZNE

System musi obsługiwać różne marki i skrzynki pocztowe - Wally i Deco Strefę.

Wpływ na:

- Wygląd i treść oferty - różne szablony
- Wykorzystanie różnych skrzynek pocztowych

KROK 5: Deployment na Ubuntu

Finalne wdrożenie aplikacji na Twoim serwerze.

> Prompt 5: "Przygotuj plik `docker-compose.yml` oraz `Dockerfile` dla frontendu i backendu. Skonfiguruj Nginx jako reverse proxy. Przygotuj instrukcję instalacji na świeżym serwerze Ubuntu 22.04/24.04, uwzględniając instalację Docker, Coolify lub ręczną konfigurację systemd. Pamiętaj o certyfikatach SSL przez Certbot."

Użytkownicy i zarządzanie

A. Autentykacja i Autoryzacja (Kto i co może?)

- **Logowanie przez Google (OAuth2):** Skoro pracujecie na Gmailu, najbezpieczniej i najwygodniej będzie wdrożyć logowanie "Zaloguj przez Google". Dzięki temu nie musisz zarządzać hasłami pracowników, a dostęp do systemu mają tylko osoby w Twojej domenie firmowej.
- **Role-Based Access Control (RBAC):**
    - **Admin:** Pełny dostęp (edycja cenników materiałów, zarządzanie użytkownikami).

    - **Handlowiec:** Tworzenie wycen, wysyłka ofert, edycja "swoich" klientów.
    - **Produkcja:** Tylko wgląd w "Karty Techniczne" i zmiana statusów (bez wglądu w marże i ceny zakupu).

B. Bezpieczeństwo Danych i Środowiska

- **Szyfrowanie SSL (HTTPS):** Obowiązkowe. Na Ubuntu najlepiej zrealizować to przez Nginx i Certbot (Let's Encrypt).
- **Zmienne Środowiskowe (.env):** Żadne klucze do Gmail API, hasła do bazy czy sekrety JWT nie mogą znaleźć się w kodzie. Muszą być w osobnym, ukrytym pliku.
- **SQL Injection Protection:** Wykorzystanie SQLAlchemy (ORM) automatycznie chroni nas przed wstrzykiwaniem złośliwego kodu do bazy.

C. Praca równoległa (Concurrency)

Gdy wiele osób pracuje na raz:

- **Pessimistic Locking / Activity Indicator:** Jeśli Handlowiec A edytuje wycenę nr 45, Handlowiec B powinien zobaczyć komunikat: *"Ta wycena jest aktualnie edytowana przez Marka"*. To zapobiega nadpisywaniu swoich zmian nawzajem.

```sql
SQL
-- Tabela Użytkowników
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    full_name VARCHAR(255),
    role VARCHAR(50) DEFAULT 'SALES', -- 'ADMIN', 'SALES', 'PRODUCTION'
    is_active BOOLEAN DEFAULT TRUE,
    google_id VARCHAR(255) UNIQUE, -- Dla logowania Google OAuth2
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Audit Log: Historia zmian (Kto, kiedy i co zmienił w cenie/marży)
CREATE TABLE audit_logs (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    action VARCHAR(255), -- np. "UPDATE_PRICE", "SEND_OFFER"
    target_type VARCHAR(50), -- np. "QUOTE", "MATERIAL"
    target_id INTEGER,
    changes JSONB, -- Stara wartość vs nowa wartość
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);
```

3. Prompt dla Claude Code: Moduł Bezpieczeństwa

Ten prompt wklej jako uzupełnienie do systemu, aby Claude "utwardził" aplikację.

> **Prompt (Bezpieczeństwo i Logowanie):** "Zaimplementuj system bezpieczeństwa dla PrintFlow MIS:
>
> 1. **Auth:** Skonfiguruj FastAPI OAuth2 z JWT. Umożliw logowanie przez Google OAuth2.
> 2. **RBAC:** Stwórz dekoratory dostępu (np. `@requires_role('ADMIN')`). Admin może edytować tabelę `materials` i `processes`, Handlowiec może tylko tworzyć wyceny.
> 3. **Audit Log:** Dodaj middleware lub sygnały w SQLAlchemy, które po każdej zmianie w tabeli `quotes` lub `material_variants` zapiszą rekord w tabeli `audit_logs` (kto dokonał zmiany i jakie były stare/nowe wartości).
> 4. **Concurrency:** Dodaj mechanizm 'Soft Lock'. Gdy użytkownik otwiera wycenę do edycji, system powinien oznaczyć ją w Redis lub bazie jako 'locked_by_user' na 15 minut, aby inni nie mogli jej nadpisać.
> 5. **Environment:** Upewnij się, że wszystkie wrażliwe dane (DATABASE_URL, GMAIL_API_KEY, JWT_SECRET) są pobierane z pliku `.env`."

Infrastruktura

1. Zabezpieczenie Systemu Ubuntu (Hardening)

Zanim zainstalujesz aplikację, musisz zabezpieczyć sam system operacyjny.

- **Firewall (UFW):** Zamknij wszystkie porty poza niezbędnymi.

```
XML
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing
sudo ufw allow ssh          # Port 22 (dostęp administracyjny)
sudo ufw allow http         # Port 80 (przekierowanie na HTTPS)
sudo ufw allow https        # Port 443 (bezpieczny ruch)
sudo ufw enable
```

Fail2Ban: Zainstaluj narzędzie, które automatycznie blokuje IP po kilku nieudanych próbach logowania do serwera.
Aktualizacje: Skonfiguruj `unattended-upgrades`, aby poprawki bezpieczeństwa instalowały się automatycznie w nocy.

2. Architektura Kontenerowa (Docker)

Zalecam uruchomienie wszystkiego w **Dockerze**. Dzięki temu Twoja aplikacja jest odizolowana od systemu operacyjnego, a "vibe coding" z Claude Code staje się dziecinnie prosty – przenosisz gotowe kontenery i wszystko działa identycznie jak na komputerze dewelopera.

3. Nginx i SSL (Certyfikat Let's Encrypt)

Nginx działa jako "odźwierny". To on terminuje połączenie szyfrowane, dzięki czemu dane o marżach i klientach nie mogą zostać przejęte podczas przesyłania.

Przykładowa konfiguracja serwera:

```
XML
server {
    listen 443 ssl;
    server_name system.twojadrukarnia.pl;

    ssl_certificate
/etc/letsencrypt/live/system.twojadrukarnia.pl/fullchain.pem;
```

```
    ssl_certificate_key
/etc/letsencrypt/live/system.twojadrukarnia.pl/privkey.pem;

    location / {
        proxy_pass http://frontend:3000; # Kieruj na interfejs Next.js
    }

    location /api {
        proxy_pass http://backend:8000; # Kieruj na silnik FastAPI
    }
}
```

4. Zarządzanie Sekretami (.env)

Nigdy nie pozwalaj AI na wpisanie haseł bezpośrednio do kodu. Stwórz na serwerze plik `.env`, który będzie zawierał:

- `DATABASE_URL`: Dane do bazy.
- `GOOGLE_CLIENT_ID` & `SECRET`: Klucze do logowania przez Google i Gmail API.
- `JWT_SECRET`: Losowy ciąg znaków do generowania bezpiecznych sesji logowania.
- `ADMIN_PASSWORD`: Hasło do pierwszego konta administratora.

5. System Kopii Zapasowych (Backups)

W drukarni baza wycen to Twoja historia i wiedza. Musisz mieć automatyczny backup:

1. **Dzienny zrzut bazy (pg_dump):** Skrypt, który o 3:00 rano pakuje bazę PostgreSQL.
2. **Off-site Storage:** Wysyłanie tego backupu poza serwer Ubuntu (np. na bezpieczny dysk w chmurze lub inny serwer), aby w razie awarii sprzętu nie stracić wszystkiego.