KYB Tool - Feature Specifications Document

Part 3: Advanced Features, Analytics, and Integration Requirements

---

Document Information

- **Document Type**: Feature Specifications Document
- **Project**: KYB Tool - Enterprise-Grade Know Your Business Platform
- **Version**: 1.0
- **Date**: January 2025
- **Status**: Final Specification
- **Pages**: Part 3 of 4

---

7. Epic 6: Advanced Analytics and Reporting

7.1 Epic Overview

Epic Description: Comprehensive analytics platform providing business intelligence, performance metrics, risk trends, and customizable reporting capabilities for data-driven decision making.

Business Value: Enables proactive risk management, improves operational efficiency by 40%, and provides regulatory-ready reports that reduce compliance overhead.

Success Metrics:

- Report generation time: <30 seconds for complex reports
- Data freshness: Real-time for operational metrics, <1 hour for analytical data

- User adoption: >80% of users access analytics monthly
- Custom report creation: <10 minutes average time

7.2 User Stories

Story 6.1: Real-time Operational Dashboard

As a operations manager
I want real-time visibility into KYB processing performance and metrics
So that I can monitor system health and team productivity

Acceptance Criteria:

```gherkin
```

Given I have dashboard access permissions
When I view the operations dashboard
Then I should see real-time metrics updating every 30 seconds
And I should see processing volume trends over time
And I should see team performance metrics and SLA compliance
And I should see system health indicators and error rates
And I should be able to drill down into specific metrics
And I should be able to set custom alerts for key thresholds
And dashboard should load within 3 seconds

Scenario: Processing volume monitoring
Given the system is processing merchant applications
When I view the operations dashboard
Then I see current hourly/daily/weekly processing volumes
And I see comparison to historical averages
And I see breakdown by business type and risk level
And I see processing time distribution charts
And alerts trigger if volumes drop below thresholds

Scenario: Team performance tracking
Given multiple users are processing reviews
When I view team metrics
Then I see individual and team productivity metrics
And I see average review times by user
And I see quality scores based on audit results
And I can identify bottlenecks and training needs
And metrics respect privacy settings and permissions

Dashboard Components Specification:

typescript

```
// Real-time Operations Dashboard Components
interface OperationsDashboard {
  // Key Performance Indicators
  kpiCards: {
    totalProcessedToday: {
      value: number;
      trend: TrendIndicator;
      comparison: ComparisonPeriod;
    };
    averageProcessingTime: {
      value: string; // "4.2 minutes"
      trend: TrendIndicator;
      slaStatus: 'meeting' | 'at-risk' | 'breached';
    };
    approvalRate: {
      value: number; // percentage
      trend: TrendIndicator;
      breakdown: ApprovalBreakdown;
    };
    highRiskMerchantsCount: {
      value: number;
      trend: TrendIndicator;
      urgentCount: number;
    };
  };

  // Real-time Charts
  charts: {
    processingVolumeTimeline: {
      type: 'line';
      timeRange: '24h' | '7d' | '30d';
      granularity: 'hour' | 'day';
      data: ProcessingVolumePoint[];
```

```
    };
    riskDistribution: {
      type: 'donut';
      data: RiskLevelDistribution;
    };
    processingTimeDistribution: {
      type: 'histogram';
      data: ProcessingTimeDistribution;
    };
    teamPerformance: {
      type: 'bar';
      data: TeamPerformanceData[];
    };
  };

  // System Health Indicators
  systemHealth: {
    apiResponseTime: HealthMetric;
    databasePerformance: HealthMetric;
    mlModelPerformance: HealthMetric;
    externalApiStatus: HealthMetric[];
  };

  // Alerts and Notifications
  activeAlerts: Alert[];
  alertConfiguration: AlertConfig[];
}

interface ProcessingVolumePoint {
  timestamp: string;
  processed: number;
  approved: number;
  rejected: number;
```

```
  pending: number;
}

interface TeamPerformanceData {
  userId: string;
  userName: string;
  reviewsCompleted: number;
  averageReviewTime: number;
  qualityScore: number;
  productivityRank: number;
}

// Real-time data service
class RealTimeDashboardService {
  private websocket: WebSocket;
  private updateInterval: number = 30000; // 30 seconds

  constructor(private dashboardStore: DashboardStore) {
    this.initializeWebSocket();
    this.startPeriodicUpdates();
  }

  private initializeWebSocket() {
    this.websocket = new WebSocket('/ws/dashboard');

    this.websocket.onmessage = (event) => {
      const update = JSON.parse(event.data);
      this.handleRealTimeUpdate(update);
    };
  }

  private handleRealTimeUpdate(update: DashboardUpdate) {
    switch (update.type) {
```

```
      case 'processing_volume':
        this.dashboardStore.updateProcessingVolume(update.data);
        break;
      case 'team_metrics':
        this.dashboardStore.updateTeamMetrics(update.data);
        break;
      case 'system_health':
        this.dashboardStore.updateSystemHealth(update.data);
        break;
      case 'alert':
        this.dashboardStore.addAlert(update.data);
        break;
    }
  }
}
```

Story 6.2: Risk Analytics and Trends

As a risk manager
I want comprehensive risk analytics showing portfolio risk trends
So that I can make data-driven decisions about risk appetite and controls

Acceptance Criteria:

```gherkin
```

Given I have risk analytics access
When I view the risk analytics dashboard
Then I should see portfolio risk distribution over time
And I should see risk trends by industry, geography, and business size
And I should see predictive risk models and forecasting
And I should see correlation analysis between risk factors
And I should see early warning indicators for portfolio deterioration
And I should be able to create custom risk reports
And analytics should support drill-down to individual merchants

Scenario: Portfolio risk monitoring
Given a portfolio of 1000+ merchants
When I view risk trends
Then I see risk score distribution changes over time
And I see breakdown by risk categories (operational, financial, etc.)
And I see geographic and industry risk concentrations
And I see early warning indicators for emerging risks
And I can set alerts for significant risk changes

Scenario: Predictive risk analysis
Given historical risk and performance data
When I access predictive analytics
Then I see 3, 6, and 12-month portfolio risk forecasts
And I see scenarios for different market conditions
And I see recommendations for portfolio optimization
And I can model impact of policy changes
And forecasts include confidence intervals

Risk Analytics Implementation:

python

```python
class RiskAnalyticsEngine:
    """
    Advanced risk analytics and trend analysis
    """

    def __init__(self):
        self.time_series_analyzer = TimeSeriesAnalyzer()
        self.correlation_analyzer = CorrelationAnalyzer()
        self.predictive_modeler = PredictiveRiskModeler()
        self.trend_detector = TrendDetector()

    async def generate_portfolio_analysis(self,
                                          portfolio_filters: dict,
                                          analysis_period: str) -> PortfolioAnalysis:
        """
        Generate comprehensive portfolio risk analysis
        """
        # Fetch portfolio data
        portfolio_data = await self.fetch_portfolio_data(
            portfolio_filters, analysis_period
        )

        # Parallel analysis tasks
        analysis_tasks = [
            self.analyze_risk_distribution(portfolio_data),
            self.analyze_risk_trends(portfolio_data),
            self.analyze_industry_concentrations(portfolio_data),
            self.analyze_geographic_concentrations(portfolio_data),
            self.generate_risk_forecasts(portfolio_data),
            self.detect_emerging_risks(portfolio_data)
        ]

        results = await asyncio.gather(*analysis_tasks)
```

```
    return PortfolioAnalysis(
        risk_distribution=results[0],
        risk_trends=results[1],
        industry_analysis=results[2],
        geographic_analysis=results[3],
        risk_forecasts=results[4],
        emerging_risks=results[5],
        generated_at=datetime.utcnow(),
        data_freshness=self.calculate_data_freshness(portfolio_data)
    )

async def analyze_risk_trends(self, portfolio_data: list) -> RiskTrendAnalysis:
    """
    Analyze portfolio risk trends over time
    """
    # Group data by time periods
    monthly_data = self.group_by_month(portfolio_data)

    trend_analysis = {}

    for risk_category in ['overall', 'operational', 'financial', 'regulatory']:
        # Calculate monthly averages
        monthly_scores = [
            np.mean([merchant[f'{risk_category}_risk'] for merchant in month_data])
            for month_data in monthly_data.values()
        ]

        # Trend detection
        trend = self.trend_detector.detect_trend(monthly_scores)

        # Statistical significance
        significance = self.calculate_trend_significance(monthly_scores)
```

```
        trend_analysis[risk_category] = {
            'monthly_scores': monthly_scores,
            'trend_direction': trend.direction,  # 'increasing', 'decreasing', 'stable'
            'trend_strength': trend.strength,    # 0.0 to 1.0
            'statistical_significance': significance,
            'key_drivers': await self.identify_trend_drivers(
                portfolio_data, risk_category, trend
            )
        }

    return RiskTrendAnalysis(
        analysis_period=len(monthly_data),
        category_trends=trend_analysis,
        overall_trend_score=self.calculate_overall_trend_score(trend_analysis),
        recommendations=await self.generate_trend_recommendations(trend_analysis)
    )

async def generate_risk_forecasts(self, portfolio_data: list) -> RiskForecasts:
    """
    Generate predictive risk forecasts
    """
    forecasting_models = {
        '3_month': self.predictive_modeler.get_short_term_model(),
        '6_month': self.predictive_modeler.get_medium_term_model(),
        '12_month': self.predictive_modeler.get_long_term_model()
    }

    forecasts = {}

    for horizon, model in forecasting_models.items():
        # Prepare features for prediction
        features = await self.prepare_forecasting_features(
```

```
            portfolio_data, horizon
        )

        # Generate base forecast
        base_forecast = await model.predict(features)

        # Generate scenario forecasts
        scenarios = await self.generate_scenario_forecasts(
            model, features, horizon
        )

        # Calculate confidence intervals
        confidence_intervals = await self.calculate_confidence_intervals(
            model, features, base_forecast
        )

        forecasts[horizon] = {
            'base_forecast': base_forecast,
            'scenarios': scenarios,
            'confidence_intervals': confidence_intervals,
            'key_assumptions': model.get_key_assumptions(),
            'forecast_accuracy_history': await self.get_model_accuracy_history(model)
        }

    return RiskForecasts(
        forecasts=forecasts,
        model_metadata=self.get_model_metadata(),
        generated_at=datetime.utcnow()
    )

# Risk analytics data models
@dataclass
class RiskTrendAnalysis:
```

```
    analysis_period: int
    category_trends: dict
    overall_trend_score: float
    recommendations: List[TrendRecommendation]

@dataclass
class PortfolioRiskDistribution:
    low_risk_count: int
    medium_risk_count: int
    high_risk_count: int
    critical_risk_count: int
    distribution_changes: dict  # Changes from previous period
    risk_concentration_score: float  # Measure of risk clustering

@dataclass
class EmergingRisk:
    risk_type: str
    severity: str  # 'low', 'medium', 'high', 'critical'
    affected_merchant_count: int
    trend_strength: float
    description: str
    recommended_actions: List[str]
    monitoring_frequency: str
```

Story 6.3: Custom Reporting and Export

As a compliance officer
I want to create custom reports with specific data fields and filters
So that I can generate regulatory reports and board presentations

Acceptance Criteria:

```
gherkin
```

Given I have reporting access permissions
When I access the report builder
Then I should be able to select data sources and fields
And I should be able to apply filters and date ranges
And I should be able to choose from multiple output formats
And I should be able to schedule recurring reports
And I should be able to save report templates for reuse
And generated reports should be audit-ready with timestamps and signatures
And report generation should complete within 5 minutes for large datasets

Scenario: Custom regulatory report creation
Given I need to create a monthly risk assessment report
When I use the report builder
Then I can select relevant merchants and time periods
And I can include risk scores, classification data, and audit trails
And I can format the report for regulatory submission
And I can schedule automatic generation on the last day of each month
And reports are stored securely with access controls

Scenario: Executive dashboard export
Given I need to present portfolio metrics to the board
When I create an executive summary report
Then I can include high-level KPIs and trend charts
And I can export to PowerPoint format with branded templates
And charts should be high-resolution and printer-friendly
And data should include appropriate executive-level summaries

---

8. Epic 7: SDK and Integration Ecosystem

8.1 Epic Overview

Epic Description: Comprehensive SDK ecosystem supporting multiple programming languages with code examples, integration guides, and developer tools to accelerate customer adoption.

Business Value: Reduces customer integration time by 70%, increases developer satisfaction, and enables rapid market expansion through easier technical adoption.

Success Metrics:

- SDK download and usage: >1,000 active integrations within 12 months
- Integration success rate: >90% of customers successfully integrate within 1 week
- Developer documentation satisfaction: >4.7/5.0 rating
- Support ticket reduction: 50% fewer integration-related tickets

8.2 User Stories

Story 7.1: Python SDK Development

As a Python developer
I want a well-designed Python SDK for KYB services
So that I can integrate KYB functionality with minimal code

Acceptance Criteria:

```gherkin
```

```
Given I am developing a Python application
When I install the KYB Python SDK
Then I should be able to authenticate and make API calls with minimal setup
And the SDK should handle authentication, retries, and error handling automatically
And I should have access to all API endpoints through intuitive methods
And the SDK should support both synchronous and asynchronous operations
And comprehensive documentation and examples should be included
And the SDK should follow Python best practices and PEP standards

Scenario: Simple business classification
Given I have installed the kyb-python package
When I write classification code using the SDK
Then I can classify a business with 3-5 lines of code
And error handling is built into the SDK methods
And response objects have convenient properties for accessing data
And the SDK automatically handles pagination for list operations

Scenario: Advanced integration with custom settings
Given I need enterprise-level configuration
When I initialize the SDK with custom settings
Then I can configure timeouts, retry policies, and logging
And I can use custom authentication methods
And I can set up webhook verification helpers
And the SDK integrates well with popular Python frameworks
```

Python SDK Implementation:

```python
```

```python
# KYB Python SDK - Core Implementation
from typing import Optional, List, Dict, Union, AsyncIterator
import asyncio
import aiohttp
import requests
from dataclasses import dataclass
from datetime import datetime, timedelta

class KYBClient:
    """
    Main client for KYB API interactions
    """

    def __init__(self,
                 api_key: str,
                 base_url: str = "https://api.kybtool.com",
                 timeout: int = 30,
                 max_retries: int = 3,
                 retry_delay: float = 1.0):
        self.api_key = api_key
        self.base_url = base_url.rstrip('/')
        self.timeout = timeout
        self.max_retries = max_retries
        self.retry_delay = retry_delay

        self._session = self._create_session()

        # Initialize service modules
        self.classify = ClassificationService(self)
        self.risk = RiskAssessmentService(self)
        self.businesses = BusinessService(self)
        self.compliance = ComplianceService(self)
        self.webhooks = WebhookService(self)
```

```python
def _create_session(self) -> requests.Session:
    """Create configured requests session"""
    session = requests.Session()
    session.headers.update({
        'Authorization': f'Bearer {self.api_key}',
        'Content-Type': 'application/json',
        'User-Agent': f'kyb-python-sdk/{self._get_version()}'
    })
    session.timeout = self.timeout
    return session

def _make_request(self, method: str, endpoint: str, **kwargs) -> Dict:
    """Make HTTP request with automatic retries"""
    url = f"{self.base_url}/api/v1{endpoint}"

    for attempt in range(self.max_retries + 1):
        try:
            response = self._session.request(method, url, **kwargs)

            if response.status_code == 429:  # Rate limited
                retry_after = int(response.headers.get('Retry-After', self.retry_delay))
                if attempt < self.max_retries:
                    time.sleep(retry_after)
                    continue

            response.raise_for_status()
            return response.json()

        except requests.RequestException as e:
            if attempt == self.max_retries:
                raise KYBAPIException(f"Request failed after {self.max_retries} retries: {e}")
            time.sleep(self.retry_delay * (2 ** attempt))  # Exponential backoff
```

```python
class ClassificationService:
    """Business classification operations"""

    def __init__(self, client: KYBClient):
        self.client = client

    def classify_business(self,
                          business_description: str,
                          business_name: Optional[str] = None,
                          website_url: Optional[str] = None,
                          country: str = "US",
                          include_similar: bool = False) -> ClassificationResult:
        """
        Classify a single business

        Args:
            business_description: Description of the business (required)
            business_name: Business name for additional context
            website_url: Website URL for enhanced classification
            country: ISO country code (default: US)
            include_similar: Include similar code suggestions

        Returns:
            ClassificationResult object with codes and confidence scores

        Example:
            >>> client = KYBClient(api_key="your_key")
            >>> result = client.classify.classify_business(
            ...     business_description="Online clothing retailer selling fashion apparel",
            ...     include_similar=True
            ... )
            >>> print(f"MCC Code: {result.mcc.code} - {result.mcc.description}")
```

```
        >>> print(f"Confidence: {result.mcc.confidence:.2%}")
    """
    payload = {
        'business_description': business_description,
        'country': country,
        'include_similar': include_similar
    }

    if business_name:
        payload['business_name'] = business_name
    if website_url:
        payload['website_url'] = website_url

    response = self.client._make_request('POST', '/classify', json=payload)
    return ClassificationResult.from_dict(response)

def classify_batch(self,
                   businesses: List[Dict],
                   webhook_url: Optional[str] = None) -> BatchJob:
    """
    Classify multiple businesses asynchronously

    Args:
        businesses: List of business data dictionaries
        webhook_url: Optional webhook URL for completion notification

    Returns:
        BatchJob object for tracking progress

    Example:
        >>> businesses = [
        ...     {"id": "biz1", "business_description": "Restaurant serving Italian food"},
        ...     {"id": "biz2", "business_description": "Software consulting services"}
```

```
        ... ]
        >>> job = client.classify.classify_batch(businesses)
        >>> while not job.is_complete():
        ...     time.sleep(30)
        ...     job.refresh()
        >>> results = job.get_results()
        """
        payload = {'businesses': businesses}
        if webhook_url:
            payload['webhook_url'] = webhook_url

        response = self.client._make_request('POST', '/classify/batch', json=payload)
        return BatchJob(self.client, response['job_id'])

class RiskAssessmentService:
    """Risk assessment operations"""

    def __init__(self, client: KYBClient):
        self.client = client

    def assess_risk(self,
                    business_id: str,
                    assessment_type: str = "initial",
                    include_predictions: bool = False,
                    risk_tolerance: str = "moderate") -> RiskAssessment:
        """
        Perform comprehensive risk assessment

        Args:
            business_id: Unique business identifier
            assessment_type: Type of assessment ('initial', 'periodic', 'triggered')
            include_predictions: Include 3/6/12 month predictions
            risk_tolerance: Risk tolerance level ('conservative', 'moderate', 'aggressive')
```

```
        Returns:
            RiskAssessment object with scores and analysis

        Example:
            >>> assessment = client.risk.assess_risk(
            ...     business_id="biz_123",
            ...     include_predictions=True
            ... )
            >>> print(f"Risk Score: {assessment.overall_score}/100")
            >>> print(f"Risk Level: {assessment.risk_level}")
            >>> for factor in assessment.risk_factors:
            ...     print(f"- {factor.name}: {factor.impact}")
        """
        payload = {
            'business_id': business_id,
            'assessment_type': assessment_type,
            'include_predictions': include_predictions,
            'risk_tolerance': risk_tolerance
        }

        response = self.client._make_request('POST', '/risk/assess', json=payload)
        return RiskAssessment.from_dict(response)

# Data models for SDK responses
@dataclass
class Classification:
    code: str
    description: str
    confidence: float

@dataclass
class ClassificationResult:
```

```
    classification_id: str
    mcc: Classification
    naics: Classification
    sic: Classification
    alternative_suggestions: List[Dict]
    processing_time_ms: int
    timestamp: datetime

    @classmethod
    def from_dict(cls, data: dict) -> 'ClassificationResult':
        """Create ClassificationResult from API response"""
        primary = data['primary_classifications']
        return cls(
            classification_id=data['classification_id'],
            mcc=Classification(**primary['mcc']),
            naics=Classification(**primary['naics']),
            sic=Classification(**primary['sic']),
            alternative_suggestions=data.get('alternative_suggestions', []),
            processing_time_ms=data['analysis_details']['processing_time_ms'],
            timestamp=datetime.fromisoformat(data['timestamp'].replace('Z', '+00:00'))
        )

class BatchJob:
    """Represents an asynchronous batch processing job"""

    def __init__(self, client: KYBClient, job_id: str):
        self.client = client
        self.job_id = job_id
        self._status = None
        self._results = None

    def refresh(self) -> None:
        """Refresh job status from server"""
```

```
        response = self.client._make_request('GET', f'/jobs/{self.job_id}')
        self._status = response

    def is_complete(self) -> bool:
        """Check if job has completed"""
        if not self._status:
            self.refresh()
        return self._status['status'] in ['completed', 'failed']

    def get_results(self) -> List[ClassificationResult]:
        """Get job results (only available when complete)"""
        if not self.is_complete():
            raise ValueError("Job not yet complete")

        if self._status['status'] == 'failed':
            raise KYBAPIException(f"Batch job failed: {self._status.get('error')}")

        if not self._results:
            response = self.client._make_request('GET', f'/jobs/{self.job_id}/results')
            self._results = [
                ClassificationResult.from_dict(result)
                for result in response['results']
            ]

        return self._results

    def wait_for_completion(self,
                            poll_interval: int = 30,
                            timeout: int = 1800) -> List[ClassificationResult]:
        """
        Wait for job completion and return results

        Args:
```

```
        poll_interval: Seconds between status checks
        timeout: Maximum wait time in seconds

    Returns:
        List of classification results when complete
    """
    start_time = time.time()

    while not self.is_complete():
        if time.time() - start_time > timeout:
            raise TimeoutError(f"Job did not complete within {timeout} seconds")

        time.sleep(poll_interval)
        self.refresh()

    return self.get_results()

# Async version of the client
class AsyncKYBClient:
    """Async version of KYB client for high-performance applications"""

    def __init__(self,
                 api_key: str,
                 base_url: str = "https://api.kybtool.com",
                 timeout: int = 30):
        self.api_key = api_key
        self.base_url = base_url.rstrip('/')
        self.timeout = aiohttp.ClientTimeout(total=timeout)

        self.classify = AsyncClassificationService(self)
        self.risk = AsyncRiskAssessmentService(self)

    async def __aenter__(self):
```

```python
        self.session = aiohttp.ClientSession(
            headers={
                'Authorization': f'Bearer {self.api_key}',
                'Content-Type': 'application/json'
            },
            timeout=self.timeout
        )
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        await self.session.close()

# Example usage patterns
def example_usage():
    """Example usage of the KYB Python SDK"""

    # Initialize client
    client = KYBClient(api_key="your_api_key_here")

    # Simple classification
    result = client.classify.classify_business(
        business_description="Online retailer selling electronics and gadgets",
        business_name="TechMart Electronics",
        website_url="https://techmart.example.com"
    )

    print(f"Classification completed in {result.processing_time_ms}ms")
    print(f"MCC: {result.mcc.code} - {result.mcc.description}")
    print(f"Confidence: {result.mcc.confidence:.1%}")

    # Risk assessment
    # First create or get business ID
    business = client.businesses.create({
```

```
    'legal_name': 'TechMart Electronics',
    'business_description': 'Online retailer selling electronics',
    'website_url': 'https://techmart.example.com'
})

# Assess risk
risk_assessment = client.risk.assess_risk(
    business_id=business.id,
    include_predictions=True
)

print(f"Risk Score: {risk_assessment.overall_score}/100")
print(f"Risk Level: {risk_assessment.risk_level}")

if risk_assessment.predictions:
    print("Risk Predictions:")
    for horizon, prediction in risk_assessment.predictions.items():
        print(f"  {horizon}: {prediction.predicted_score} ({prediction.trend})")

# Async example
async def async_example():
    """Example of async usage for high-performance applications"""

    async with AsyncKYBClient(api_key="your_api_key_here") as client:

        # Classify multiple businesses concurrently
        businesses = [
            "Restaurant serving Italian cuisine",
            "Software development consulting",
            "E-commerce clothing store",
            "Digital marketing agency"
        ]
```

```
    # Create classification tasks
    tasks = [
        client.classify.classify_business(desc)
        for desc in businesses
    ]

    # Execute all classifications concurrently
    results = await asyncio.gather(*tasks)

    for i, result in enumerate(results):
        print(f"{businesses[i]}: {result.mcc.code}")

if __name__ == "__main__":
    example_usage()
```

Story 7.2: JavaScript/Node.js SDK

As a Node.js developer
I want a comprehensive JavaScript SDK for both Node.js and browser environments
So that I can build web applications with KYB functionality

Acceptance Criteria:

```gherkin
```

Given I am developing a JavaScript/Node.js application
When I install the KYB JavaScript SDK
Then I should be able to use it in both Node.js and browser environments
And the SDK should support modern JavaScript features (async/await, Promises)
And the SDK should have TypeScript definitions included
And the SDK should handle CORS and authentication properly for browser usage
And comprehensive examples should be provided for common frameworks
And the SDK should follow JavaScript best practices and conventions

Scenario: React.js integration
Given I am building a React application
When I use the JavaScript SDK
Then I can easily integrate KYB services into React components
And the SDK should work well with React hooks and state management
And examples should show proper error handling and loading states
And the SDK should support client-side and server-side rendering

Scenario: Express.js backend integration
Given I am building a Node.js API with Express
When I integrate the KYB SDK
Then I can create middleware for automatic KYB processing
And the SDK should handle API rate limiting gracefully
And error handling should integrate with Express error middleware
And the SDK should support request logging and monitoring

This completes Part 3 of the Feature Specifications Document, covering Advanced Analytics/Reporting and the beginning of the SDK ecosystem.

Should I continue with **Part 4: Testing Specifications, Integration Requirements, and Implementation Guidelines**?