Low level documentation for API

Overview

This documentation provides detailed technical specifications for the FastAPI-based backend service that powers the Dynamic Forecasting Visualization application. The service handles model training, prediction, drift detection, and monitoring.

System Architecture

Core Components

1. **FastAPI Application**: Main web service framework
2. **Forecasting Model**: Prophet-based time series forecasting
3. **Drift Detector**: Statistical concept drift detection
4. **MLflow Integration**: Model tracking and experimentation
5. **Prometheus Metrics**: System and application monitoring

Configuration

Key Configuration Parameters

Parameter	Description	Default Value
`BASE_DIR`	Application root directory	Path to parent directory
`MODEL_PATH`	Path to serialized model	`./models/model.pkl`
`METADATA_PATH`	Path to model metadata	`./models/metadata.json`
`SEASONAL_PERIODS`	Seasonality parameter	30
`PROMETHEUS_PORT`	Metrics endpoint port	8001
`MLFLOW_TRACKING_URI`	MLflow tracking server	`http://localhost:5001`
`MLFLOW_EXPERIMENT_NAME`	MLflow experiment name	`DemandForecasting`

State Management

Global Variables

Variable	Type	Description
`model`	`Optional[ForecastingModel]`	Loaded forecasting model
`df_global`	`Optional[pd.DataFrame]`	Loaded dataset
`drift_detector`	`Optional[DriftDetector]`	Drift detection instance

Metrics System

Prometheus Metrics

Endpoint Metrics

Metric Name	Type	Description	Labels
`model_init_requests_total`	Counter	Total /model_init requests	method, status_code
`model_init_request_latency_seconds`	Histogram	/model_init latency	method
`last_trained_date_requests_total`	Counter	Total /last_trained_date requests	method, status_code
`last_trained_date_request_latency_seconds`	Histogram	/last_trained_date latency	method
`predict_for_date_requests_total`	Counter	Total /predict_for_date requests	method, status_code
`predict_for_date_request_latency_seconds`	Histogram	/predict_for_date latency	method
`api_requests_per_ip_total`	Counter	Requests per client IP	client_ip, endpoint, method

System Metrics

Metric Name	Type	Description
`system_cpu_usage_percent`	Gauge	CPU utilization

system_memory_usage_percent	Gauge	Memory utilization
system_disk_usage_percent	Gauge	Disk utilization
system_network_bytes_sent	Gauge	Network bytes sent
system_network_bytes_recv	Gauge	Network bytes received
system_file_handles	Gauge	Open file handles

Middleware

EndpointMetricsMiddleware

- **Purpose**: Tracks request metrics for all endpoints
- **Behavior**:
  1. Records start time
  2. Processes request
  3. Updates relevant metrics:
     - Increments appropriate counter
     - Records latency
     - Tracks requests per IP

Core Functions

train_initial_model(df: pd.DataFrame)

- **Purpose**: Trains the first model instance
- **Parameters**:
  - `df`: Initial training data (first 6 months)
- **Behavior**:
  1. Trains Prophet model
  2. Saves model to disk
  3. Saves metadata

retrain_model(start_date, df, drift_metadata=None)

- **Purpose**: Retrains model with updated data
- **Parameters**:
  - `start_date`: Starting date for training data
  - `df`: Complete dataset
  - `drift_metadata`: Optional drift detection info
- **Behavior**:
  1. Subsets data up to `start_date`
  2. Starts MLflow run with appropriate tags
  3. Trains new model
  4. Logs drift parameters if applicable

`get_last_trained_date()`

- **Purpose**: Retrieves last training date from metadata
- **Returns**: `datetime` or `None`
- **Behavior**:
  1. Checks for metadata file
  2. Parses last trained date if exists

API Endpoints

`POST /model_init`

- **Purpose**: Initialize forecasting system
- **Parameters**:
  - `file`: CSV data file
  - `window_size`: Drift detection window
  - `threshold`: Drift detection threshold
- **Flow**:
  1. Loads and validates data

2. Initializes drift detector
3. Trains initial model
4. Returns success/error response

GET /last_trained_date

- **Purpose**: Get last model training date
- **Response**:

```
{
  "last_trained_date": "YYYY-MM-DD"
}
```

- **Error Handling**: Returns 404 if no model exists

GET /predict_for_date

- **Purpose**: Get prediction for specific date
- **Parameters**:
  - `current_date`: Date to predict (YYYY-MM-DD)
- **Response**:

```
{
  "date": "YYYY-MM-DD",
  "predicted": float,
  "actual": float,
  "error": float,
  "drift_detected": bool
}
```

- **Flow**:
  1. Validates date
  2. Generates prediction
  3. Checks for drift

4. Triggers retraining if drift detected

Startup Sequence

1. Configures MLflow tracking
2. Starts Prometheus metrics server
3. Launches system metrics collector thread
4. Loads existing model if available

Monitoring

- **Prometheus**: Scrapes metrics from `:8001`
- **MLflow**: Tracks all model training runs
- **System Metrics**: Collected every 5 seconds

Error Handling

- All endpoints return 500 status for unexpected errors
- Detailed error messages logged
- MLflow tracks failed runs

Dependencies

- FastAPI (web framework)
- Prophet (forecasting)
- Pandas (data processing)
- MLflow (experiment tracking)
- Prometheus Client (metrics)
- Psutil (system monitoring)