🛠️ Spécifications Techniques : La Réaction en Chaîne "Voltage Pricer"

Ce document détaille l'ingénierie derrière chaque fichier, expliquant comment chaque étape est la conséquence vitale de la précédente (l'effet domino), en utilisant la nouvelle nomenclature sémantique.

🧱 ÉTAPE 1 : L'Acquisition de la "Mémoire" du Marché

Fichier : `src/ingestion/api_connector.py` (anciennement *elia_client.py*)

Le Problème : Pour pricer le futur, l'Intelligence Artificielle a besoin de connaître les comportements passés (saisonnalité, pics de froid, volatilité réelle). Sans données historiques, on ne fait que des suppositions mathématiques vides.

Comment on fait ça ? On utilise la bibliothèque `requests` pour interroger l'API REST v2.1 d'ELIA et `pandas` pour transformer le flux JSON brut en séries temporelles exploitables.

- **Code Précis (Nettoyage et Normalisation) :**

```
# Téléchargement et rééchantillonnage horaire
df = pd.DataFrame(requests.get(url, params=params).json())
# On force un point par heure, on interpole les trous et on remplit les vides
hourly_data = df[target_col].resample("h").mean().interpolate().bfill()
```

- **Formule de Nettoyage :** Si une donnée est manquante à l'instant t, on applique une interpolation linéaire :

$$X_t = \frac{X_{t-1} + X_{t+1}}{2}$$

Conséquence pour l'étape suivante : Nous avons maintenant une base de données historique propre, mais elle est bloquée dans le passé. Nous devons maintenant définir le niveau de prix pour demain.

⚖️ ÉTAPE 2 : La Calibration de l'Ancre Marché (Forward)

Fichier : `src/ingestion/data_calibration.py` (anciennement *market_data.py*)

La Conséquence de l'étape 1 : Les données historiques nous disent ce qui s'est passé, mais le marché pour 2026 se négocie sur des niveaux différents. On doit "ancrer" le modèle sur le prix de marché futur.

Comment on fait ça ? On calcule un prix de référence (Forward) en pondérant le prix Spot récent et un niveau d'équilibre théorique long terme.

- **Code Précis :**

```
# Calcul du prix de base 2026 (40% réel récent / 60% équilibre)
cal_base = (spot_ref * 0.4) + (85.0 * 0.6)
# Le Peak (heures pleines) est pricé avec une prime de 15%
cal_peak = cal_base * 1.15
```

- **Formule Mathématique :**

$$P_{forward} = (w \cdot P_{spot}) + ((1 - w) \cdot P_{équilibre})$$

où $w = 0.4$ représente le poids accordé à la conjoncture actuelle.

Conséquence pour l'étape suivante : On connaît maintenant le prix moyen cible pour 2026, mais on ne sait pas encore "quand" l'électricité sera chère ou gratuite durant l'année.

🧠 ÉTAPE 3 : La Synthèse Prévisionnelle (HPFC via ML)

Fichier : `src/domain/hpfc_forecasting.py` (anciennement *ml_forecasting.py*)

La Conséquence de l'étape 2 : Nous avons une cible financière (ex: 95€), mais pour pricer un client, nous avons besoin d'une courbe de **8760 points** (une valeur par heure de l'année).

Comment on fait ça ? On entraîne un modèle **XGBoost** sur 10 ans de données réelles pour apprendre la corrélation entre le temps (heure, jour, mois) et le prix.

- **Code Précis (Feature Engineering & Level Shift) :**

```
# On apprend à l'IA les variables temporelles
df['hour'] = dt_idx.hour
df['is_peak'] = ((df['hour'] >= 8) & (df['hour'] < 20)).astype(int)
# Après prédiction, on décale la courbe pour qu'elle touche notre cible (Etape 2)
adjustment = target_forward - predicted_mean
final_hpfc = predicted_prices + adjustment
```

- **Formule de "Level Shift" (Recalage) :**

$$HPFC_{t,final} = \hat{P}_{t,ML} + \left(P_{target} - \frac{1}{N} \sum \hat{P}_{i,ML} \right)$$

Conséquence pour l'étape suivante : On a enfin les prix de 2026 heure par heure. Il nous manque maintenant de savoir combien le client va consommer.

👤 ÉTAPE 4 : Le Portrait-Robot du Client (Load Curve)

Fichier : `src/ingestion/client_load_profiling.py` (anciennement *curve_generator.py*)

La Conséquence de l'étape 3 : On connaît le prix horaire, mais on doit savoir si le client consomme durant les heures chères ou les heures bon marché.

Comment on fait ça ? On crée une courbe de charge (Load Curve) normalisée qui respecte le volume annuel demandé par le client.

- **Code Précis (Scaling de volume) :**

```
# On prend une forme (pattern) et on l'étire pour atteindre le volume MWh
total_units = base_pattern.sum()
normalized_curve = (base_pattern / total_units) * annual_volume_mwh
```

- **Logique de conservation :** On s'assure mathématiquement que :

$$\int_{0}^{8760} Load(t)\,dt = Volume_{Contrat}$$

Conséquence pour l'étape suivante : Nous avons les deux pièces du puzzle : la courbe des prix (Etape 3) et la courbe des volumes (Etape 4).

➗ ÉTAPE 5 : La Fusion (Calcul du Coût d'Achat)

Fichier : `src/domain/mix_load_&_price.py` (anciennement *pricing_models.py*)

La Conséquence de l'étape 4 : On doit maintenant "croiser" ces deux courbes pour savoir combien va coûter l'énergie brute pour ce client spécifique.

Comment on fait ça ? On réalise un produit scalaire entre le vecteur "Consommation" et le vecteur "Prix".

- **Code Précis :**

```
# Produit heure par heure
commodity_cost = (load_curve * hpfc).sum()
# Coût unitaire moyen
unit_cost = commodity_cost / total_volume
```

- **Formule de Valorisation :**

$$Coût_{Total} = \sum_{t=1}^{8760} (Volume_t \times Prix_t)$$

Conséquence pour l'étape suivante : On a le coût d'achat "sec". Mais ce prix doit être protégé contre les incertitudes de consommation.

🛡️ ÉTAPE 6 : L'Ingénierie du Risque (Primes)

Fichier : `src/domain/risk_models.py` (inchangé)

La Conséquence de l'étape 5 : Le coût sec est le prix minimum. Si le client consomme plus que prévu lors d'un pic de prix, le fournisseur perd sa marge. On ajoute des primes de protection.

Comment on fait ça ? On calcule le **Profiling Cost** (prime liée à la forme) et une **Prime de Swing** (liée à l'incertitude du volume).

- **Code Précis :**

```
# Prime de profilage : différence entre le prix capturé et le prix moyen
profiling_premium = (total_cost / total_vol) - market_base_price
# Prime de Swing basée sur la volatilité historique (Etape 1)
swing_premium = (volatility * 5) * size_factor
```

- **Formule du Profiling :**

$$P_{profiling} = \frac{\sum(L_t \cdot P_t)}{\sum L_t} - \bar{P}_{marché}$$

Conséquence pour l'étape suivante : Le prix est sécurisé. Il ne reste plus qu'à générer le livrable.

📈 ÉTAPE 7 : Le Livrable Final (Reporting)

Fichiers : `src/reporting/deal_ticket_gen.py` & `app.py`

La Conséquence de l'étape 6 : Les calculs sont finis. Il faut transformer ces milliers de lignes de données en un document commercial.

Comment on fait ça ? On utilise `xlsxwriter` pour générer un fichier Excel avec des graphiques natifs et `streamlit` pour piloter l'ensemble.

🏆 RÉSUMÉ DE LA CHAÎNE LOGIQUE (EFFET DOMINO)

1. **API CONNECTOR** ➡️ Donne la **Mémoire** (Passé).
2. **DATA CALIBRATION** ➡️ Fixe la **Cible** (Ancre 2026).
3. **HPFC FORECASTING** ➡️ Prédit la **Structure** (Futur).
4. **CLIENT LOAD PROFILING** ➡️ Définit le **Besoin** (Quantité).
5. **MIX LOAD & PRICE** ➡️ Calcule la **Facture** (Coût sec).
6. **RISK MODELS** ➡️ Ajoute la **Sécurité** (Marges).