KYB Tool - Technical Architecture Document

Part 2: AI/ML Architecture, External Integrations, and Security

---

Document Information

- **Document Type**: Technical Architecture Document
- **Project**: KYB Tool - Enterprise-Grade Know Your Business Platform
- **Version**: 1.0
- **Date**: January 2025
- **Status**: Final Specification
- **Pages**: Part 2 of 3

---

4. AI/ML Architecture and Model Design

4.1 Machine Learning Pipeline Architecture

MLOps Infrastructure

```
┌─────────────────────────────────────────────────────────────
│                    ML Pipeline Architecture                    │
└─────────────────────────────────────────────────────────────

Data Sources
├──── Business Registry APIs
├──── Website Content Scraping
├──── Financial Data APIs
└──── News and Media APIs
    │
    ▼
Feature Engineering Pipeline (Apache Airflow)
├──── Data Validation & Cleaning
├──── Feature Extraction & Transformation
├──── Feature Store Updates (Feast)
└──── Data Quality Monitoring
    │
    ▼
Model Training & Evaluation (Kubernetes Jobs)
├──── Distributed Training (PyTorch)
├──── Hyperparameter Optimization (Optuna)
├──── Cross-Validation & Testing
└──── Model Versioning (MLflow)
    │
    ▼
Model Deployment Pipeline
├──── Model Validation & Testing
├──── A/B Testing Framework
├──── Canary Deployment
└──── Production Monitoring
    │
    ▼
Inference Services (TorchServe + FastAPI)
```

```
├─── Real-time Prediction APIs
├─── Batch Processing Jobs
├─── Model Performance Monitoring
└─── Drift Detection & Alerts
```

4.2 Business Classification Models

Primary Classification Model: Enhanced BERT Architecture

```
python
```

```python
import torch
import torch.nn as nn
from transformers import AutoModel, AutoTokenizer

class BusinessClassificationModel(nn.Module):
    """
    Multi-output classification model for MCC, NAICS, and SIC codes
    Uses pre-trained BERT with custom classification heads
    """
    def __init__(self, model_name='distilbert-base-uncased', num_mcc=1000,
                 num_naics=2000, num_sic=1000):
        super().__init__()

        # Base BERT model
        self.bert = AutoModel.from_pretrained(model_name)
        hidden_size = self.bert.config.hidden_size

        # Classification heads for different code systems
        self.mcc_classifier = nn.Sequential(
            nn.Linear(hidden_size, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, num_mcc)
        )

        self.naics_classifier = nn.Sequential(
            nn.Linear(hidden_size, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, num_naics)
        )

        self.sic_classifier = nn.Sequential(
            nn.Linear(hidden_size, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, num_sic)
        )

        # Confidence estimation head
        self.confidence_head = nn.Sequential(
            nn.Linear(hidden_size, 128),
            nn.ReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()  # Output 0-1 confidence score
        )

    def forward(self, input_ids, attention_mask):
        # Get BERT embeddings
        bert_output = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = bert_output.pooler_output

        # Generate predictions
        mcc_logits = self.mcc_classifier(pooled_output)
        naics_logits = self.naics_classifier(pooled_output)
        sic_logits = self.sic_classifier(pooled_output)
        confidence = self.confidence_head(pooled_output)

        return {
            'mcc_logits': mcc_logits,
            'naics_logits': naics_logits,
            'sic_logits': sic_logits,
            'confidence': confidence
        }

# Training configuration
TRAINING_CONFIG = {
    'model_name': 'distilbert-base-uncased',
    'batch_size': 32,
    'learning_rate': 2e-5,
    'epochs': 10,
    'warmup_steps': 1000,
    'weight_decay': 0.01,
    'max_length': 512,
    'gradient_accumulation': 4,
    'mixed_precision': True
}
```

Similarity-Based Classification (Fallback Model)

```
python
```

```python
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np

class SimilarityClassifier:
    """
    Embedding-based similarity classifier for handling edge cases
    Uses sentence transformers for semantic similarity
    """
    def __init__(self):
        self.encoder = SentenceTransformer('all-MiniLM-L6-v2')
        self.mcc_index = None
        self.naics_index = None
        self.code_mappings = {}

    def build_index(self, code_descriptions, code_type='mcc'):
        """Build FAISS index for fast similarity search"""
        embeddings = self.encoder.encode(code_descriptions)

        # Create FAISS index
        dimension = embeddings.shape[1]
        index = faiss.IndexFlatIP(dimension)  # Inner product similarity
        index.add(embeddings.astype('float32'))

        if code_type == 'mcc':
            self.mcc_index = index
        elif code_type == 'naics':
            self.naics_index = index

        return index

    def find_similar_codes(self, business_description, code_type='mcc', top_k=5):
        """Find most similar codes based on semantic similarity"""
        query_embedding = self.encoder.encode([business_description])

        index = self.mcc_index if code_type == 'mcc' else self.naics_index

        similarities, indices = index.search(query_embedding.astype('float32'), top_k)

        results = []
        for sim, idx in zip(similarities[0], indices[0]):
            results.append({
                'code': self.code_mappings[code_type][idx]['code'],
                'description': self.code_mappings[code_type][idx]['description'],
                'similarity_score': float(sim)
            })

        return results
```

4.3 Risk Assessment Models

Ensemble Risk Prediction Model

```
python
```

```
import xgboost as xgb
from sklearn.ensemble import IsolationForest
import torch
import torch.nn as nn

class RiskAssessmentEnsemble:
    """
    Ensemble model combining multiple approaches for comprehensive risk assessment
    """
    def __init__(self):
        self.xgboost_model = None
        self.lstm_model = None
        self.isolation_forest = None
        self.feature_importance = {}

    def initialize_models(self):
        # XGBoost for traditional risk factors
        self.xgboost_model = xgb.XGBRegressor(
            n_estimators=1000,
            max_depth=8,
            learning_rate=0.1,
            subsample=0.8,
            colsample_bytree=0.8,
            random_state=42
        )

        # LSTM for time-series risk prediction
        self.lstm_model = RiskLSTM(
            input_size=50,  # Number of features
            hidden_size=128,
            num_layers=2,
            output_size=1,
            dropout=0.2
        )

        # Isolation Forest for anomaly detection
        self.isolation_forest = IsolationForest(
            contamination=0.1,
            random_state=42,
            n_jobs=-1
        )

class RiskLSTM(nn.Module):
    """
    LSTM model for time-series risk prediction
    Predicts risk evolution over 3, 6, and 12-month horizons
    """
    def __init__(self, input_size, hidden_size, num_layers, output_size, dropout=0.2):
        super().__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            dropout=dropout,
            batch_first=True,
            bidirectional=True
        )

        # Multi-horizon prediction heads
        lstm_output_size = hidden_size * 2  # Bidirectional

        self.risk_head = nn.Sequential(
            nn.Linear(lstm_output_size, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, output_size)
        )

        # Separate heads for different time horizons
        self.prediction_3m = nn.Linear(lstm_output_size, 1)
        self.prediction_6m = nn.Linear(lstm_output_size, 1)
        self.prediction_12m = nn.Linear(lstm_output_size, 1)

        # Confidence interval estimation
        self.confidence_head = nn.Sequential(
            nn.Linear(lstm_output_size, 64),
            nn.ReLU(),
            nn.Linear(64, 2),  # Lower and upper bounds
            nn.Softplus()  # Ensure positive values
        )

    def forward(self, x):
        # x shape: (batch_size, sequence_length, input_size)
        lstm_out, (hidden, cell) = self.lstm(x)

        # Use the last time step
        last_output = lstm_out[:, -1, :]

        # Current risk score
        current_risk = self.risk_head(last_output)

        # Future predictions
        pred_3m = self.prediction_3m(last_output)
        pred_6m = self.prediction_6m(last_output)
        pred_12m = self.prediction_12m(last_output)

        # Confidence intervals
        confidence_bounds = self.confidence_head(last_output)

        return {
            'current_risk': current_risk,
            'prediction_3m': pred_3m,
            'prediction_6m': pred_6m,
            'prediction_12m': pred_12m,
            'confidence_lower': confidence_bounds[:, 0],
            'confidence_upper': confidence_bounds[:, 1]
        }

# Feature engineering for risk models
RISK_FEATURES = {
    'business_features': [
        'business_age_days',
        'registration_completeness_score',
        'website_quality_score',
        'social_media_presence_score',
        'customer_review_sentiment',
        'number_of_online_reviews',
        'business_address_type',  # Commercial, residential, PO Box
        'industry_risk_category',
        'seasonal_business_indicator'
    ],
    'financial_features': [
        'estimated_revenue_range',
        'payment_processing_history_length',
        'chargeback_ratio',
        'refund_ratio',
        'transaction_velocity',
        'average_transaction_size',
        'transaction_volume_trend',
        'credit_score_estimate'
    ],
    'compliance_features': [
        'sanctions_screening_result',
        'adverse_media_mentions',
        'regulatory_violations_count',
        'license_verification_status',
        'tax_compliance_status',
        'business_registration_validity'
    ],
    'external_features': [
        'economic_conditions_score',
        'industry_growth_rate',
        'regional_business_risk',
        'competitive_landscape_score',
        'market_saturation_level'
    ]
}
```

4.4 Fraud Detection Models

Graph Neural Network for Transaction Analysis

```
python
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_mean_pool

class TransactionGNN(nn.Module):
    """
    Graph Neural Network for analyzing transaction patterns and merchant networks
    Detects transaction laundering and suspicious merchant relationships
    """
    def __init__(self, node_features, edge_features, hidden_dim=128, num_layers=3):
        super().__init__()

        self.node_embedding = nn.Linear(node_features, hidden_dim)
        self.edge_embedding = nn.Linear(edge_features, hidden_dim)

        # Graph convolutional layers
        self.conv_layers = nn.ModuleList([
            GCNConv(hidden_dim, hidden_dim) for _ in range(num_layers)
        ])

        # Fraud detection head
        self.fraud_classifier = nn.Sequential(
            nn.Linear(hidden_dim, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )

        # Anomaly scoring head
        self.anomaly_scorer = nn.Sequential(
            nn.Linear(hidden_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

    def forward(self, x, edge_index, edge_attr, batch):
        # Node embeddings
        x = self.node_embedding(x)

        # Graph convolutions with residual connections
        for conv in self.conv_layers:
            x_new = conv(x, edge_index)
            x_new = F.relu(x_new)
            x = x + x_new  # Residual connection

        # Global graph representation
        graph_embedding = global_mean_pool(x, batch)

        # Predictions
        fraud_prob = self.fraud_classifier(graph_embedding)
        anomaly_score = self.anomaly_scorer(graph_embedding)

        return {
            'fraud_probability': fraud_prob,
            'anomaly_score': anomaly_score,
            'node_embeddings': x
        }
```

Synthetic Identity Detection

python

```python
from sklearn.ensemble import IsolationForest
from sklearn.cluster import DBSCAN
import numpy as np

class SyntheticIdentityDetector:
    """
    Detects synthetic business identities using multiple signals
    """
    def __init__(self):
        self.isolation_forest = IsolationForest(contamination=0.05, random_state=42)
        self.clustering_model = DBSCAN(eps=0.5, min_samples=5)
        self.identity_features = [
            'business_name_entropy',
            'address_authenticity_score',
            'phone_number_validity',
            'website_age_days',
            'domain_registration_consistency',
            'social_media_creation_date_gap',
            'business_registration_speed',
            'document_consistency_score'
        ]

    def detect_synthetic_identity(self, business_data):
        """
        Returns probability that business identity is synthetic
        """
        features = self.extract_identity_features(business_data)

        # Anomaly detection
        anomaly_score = self.isolation_forest.decision_function([features])[0]

        # Clustering-based detection
        cluster_label = self.clustering_model.fit_predict([features])[0]
        is_outlier = cluster_label == -1

        # Combined score
        synthetic_probability = self.combine_scores(anomaly_score, is_outlier)

        return {
            'synthetic_probability': synthetic_probability,
            'anomaly_score': anomaly_score,
            'is_cluster_outlier': is_outlier,
            'risk_factors': self.identify_risk_factors(features, business_data)
        }
```

---

5. External API Integrations

5.1 Data Source Integration Strategy

Priority 1: Free/Open Source APIs

```
python
```

```python
# Business Registry APIs (Free/Government)
BUSINESS_REGISTRY_APIS = {
    'us_secretary_of_state': {
        'states': ['CA', 'NY', 'TX', 'FL', 'IL', 'PA', 'OH', 'GA', 'NC', 'MI'],
        'rate_limit': '100/hour',
        'cost': 'Free',
        'data_quality': 'High',
        'implementation': 'Direct HTTP API'
    },
    'canada_corporations': {
        'url': 'https://ised-isde.canada.ca/cc/lgcy/fdrlCrpSrch.html',
        'provinces': 'All Canadian provinces',
        'rate_limit': '50/hour',
        'cost': 'Free',
        'data_quality': 'High'
    },
    'uk_companies_house': {
        'url': 'https://developer.company-information.service.gov.uk/',
        'rate_limit': '600/5min',
        'cost': 'Free',
        'data_quality': 'High',
        'api_key_required': True
    },
    'opencorporates': {
        'url': 'https://api.opencorporates.com/',
        'coverage': '130+ jurisdictions',
        'rate_limit': '500/month (free)',
        'cost': 'Free tier available',
        'data_quality': 'Medium-High'
    }
}

# Sanctions and Watchlists (Free/Government)
```

```
SANCTIONS_APIS = {
    'ofac_sdn_list': {
        'url': 'https://www.treasury.gov/ofac/downloads/sdn.xml',
        'update_frequency': 'Weekly',
        'cost': 'Free',
        'format': 'XML',
        'implementation': 'Download and parse'
    },
    'un_sanctions': {
        'url': 'https://scsanctions.un.org/resources/',
        'update_frequency': 'Daily',
        'cost': 'Free',
        'format': 'XML/JSON'
    },
    'eu_sanctions': {
        'url': 'https://ec.europa.eu/info/business-economy-euro/banking-and-finance/international-relations/re
        'update_frequency': 'Weekly',
        'cost': 'Free',
        'format': 'XML'
    }
}

# Website Analysis (Free/Open Source)
WEBSITE_ANALYSIS_TOOLS = {
    'playwright': {
        'purpose': 'Website scraping and screenshots',
        'cost': 'Free (open source)',
        'capabilities': 'Full browser automation',
        'implementation': 'Self-hosted'
    },
    'lighthouse': {
        'purpose': 'Website performance and SEO analysis',
        'cost': 'Free (open source)',
        'capabilities': 'Performance, accessibility, SEO scoring',
        'implementation': 'Chrome headless'
    },
    'wayback_machine': {
        'url': 'https://archive.org/wayback/available',
        'purpose': 'Historical website data',
        'cost': 'Free',
        'rate_limit': '1000/day'
    }
}
```

Priority 2: Low-Cost Premium APIs

```python
```

```
# Financial and Credit Data (Low cost)
PREMIUM_DATA_SOURCES = {
    'dun_bradstreet_api': {
        'cost': '$0.10-0.50 per lookup',
        'data_quality': 'Very High',
        'coverage': 'Global',
        'use_case': 'Enterprise customers only',
        'integration_priority': 'Phase 2'
    },
    'experian_business_api': {
        'cost': '$0.15-0.75 per lookup',
        'data_quality': 'Very High',
        'coverage': 'US, UK, some international',
        'use_case': 'Premium features',
        'integration_priority': 'Phase 2'
    },
    'clearbit_api': {
        'cost': '$0.20 per enrichment',
        'data_quality': 'High',
        'coverage': 'Global tech companies',
        'use_case': 'Website and company enrichment',
        'integration_priority': 'Phase 3'
    }
}

# News and Media Monitoring
NEWS_APIS = {
    'newsapi_org': {
        'cost': 'Free tier: 1000/day, $449/month for business',
        'coverage': '80,000+ news sources',
        'languages': '14 languages',
        'use_case': 'Adverse media monitoring'
    },
    'gdelt_project': {
        'url': 'https://api.gdeltproject.org/',
        'cost': 'Free',
        'coverage': 'Global news and events',
        'use_case': 'Free alternative for adverse media',
        'data_format': 'JSON/CSV'
    }
}
```

5.2 Integration Architecture

API Integration Service (Go)

```
go
```

```go
package integrations

import (
    "context"
    "net/http"
    "time"
    "sync"
)

type IntegrationManager struct {
    httpClient     *http.Client
    rateLimiters   map[string]*RateLimiter
    circuitBreakers map[string]*CircuitBreaker
    cache          CacheInterface
    metrics        MetricsInterface
}

type DataProvider interface {
    GetBusinessInfo(ctx context.Context, query BusinessQuery) (*BusinessInfo, error)
    GetSanctionsInfo(ctx context.Context, entity EntityInfo) (*SanctionsResult, error)
    GetFinancialInfo(ctx context.Context, businessID string) (*FinancialInfo, error)
    GetNewsData(ctx context.Context, companyName string) (*NewsResult, error)
}

// Rate limiting implementation
type RateLimiter struct {
    requests   chan struct{}
    refillRate time.Duration
    capacity   int
    tokens     int
    lastRefill time.Time
    mutex      sync.Mutex
}
```

```go
func (r *RateLimiter) Allow() bool {
    r.mutex.Lock()
    defer r.mutex.Unlock()

    // Refill tokens based on time passed
    now := time.Now()
    tokensToAdd := int(now.Sub(r.lastRefill) / r.refillRate)
    r.tokens = min(r.capacity, r.tokens + tokensToAdd)
    r.lastRefill = now

    if r.tokens > 0 {
        r.tokens--
        return true
    }

    return false
}

// Circuit breaker pattern for external API resilience
type CircuitBreaker struct {
    maxRequests uint32
    interval    time.Duration
    timeout     time.Duration

    mutex       sync.Mutex
    state       State
    generation  uint64
    counts      *Counts
    expiry      time.Time
}

// Business registry integration example
```

```
type SecretaryOfStateAPI struct {
    baseURL     string
    apiKey      string
    rateLimiter *RateLimiter
    client      *http.Client
}

func (s *SecretaryOfStateAPI) GetBusinessInfo(ctx context.Context, query BusinessQuery) (*BusinessInfo,
    // Check rate limit
    if !s.rateLimiter.Allow() {
        return nil, ErrRateLimited
    }

    // Build request
    req, err := s.buildBusinessSearchRequest(query)
    if err != nil {
        return nil, err
    }

    // Execute with timeout
    resp, err := s.client.Do(req.WithContext(ctx))
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    // Parse response
    return s.parseBusinessResponse(resp)
}
```

Data Quality and Validation

```
python
```

```python
class DataQualityValidator:
    """
    Validates and scores data quality from external sources
    """
    def __init__(self):
        self.validation_rules = {
            'business_name': {
                'min_length': 2,
                'max_length': 200,
                'allowed_chars': r'[a-zA-Z0-9\s\-\.\,\&]',
                'required': True
            },
            'tax_id': {
                'formats': {
                    'US': r'^\d{2}-\d{7}$',  # EIN format
                    'CA': r'^\d{9}RT\d{4}$'  # Canadian business number
                },
                'required': False
            },
            'website_url': {
                'format': r'^https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}',
                'required': False,
                'validate_accessibility': True
            }
        }

    def validate_business_data(self, data: dict, source: str) -> dict:
        """
        Validates business data and returns quality score
        """
        validation_results = {
            'overall_score': 0.0,
            'field_scores': {},
            'missing_fields': [],
            'invalid_fields': [],
            'warnings': []
        }

        total_weight = 0
        weighted_score = 0

        for field, rules in self.validation_rules.items():
            weight = rules.get('weight', 1.0)
            total_weight += weight

            if field not in data or data[field] is None:
                if rules.get('required', False):
                    validation_results['missing_fields'].append(field)
                    # Required field missing = 0 score
                    field_score = 0.0
                else:
                    # Optional field missing = neutral
                    continue
            else:
                field_score = self.validate_field(data[field], rules, field)

            validation_results['field_scores'][field] = field_score
            weighted_score += field_score * weight

        validation_results['overall_score'] = weighted_score / total_weight if total_weight > 0 else 0.0
        validation_results['source_reliability'] = self.get_source_reliability(source)

        return validation_results

    def get_source_reliability(self, source: str) -> float:
        """
    Returns reliability score for different data sources
    """
    source_reliability = {
        'secretary_of_state': 0.95,
        'companies_house': 0.95,
        'opencorporates': 0.80,
        'dun_bradstreet': 0.90,
        'experian': 0.90,
        'website_scraping': 0.70,
        'social_media': 0.60
    }

    return source_reliability.get(source, 0.50)
```

---

6. Security Architecture and Compliance

6.1 Security-by-Design Architecture

Zero-Trust Network Architecture

```
┌──────────────────────────────────────────────────────────────────
│                Security Architecture                  │
└──────────────────────────────────────────────────────────────────

Edge Security Layer
├──── WAF (Web Application Firewall)
├──── DDoS Protection
├──── Rate Limiting & Bot Detection
└──── Geographic IP Filtering
  │
  ▼
Identity & Access Management (IAM)
├──── Multi-Factor Authentication (MFA)
├──── JWT Token Management
├──── API Key Authentication
├──── Role-Based Access Control (RBAC)
└──── Session Management
  │
  ▼
Network Security Layer (Service Mesh - Istio)
├──── mTLS Between Services
├──── Network Policies & Segmentation
├──── Traffic Encryption (TLS 1.3)
├──── Certificate Management (Cert-Manager)
└──── Service-to-Service Authentication
  │
  ▼
Application Security Layer
├──── Input Validation & Sanitization
├──── SQL Injection Prevention
├──── XSS Protection
├──── CSRF Protection
└──── Secure Headers (CSP, HSTS, etc.)
```

```
    |
    ▼
Data Security Layer
 ├─── AES-256 Encryption at Rest
 ├─── TLS 1.3 Encryption in Transit
 ├─── Field-Level Encryption (PII/PCI)
 ├─── Key Management (HashiCorp Vault)
 └─── Data Loss Prevention (DLP)
    |
    ▼
Infrastructure Security Layer
 ├─── Container Security (Falco, OPA Gatekeeper)
 ├─── Kubernetes Security Policies
 ├─── Vulnerability Scanning (Trivy, Clair)
 ├─── Runtime Security Monitoring
 └─── Infrastructure as Code Security
```

6.2 Compliance Framework Implementation

SOC 2 Type II Compliance Controls

```
yaml
```

```yaml
Security Controls (CC6):
  CC6.1_Logical_Access:
    implementation: "Multi-factor authentication, role-based access control"
    evidence: "Access logs, user provisioning records, MFA enrollment"
    automated_testing: "Daily access review reports"

  CC6.2_Authentication:
    implementation: "JWT tokens, API key rotation, session management"
    evidence: "Authentication logs, token expiration policies"
    automated_testing: "Token validation and expiration tests"

  CC6.3_Authorization:
    implementation: "RBAC with principle of least privilege"
    evidence: "Role definitions, permission matrices, access reviews"
    automated_testing: "Automated privilege escalation detection"

Availability Controls (CC7):
  CC7.1_Infrastructure:
    implementation: "Multi-region deployment, auto-scaling, load balancing"
    evidence: "Uptime metrics, incident response logs"
    automated_testing: "Health checks, failover testing"

  CC7.2_Monitoring:
    implementation: "24/7 monitoring, alerting, incident response"
    evidence: "Monitoring dashboards, alert configurations, response times"
    automated_testing: "Synthetic monitoring, canary deployments"

Processing_Integrity (CC8):
  CC8.1_Data_Processing:
    implementation: "Input validation, checksums, audit trails"
    evidence: "Processing logs, data integrity checks"
    automated_testing: "Data validation tests, integrity monitoring"
```

```
Confidentiality (CC7):
 CC7.1_Data_Protection:
  implementation: "Encryption at rest/transit, access controls"
  evidence: "Encryption configurations, key management logs"
  automated_testing: "Encryption validation, key rotation tests"
```

PCI DSS Compliance for Payment Data

```
python
```

```python
class PCIComplianceManager:
    """
    Manages PCI DSS compliance requirements for handling payment card data
    """
    def __init__(self):
        self.encryption_key_manager = KeyManager()
        self.audit_logger = AuditLogger()
        self.access_controller = AccessController()

    def tokenize_sensitive_data(self, data: dict) -> dict:
        """
        Tokenizes sensitive payment data (PCI DSS Requirement 3)
        """
        sensitive_fields = ['card_number', 'cvv', 'bank_account']

        tokenized_data = data.copy()
        for field in sensitive_fields:
            if field in data and data[field]:
                # Generate secure token
                token = self.generate_secure_token()
                # Store mapping securely
                self.store_token_mapping(token, data[field])
                # Replace with token
                tokenized_data[field] = token

                # Log tokenization event
                self.audit_logger.log_data_tokenization(field, token)

        return tokenized_data

    def validate_network_security(self) -> bool:
        """
        Validates network security controls (PCI DSS Requirement 1 & 2)
        """
        security_checks = [
            self.check_firewall_configuration(),
            self.validate_default_passwords_changed(),
            self.verify_network_segmentation(),
            self.check_wireless_security(),
            self.validate_vulnerability_management()
        ]

        return all(security_checks)

    def generate_compliance_report(self) -> dict:
        """
        Generates PCI DSS compliance assessment report
        """
        return {
            'assessment_date': datetime.utcnow(),
            'requirements_status': self.assess_all_requirements(),
            'vulnerabilities': self.scan_for_vulnerabilities(),
            'remediation_needed': self.identify_remediation_items(),
            'next_assessment_due': self.calculate_next_assessment_date()
        }

# PCI DSS Requirements Implementation
PCI_DSS_REQUIREMENTS = {
    'requirement_1': {
        'description': 'Install and maintain firewall configuration',
        'implementation': 'Kubernetes NetworkPolicies + Cloud WAF',
        'validation': 'Automated firewall rule testing'
    },
    'requirement_2': {
        'description': 'Do not use vendor-supplied defaults',
        'implementation': 'Configuration management + security hardening',
        'validation': 'Baseline configuration scanning'
    },
    'requirement_3': {
        'description': 'Protect stored cardholder data',
        'implementation': 'AES-256 encryption + tokenization',
        'validation': 'Encryption validation + key rotation testing'
    },
    'requirement_4': {
        'description': 'Encrypt transmission of cardholder data',
        'implementation': 'TLS 1.3 + certificate management',
        'validation': 'SSL/TLS configuration testing'
    }
    # ... Additional requirements
}
```

GDPR Privacy Controls

```
python
```

```python
class GDPRPrivacyManager:
    """
    Manages GDPR compliance for EU personal data processing
    """
    def __init__(self):
        self.consent_manager = ConsentManager()
        self.data_processor = PersonalDataProcessor()
        self.retention_manager = DataRetentionManager()

    def process_data_subject_request(self, request_type: str, subject_id: str) -> dict:
        """
        Handles GDPR data subject requests (Articles 15-22)
        """
        request_handlers = {
            'access': self.handle_access_request,
            'rectification': self.handle_rectification_request,
            'erasure': self.handle_erasure_request,
            'portability': self.handle_portability_request,
            'restriction': self.handle_restriction_request,
            'objection': self.handle_objection_request
        }

        if request_type not in request_handlers:
            raise ValueError(f"Invalid request type: {request_type}")

        # Log the request
        self.audit_logger.log_data_subject_request(request_type, subject_id)

        # Process the request
        result = request_handlers[request_type](subject_id)

        # Update compliance records
        self.update_compliance_record(subject_id, request_type, result)
        return result

    def assess_data_processing_lawfulness(self, processing_activity: str) -> dict:
        """
        Assesses lawfulness of data processing under GDPR Article 6
        """
        lawful_bases = {
            'consent': 'User has given clear consent',
            'contract': 'Processing necessary for contract performance',
            'legal_obligation': 'Processing required by law',
            'vital_interests': 'Processing necessary to protect vital interests',
            'public_task': 'Processing necessary for public interest task',
            'legitimate_interests': 'Legitimate interests not overridden by data subject rights'
        }

        # Determine applicable lawful basis
        applicable_basis = self.determine_lawful_basis(processing_activity)

        return {
            'processing_activity': processing_activity,
            'lawful_basis': applicable_basis,
            'basis_description': lawful_bases.get(applicable_basis),
            'documentation_required': self.get_documentation_requirements(applicable_basis),
            'consent_required': applicable_basis == 'consent'
        }

# GDPR Data Processing Record
GDPR_PROCESSING_ACTIVITIES = {
    'business_verification': {
        'purpose': 'Know Your Business compliance and risk assessment',
        'lawful_basis': 'legal_obligation',
        'data_categories': ['business_contact_info', 'business_registration_data'],
        'data_sources': ['customer_input', 'public_registries'],
        'retention_period': '7_years_post_relationship',
        'international_transfers': ['US', 'Canada'],
        'safeguards': ['adequacy_decision', 'standard_contractual_clauses']
    },
    'risk_assessment': {
        'purpose': 'Financial risk evaluation and fraud prevention',
        'lawful_basis': 'legitimate_interests',
        'data_categories': ['business_financial_data', 'transaction_patterns'],
        'automated_decision_making': True,
        'profiling': True,
        'retention_period': '5_years_post_assessment'
    }
}
```

This completes Part 2 of the Technical Architecture document, covering comprehensive AI/ML model architectures, external API integration strategies, and detailed security/compliance frameworks.

Should I proceed with **Part 3: Performance Optimization, Infrastructure, and Deployment Architecture**?