FRIS — STEP 1: SYSTEM UNDERSTANDING (FROZEN CONTRACT)

Status: FINAL — do not modify after this step
Purpose: Establish a 100% accurate mental and technical model of the existing codebase **before** designing inference.

This document is a **read-only truth snapshot** of what exists today.
No new abstractions. No refactors. No future design decisions.

STEP 1 GOAL (WHY THIS EXISTS)

To clearly answer, on paper:

1. What each file is responsible for
2. What data each file consumes
3. What data each file produces
4. Where model outputs are generated
5. Where the system is currently broken

If this understanding is wrong, everything downstream will be wrong.

HARD RULES (LOCKED)

1. `features.py` is the **single source of truth** for feature engineering
2. No raw logic or feature creation inside `models.py` or `explain.py`
3. No notebooks inside `src/`
4. Everything starts from **raw transaction input** (conceptually)
5. There must be **one and only one inference path**

No exceptions.

PART 1 — GLOBAL SYSTEM VIEW

The Three Worlds (Current Reality)

FRIS currently exists as **three disconnected but internally correct systems**:

1. **Feature Engineering World** — `src/features.py`
2. **Model Inference World** — `src/models.py`

3. **Explainability World** — `src/explain.py`

Each world works in isolation.

Critical fact: There is currently **no function** that connects raw input to these three worlds.

That missing connector will be designed in STEP 2.

PART 2 — FEATURE ENGINEERING CONTRACT (`src/features.py`)

Responsibility

`features.py` defines the **only valid transformation** from raw transactional data to engineered features.

It answers exactly three questions: 1. What raw columns are required? 2. What engineered columns are produced? 3. What fitted preprocessing objects are required at inference?

It does **nothing else**.

Core Entry Point

```
df_final, preprocessors = feature_pipeline(
    df,
    fit=True | False,
    preprocessors=None
)
```

Input Contract (RAW DATA)

The pipeline expects a pandas DataFrame containing **at minimum**:

Column	Type	Meaning
Time	int / float	Seconds since reference epoch
Amount	float	Transaction amount

No other raw columns are required in the current simulated system.

Deterministic Pipeline Stages (Ordered)

1. **Timestamp Features**
2. `timestamp`
3. `hour`
4. `dayofweek`
5. **Amount Features**
6. `amount_log`
7. `amount_scaled` (RobustScaler)
8. **Synthetic Category Generation (Seeded RNG)**
9. `merchant_id`
10. `device_type`
11. `geo_bucket`
12. `account_id`
13. `account_age_days`
14. **Frequency Features**
15. `merchant_freq`
16. `device_freq`
17. `account_txn_count`
18. **Rolling Behavioral Features**
19. `last_5_mean_amount`
20. `last_5_count`
21. **Missing Flags**
22. `merchant_id_missing`
23. `device_type_missing`
24. `geo_bucket_missing`
25. `account_age_days_missing`
26. **Categorical Frequency Encodings**
27. `merchant_id_fe`
28. `device_type_fe`

29. `geo_bucket_fe`
30. `account_id_fe`
31. **Interaction Features**
32. `amount_times_age`
33. `is_new_merchant`
34. **PCA (Auxiliary)**
35. `pca_x`
36. `pca_y`

Output Contract

`feature_pipeline` returns:

- `df_final`: **full engineered DataFrame** (one row per transaction)
- `preprocessors`: fitted objects used during inference

Important: - This output includes **more columns than models may consume** - Models later **slice** what they need

---

Preprocessors Object

```
preprocessors = {
    "scaler": RobustScaler,
    "encoders": dict,
    "pca": PCA
}
```

These are fit **only during training** and must be reused at inference.

What `features.py` DOES NOT DO

- Run ML models
- Compute probabilities
- Compute anomaly scores
- Produce cluster IDs
- Produce latent vectors
- Perform inference

- Assign labels
- Explain predictions

If it does any of the above, the system is broken.

PART 3 — MODEL INFERENCE CONTRACT (`src/models.py`)

Responsibility

`models.py` consumes **engineered features** and produces:

- Base model signals
- Meta-features for stacking
- Final fraud probability
- Final fraud label

It assumes features already exist.

`load_models()`

Loads and caches all frozen inference artifacts:

- Models: XGBoost, IsolationForest, Autoencoder, Stacker
- Feature lists: `xgb_features`, `iforest_features`, `ae_features`
- Stacker metadata: `meta_features`, `threshold`

Important: - Feature lists define **what each model consumes**, not how features are created

---

Base Model Signals (Generated HERE)

These values do **not** exist in `features.py`.

They are computed at inference time:

- `xgb_proba` → XGBoost probability
- `anomaly_score` → IsolationForest anomaly score
- `ae_recon_error` → Autoencoder reconstruction error

These are **model outputs**, not engineered features.

Autoencoder Latent Vectors (Conceptual)

- `ae_latent_1` ... `ae_latent_8`

- Produced by the encoder forward pass
- Only included if they were used during training

They belong in `models.py`, not `features.py`.

Cluster ID (Critical Clarification)

- `cluster_id` is **not raw data**
- `cluster_id` is **not feature engineering**
- It is the output of a trained clustering model

Conceptually:

```
engineered_df → clustering_model.predict() → cluster_id
```

Status: Conceptually defined, but not fully wired or verified yet.

Meta-Feature Construction

Meta-features combine:

- Base model signals
- Cluster ID
- Selected engineered context features

They are assembled in **exact order** defined by `meta_features`.

Order mismatch = invalid predictions.

Final Prediction

- `predict_proba()` → returns fraud probability
- `predict()` → applies threshold → assigns label (`fraud` / `legit`)

This is the only place labels are assigned.

`validate_input()` (Misnamed, Transitional)

Current behavior: - Expects a FULLY ENGINEERED feature dict - Performs schema enforcement only - Does NOT accept raw input

Important: This function is architecturally wrong for inference and will be removed.

PART 4 — EXPLAINABILITY CONTRACT (`src/explain.py`)

Responsibility

Explain **why XGBoost produced its score** using SHAP.

It must explain **exactly what the model saw**.

What Works

- SHAP math is correct
- TreeExplainer usage is correct
- Feature attribution logic is correct

What Is Broken

Current flow incorrectly assumes engineered input:

```
input_dict → validate_input → SHAP
```

This violates: - Raw input rule - One-path rule - Feature source-of-truth rule

---

Correct Conceptual Flow (Future)

```
raw dict
  ↓
feature_pipeline
  ↓
engineered_df
  ↓
engineered_df[xgb_features]
  ↓
SHAP
```

No feature guessing. No re-engineering.

What `explain.py` MUST NOT Do

- No feature engineering
- No input validation
- No model inference beyond XGBoost
- No thresholding
- No decision logic

PART 5 — STEP 1 LIMITS (IMPORTANT)

STEP 1 does **NOT**:

- Design `prepare_features()`
- Decide file placement for inference glue
- Modify any `.py` file
- Remove `validate_input()` yet
- Add sanity scripts
- Touch APIs or deployment

This step exists **only** to freeze understanding.

FINAL STEP 1 VERDICT

- Understanding of current system: **FROZEN**
- Feature contracts: **CLEAR**
- Model responsibilities: **CLEAR**
- Explainability scope: **CLEAR**
- Missing integration point: **IDENTIFIED**

READY FOR STEP 2

Next step is to **design** `prepare_features()` **on paper**.

No code until that contract is frozen.