Low Level Design Document

This Low Level Design (LLD) document details the core implementation plan for **GridInsightPro - Smart Energy Demand Forecaster**. The system enables energy providers to upload, analyze, and forecast energy demand using big data and AI, supporting real-time analytics, anomaly detection, and collaborative planning.

---

1. System Components

Component	Technology	Key Responsibilities
Frontend	Next.js, React, D3.js	UI, data upload, visualization, scenario planning
Backend API	Python (FastAPI)	REST API, auth, orchestration, business logic
Data Processing	Apache Spark (GCP)	Distributed data validation, ETL, aggregation
ML/AI Service	Python (scikit-learn, TensorFlow)	Forecasting, anomaly detection
Storage	GCP BigQuery, Cloud Storage	Data, model, and file storage
Auth/Security	GCP IAM, OAuth2	Role-based access, audit logging, encryption
Collaboration	Backend, Frontend	Shared scenarios, comments, notifications

---

2. Class/Interface Overview

Class/Interface	Description	Key Methods/Attributes
`User`	System user, roles	`id`, `email`, `role`, `audit_log()`
`DataUploadHandler`	Handles file upload/validation	`upload()`, `validate()`, `store()`
`ForecastService`	Runs ML models, returns forecasts	`generate_forecast()`, `get_results()`
`AnomalyDetector`	Detects anomalies in data	`detect()`, `notify_users()`
`ScenarioManager`	Manages planning scenarios	`create()`, `share()`, `comment()`
`VisualizationAPI`	Serves chart data	`get_trends()`, `get_heatmap()`
`AuditLogger`	Tracks user/data actions	`log_action()`, `get_logs()`

Relationships:

- `User` interacts with `DataUploadHandler`, `ScenarioManager`, and `VisualizationAPI` via the frontend.
- `ForecastService` and `AnomalyDetector` operate on processed data and return results to the API.

---

3. Data Structure Overview

Model	Fields (Type)
User	id (UUID), email (str), role (enum), created_at (datetime)
ConsumptionData	id (UUID), region (str), timestamp (datetime), value (float), file_id (UUID)
Forecast	id (UUID), region (str), period (date/range), predicted_value (float), model_version (str)
Anomaly	id (UUID), data_id (UUID), type (str), severity (int), detected_at (datetime)
Scenario	id (UUID), name (str), owner_id (UUID), data_refs (list), comments (list)
AuditLog	id (UUID), user_id (UUID), action (str), target_id (UUID), timestamp (datetime)

---

4. Algorithms/Logic

Data Upload & Validation (Spark):

```
def validate_and_store(file):
    df = spark.read.csv(file)
    if df.size > 10GB: raise FileTooLarge
    if not schema_valid(df): raise SchemaError
    store_to_gcs(df)
    log_upload(user, file)
```

Forecast Generation:

```
def generate_forecast(region, period):
    data = fetch_consumption(region, period)
    model = load_model('demand_forecast')
    forecast = model.predict(data)
    store_forecast(forecast)
    return forecast
```

Anomaly Detection:

```
def detect_anomalies(data):
    anomalies = []
    for point in data:
        if is_outlier(point): anomalies.append(point)
    notify_users(anomalies)
    return anomalies
```

---

5. Error Handling

Scenario	Handling Approach

Invalid file format/size	Reject upload, return error message
Data validation failure	Log error, notify user, abort processing
ML model failure	Return fallback message, log for retraining
Unauthorized access	Return 403 Forbidden, log attempt
Storage/network errors	Retry with backoff, alert admin if persistent
Forecast accuracy < threshold	Flag for review, trigger model retraining

---

End of Document