KYC Service Architecture

Table of Contents

- Table of Contents
- Overview
- Purpose
- Architecture Diagram
- Data Flow Diagram
- Module Structure
- Key Components
  - Application Configuration
  - Core Services
  - Entities
- Data Model
  - Key Queries
- Database
  - Database Data Model
  - Entity Relationship Diagram
  - Tables
- Messaging Architecture
  - Kafka Configuration
    - Kafka Topics and Message Payloads
      - Consumer Topics
      - Producer Topics
- Scheduled Tasks
- API
  - REST API
- Security
- Observability
- Configuration
  - Environment Variables
- Dependencies
  - Internal Modules
  - External Dependencies
- Key Design Patterns
- KYC Workflow
- Related Services

Status	REVIEW
Version	1.0.0
Author(s)	@Daniel Steinhauf
Date of creation	Jan 21, 2026
Last updated	Jan 21, 2026
Approved on	
<name of the person(s) in charge to approve the document >	Date of approval: yyyy-mm-dd

Overview

lt-kyc-service handles Know Your Customer (KYC) verification and customer onboarding processes. It manages KYC sessions, identity verification, and document processing.

Service Port: 8092
Management Port: 9001
Complexity: Medium

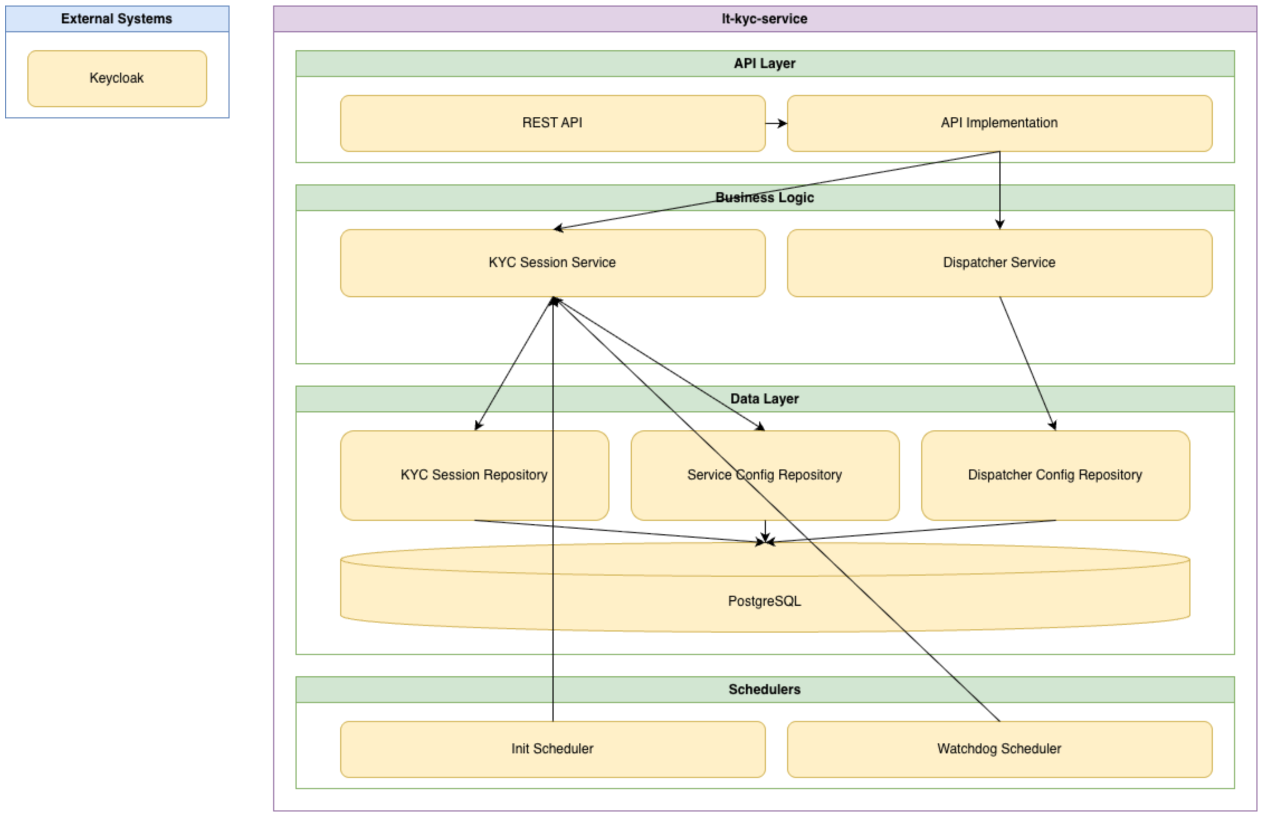
Purpose

The KYC service provides:

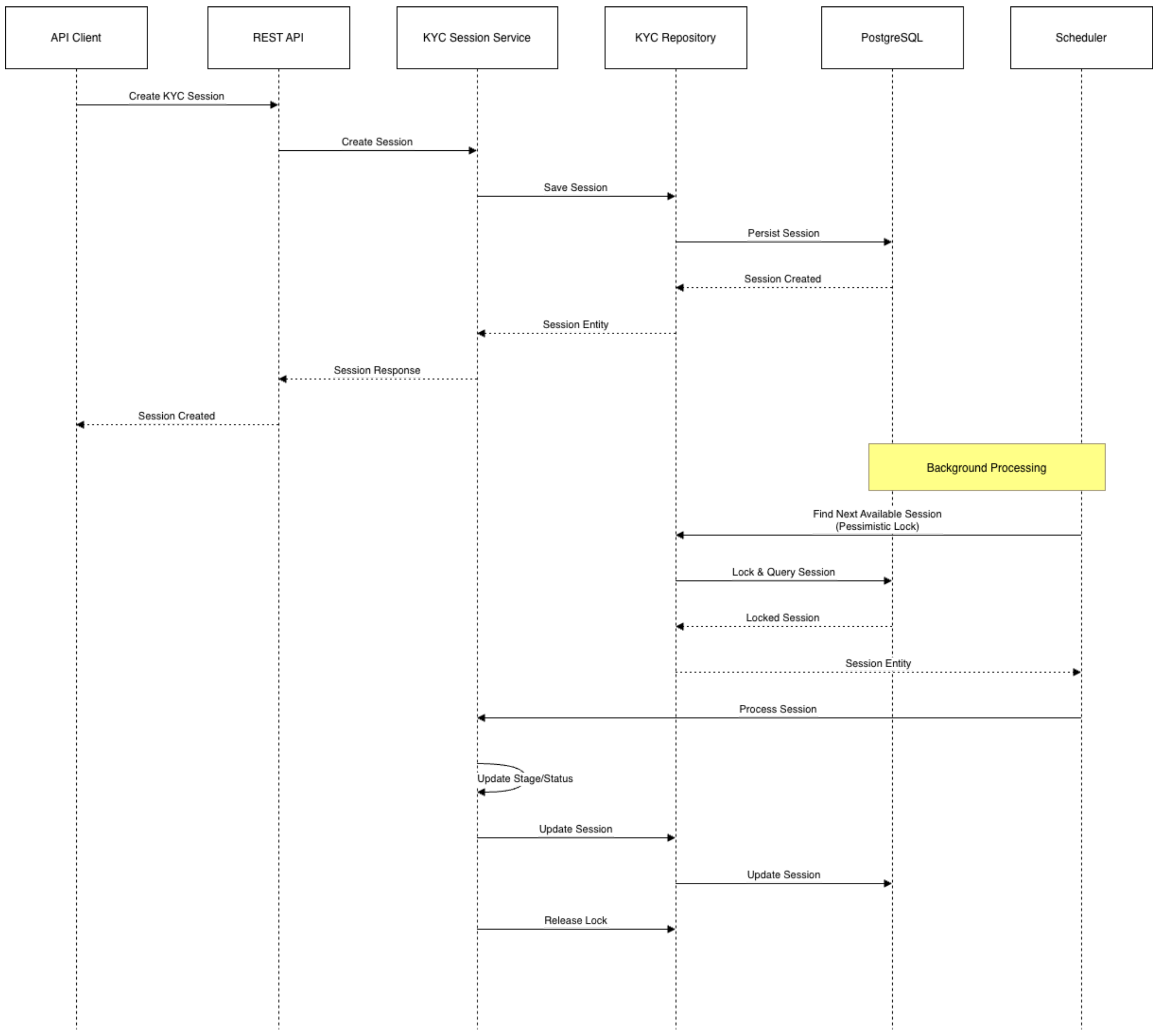
- KYC session management
- Identity verification workflows

- Document upload and processing
- Email KYC link generation for signers
- Contract document handling
- KYC stage and status management

Architecture Diagram



Data Flow Diagram



Module Structure

```
lt-kyc-service/
├── bootstrap/composite/      # Application entry point
│   ├── Application.java     # Main Spring Boot application
│   └── application.yml      # Service configuration
├── api/                     # API contracts
│   ├── impl/                # API implementation
│   └── sdk/                 # Client SDK
├── common/                  # Shared utilities
│   ├── auth/                # Authentication/authorization
│   └── [common utilities]
├── services/                # Business logic
│   ├── entity/             # Domain entities
│   ├── model/              # Domain models
│   ├── repository/         # Data repositories
│   ├── service/            # Business services
│   └── [other components]
├── messaging/              # Kafka messaging
│   └── [Kafka consumers/producers]
└── schedulers/             # Scheduled tasks
    └── [Scheduler components]
```

Key Components

Application Configuration

Main Class: `com.leaseteq.Application`

Enabled Features:

- `@EnableRetry` - Retry mechanism
- `@EnableCaching` - Caching support
- `@EnableScheduling` - Scheduled tasks
- `@EnableFeignClients` - Inter-service communication
- `@EnableJpaRepositories` - JPA repositories with Hypersistence Utils
- **Kafka** - Enabled via Spring Boot auto-configuration (Kafka dependencies and configuration present)

Core Services

1. **KycSessionService** (`services/service/`)
   - KYC session lifecycle management
   - Session status and stage tracking
   - Internal ID lookup
2. **KycSessionRepository** (`services/repository/`)
   - JPA repository for KYC sessions
   - Pessimistic locking for concurrent access
   - Query methods for finding available sessions
3. **ServiceConfigRepository** (`services/repository/`)
   - Service configuration management
   - Configuration key-value storage
4. **DispatcherConfigurationRepository** (`services/repository/`)
   - Dispatcher configuration for KYC providers
   - Provider selection rules

Entities

- **KycSession** (`services/entity/`)
  - KYC session entity
  - Status tracking (NEW, RETRY, etc.)
  - Stage tracking (INIT, KYC_IDENTITY_INITIATED, etc.)
  - Internal ID mapping
- **ServiceConfig** (`services/entity/`)
  - Service configuration storage
  - Key-value configuration pairs
- **DispatcherConfig** (`services/model/`)
  - Dispatcher configuration for provider selection

Data Model

Key Queries

Find Next Available Session for Processing:

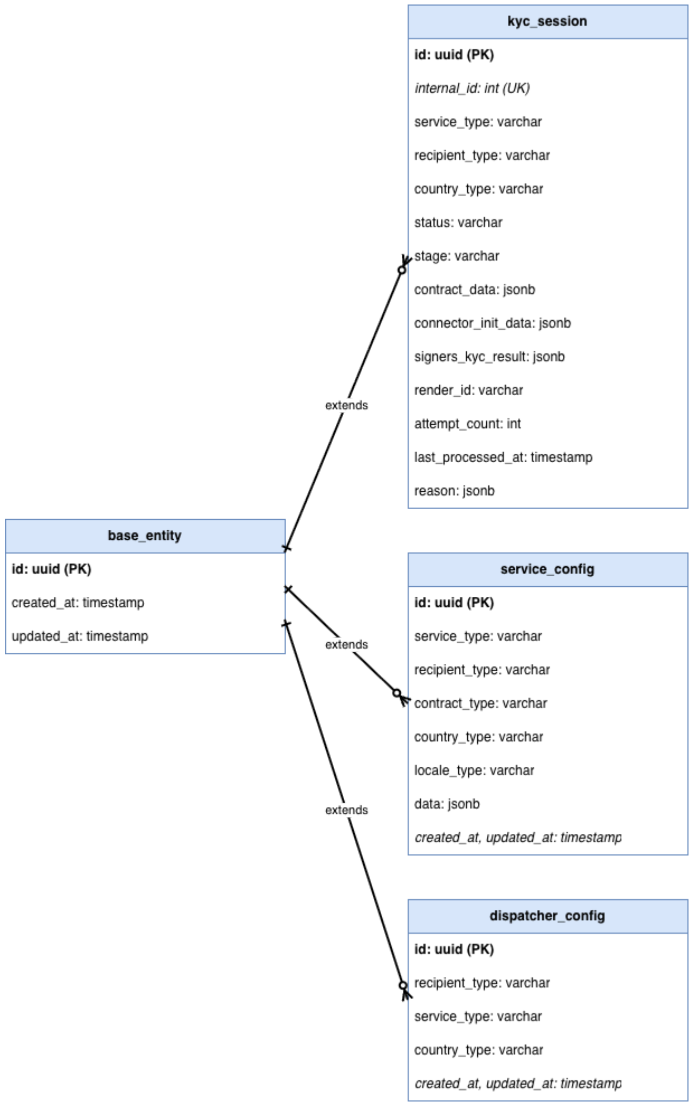
- Uses pessimistic write locking
- Finds sessions with status NEW and stage INIT
- Or status RETRY with specific stages
- Ordered by creation date (oldest first)
- Limits to 1 result

Database

- **Type**: PostgreSQL
- **Schema**: `lt_kyc_service`
- **Connection Pool**: HikariCP
  - Maximum pool size: 20
  - Minimum idle: 3
- **Migrations**: Liquibase
  - Changelog: `services/src/main/resources/db/changelog/changelog-master.yaml`
- **Audit**: Hibernate Envers enabled

Database Data Model

Entity Relationship Diagram



Tables

kyc_session

Purpose: Central table tracking the complete lifecycle of each KYC (Know Your Customer) verification session. Stores session state, status, stage progression, contract data, KYC provider responses, and failure reasons. Supports retry logic and concurrent processing with pessimistic locking.

Columns:

- `id` (uuid, PK) - Primary key, auto-generated UUID
- `created_at` (timestamp, not null) - Session creation timestamp
- `updated_at` (timestamp, not null) - Last update timestamp
- `internal_id` (int, not null, unique) - Business identifier linking to external systems (unique constraint)
- `service_type` (varchar, not null) - KYC service type: `ID_NOW_SERVICE`, etc.
- `recipient_type` (varchar, not null) - Recipient type: `CUSTOMER_PRIVATE`, `CUSTOMER_BUSINESS`, `ENTREPRENEUR`
- `country_type` (varchar, not null) - Country code: `AT`, `DE`, `CH`

- `status` (varchar, not null) - Session status: `NEW`, `RETRY`, `COMPLETED`, `FAILED`, etc.
- `stage` (varchar, not null) - Current stage: `INIT`, `KYC_IDENTITY_INITIATED`, `EMAIL_KYC_LINK_FOR_SIGNERS_SENT`, `CONTRACT_DOCUMENT_UPLOADED`, etc.
- `contract_data` (jsonb, not null) - Contract context data (structured JSON via `JpaConverterContractData`)
- `connector_init_data` (jsonb, nullable) - Initialization data for external KYC provider
- `signers_kyc_result` (jsonb, nullable) - Per-signer KYC verification results (structured JSON via `JpaConverterSignersKycResult`)
- `render_id` (varchar, nullable) - External render/flow identifier from KYC provider
- `attempt_count` (int, not null, default 0) - Number of processing attempts (for retry logic)
- `last_processed_at` (timestamp, nullable) - Timestamp of last processing attempt
- `reason` (jsonb, nullable) - Failure reason details (structured JSON via `JpaConverterIdentityFailedReason`)

Constraints:

- Primary key: `kyc_session_pk` on `id`
- Unique constraint: `kyc_session_internal_id` on `internal_id`
- All columns NOT NULL except nullable JSONB fields and timestamps

Indexes:

- `kyc_session_pk` (unique) on `id` - Primary key index
- `kyc_session_internal_id` (unique) on `internal_id` - Unique business identifier index

Relationships: None (standalone session table)

Usage:

- Backed by `KycSession` entity (extends `BaseEntity`)
- Queried by `KycSessionRepository` with pessimistic locking (`@Lock(LockModeType.PESSIMISTIC_WRITE)`)
- Used by schedulers to find next available session for processing
- Supports state machine pattern for KYC workflow progression
- Retry logic: Failed sessions move to `RETRY` status and can be reprocessed

Key Queries:

- `findByInternalId()` - Lookup by business ID
- `findNextAvailableSessionForInitIdentProcessing()` - Finds sessions ready for processing with pessimistic lock

service_config

Purpose: Stores KYC-related service configuration (templates, provider settings, email templates) per service type, recipient type, contract type, country, and locale. Used for template selection, email configuration, and provider-specific settings.

Columns:

- `id` (uuid, PK) - Primary key, auto-generated UUID
- `created_at` (timestamp, not null, default `now()`) - Record creation timestamp
- `updated_at` (timestamp, not null, default `now()`) - Last update timestamp
- `service_type` (varchar(50), not null) - Service type: `CARBONE` , `POSTMARK` , etc.
- `recipient_type` (varchar(50), nullable) - Recipient type: `CUSTOMER_PRIVATE` , `CUSTOMER_BUSINESS` , `ENTREPRENEUR`
- `contract_type` (varchar(50), nullable) - Contract type: `LOAN` , `LEASE` , `MOBILITY_PLAN`
- `country_type` (varchar(2), not null) - Country code: `AT` , `DE` , `CH`
- `locale_type` (varchar(10), nullable) - Locale identifier
- `data` (jsonb, not null) - Configuration payload as JSON containing:
    - For CARBONE: `local_code` , `templateId`
    - For POSTMARK: `from` , `subject` , `replyTo` , `fromPerson` , `templateName` , `cc`

Constraints:

- Primary key: `service_config_pk` on `id`
- `service_type` and `country_type` are NOT NULL
- Other columns nullable for flexible matching

Indexes: None explicitly defined

Relationships: None (standalone configuration table)

Usage:

- Backed by `ServiceConfig` entity (extends `BaseEntity`)
- Queried by `ServiceConfigRepository` to retrieve configuration for specific scenarios
- Used for template selection (Carbone PDF templates) and email configuration (Postmark templates)
- Pre-populated with configuration for different service/recipient/contract/country combinations

Example Data:

- CARBONE/CUSTOMER_BUSINESS/LOAN/AT → Template ID for Austrian B2B loan contracts
- POSTMARK/CUSTOMER_PRIVATE/LEASE/AT → Email template for Austrian B2C lease KYC links

dispatcher_config

Purpose: Maps combinations of recipient type, KYC service type, and country to a specific KYC provider implementation. Used by dispatcher service to route KYC requests to the correct provider (e.g., ID_NOW for video identification).

Columns:

- `id` (uuid, PK) - Primary key, auto-generated UUID
- `created_at` (timestamp, not null) - Record creation timestamp
- `updated_at` (timestamp, not null) - Last update timestamp
- `recipient_type` (varchar, not null) - Recipient type: `CUSTOMER_PRIVATE`, `CUSTOMER_BUSINESS`, `ENTREPRENEUR`
- `service_type` (varchar, not null) - KYC service type: `ID_NOW_SERVICE`, etc.
- `country_type` (varchar, not null) - Country code: `AT`, `DE`, `CH`

Constraints:

- Primary key: `dispatcher_config_pk` on `id`
- All columns NOT NULL

Indexes: None explicitly defined

Relationships: None (standalone configuration table)

Usage:

- Backed by `DispatcherConfig` entity (extends `BaseEntity`)
- Queried by `DispatcherConfigurationRepository` to find provider for given recipient/country combination
- Used by dispatcher logic to select KYC provider (e.g., ID_NOW for video identification)
- Pre-populated with provider mappings for different recipient types and countries

Example Data:

- CUSTOMER_BUSINESS/ID_NOW_SERVICE/AT → ID_NOW provider for Austrian B2B customers
- CUSTOMER_PRIVATE/ID_NOW_SERVICE/AT → ID_NOW provider for Austrian B2C customers
- CUSTOMER_BUSINESS/ID_NOW_SERVICE/DE → ID_NOW provider for German B2B customers

Messaging Architecture

Kafka Configuration

The service uses Apache Kafka for asynchronous messaging, specifically for consuming identity completion notifications from KYC connectors.

Configuration:

- **Bootstrap Servers**: `${BOOTSTRAP_SERVERS:localhost:29092}`
- **Consumer Group**: `kyc-group`
- **Security**: SASL_SSL with AWS MSK IAM authentication
- **Acknowledgment Mode**: Manual immediate
- **Auto-commit**: Disabled

Topics:

1. **kyc-identity-complete-topic** (`${KYC_IDENTITY_COMPLETE_TOPIC:kyc-identity-completed-notification-topic}`)
   - **Purpose**: Receives identity completed notifications from KYC connectors (e.g., IDNow)
   - **Consumer**: `IdentityCompletedNotificationConsumer`
   - **Message Format**: JSON (Map<String, Object>)
   - **Processing**: Maps to `IdentityCompletedDto` and processes via `ConsumerProcessor`
2. **kyc-identity-completed-dlq-topic** (`${KYC_IDENTITY_COMPLETE_DLQ_TOPIC:kyc-identity-completed-dlq-topic}`)
   - **Purpose**: Dead Letter Queue for failed identity completion notifications
   - **Error Handling**: Messages that fail after retries are sent to DLQ
   - **Retry Configuration**:
     - Fixed back-off interval: `${KAFKA_FIXED_BACK_OFF_INTERVAL:10000}` ms
     - Max attempts: `${KAFKA_FIXED_BACK_OFF_MAX_ATTEMPTS:3}`

Components:

1. **IdentityCompletedNotificationConsumer** (`messaging/`)
   - Kafka listener for identity completed notifications
   - Processes notifications and updates KYC session status
   - Uses manual acknowledgment
2. **IdentityNotificationListenerConfig** (`messaging/`)
   - Configures Kafka listener container factory
   - Sets up error handler with DLQ support
   - Configures acknowledgment mode
3. **IdentityCompletedConsumerErrorHandler** (`messaging/`)
   - Error handler for failed message processing
   - Implements retry logic with fixed back-off
   - Sends failed messages to DLQ after max retries
4. **KafkaClient** (`common/kafka/`)
   - Interface for Kafka message production
   - Used for sending messages to DLQ
5. **KafkaClientImpl** (`common/kafka/`)
   - Implementation of KafkaClient
   - Uses `KafkaTemplate` for message production

Message Flow:

1. KYC connector (e.g., IDNow) sends identity completed notification to `kyc-identity-complete-topic`
2. `IdentityCompletedNotificationConsumer` receives and processes the message
3. Message is mapped to `IdentityCompletedDto` and processed via `ConsumerProcessor`
4. On success: Message is acknowledged

5. On failure: Error handler retries up to max attempts, then sends to DLQ

Error Handling:

- Failed messages are retried with fixed back-off interval
- After max retries, messages are sent to DLQ topic
- Manual acknowledgment ensures at-least-once delivery semantics

Kafka Topics and Message Payloads

Consumer Topics

Topic: `kyc-identity-complete-topic` (configurable via `${services-config.kafka.topics.kyc-identity-complete-topic}`)

Consumer: `IdentityCompletedNotificationConsumer`

Message Format: `Map<String, Object>` (JSON)

Payload Structure:

```
{
  "transactionNumber": "string",           // Transaction number from KYC provider
  "internalId": "integer",                 // Internal business identifier
  "identityCompleted": "boolean",          // Whether identity verification completed
  "identificationResult": "string",        // Enum: SUCCESS, SUCCESS_DATA_CHANGED, CANCELLED, ABORTED, CHECK_PENDING, REVIEW_RESULT, FRAUD_SUSPICION_CONFIRMED, FRAUD_SUSPICION_PENDING
  "reason": "string",                      // Optional reason for failure
  "identStatus": "string",                 // Enum: CREATED, BOOKED, SIGNED, FAILED
  "exception": "string",                   // Optional exception message
  "signer": {                              // Signer information
    "signerType": "string",                // Enum: PRIMARY_SIGNER, SECONDARY_SIGNER, etc.
    "isSigned": "boolean",                 // Whether signer has signed
    "nextSigner": "string"                 // Optional next signer type
  },
  "connectorResult": {                     // Connector-specific result data
    // Dynamic JSON structure from KYC provider
  }
}
```

Processing:

- Message is mapped to `IdentityCompletedDto` via `IdentityCompletedMapper`
- Processed by `ConsumerProcessor<IdentityCompletedDto>`
- Updates KYC session status and stage based on identification result

Dead Letter Queue: `kyc-identity-complete-dlq-topic` (configurable via `${services-config.kafka.topics.kyc-identity-completed-dlq-topic}`)

- Messages that fail after max retry attempts are sent to DLQ
- Retry configuration: Fixed back-off interval (default: 10000ms), max attempts (default: 3)

Producer Topics

Note: This service currently does not produce Kafka messages. It only consumes identity completion notifications.

Scheduled Tasks

Schedulers are configured for:

- KYC session initialization
- Watchdog for stuck sessions
- Session state monitoring

API

REST API

- KYC session management endpoints
- Identity verification endpoints
- Document upload endpoints
- Email KYC link generation endpoints
- Contract document endpoints

API Documentation: SpringDoc OpenAPI (Swagger UI)

Security

- **Authentication**: OAuth2 Resource Server (Keycloak)
- **Authorization**: Role-based access control
- **JWT Validation**: Dynamic JWK set retrieval

Observability

- **Health Endpoint**: `/actuator/health`
- **Metrics**: `/actuator/prometheus`
- **Tracing**: Distributed tracing enabled

Configuration

Environment Variables

- `APPLICATION_PORT` - Service port (default: 8092)
- `MGT_SERVER_PORT` - Management port (default: 9001)
- `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD` - Database
- `KEYCLOAK_DOMAIN`, `KEYCLOAK_REALM` - Keycloak
- Scheduler configuration variables

Dependencies

Internal Modules

- `api/impl` - API implementation
- `services` - Business logic
- `common` - Shared utilities
- `messaging` - Messaging components (if applicable)
- `schedulers` - Scheduled tasks

External Dependencies

- Spring Boot 3.4.4
- Spring Cloud OpenFeign
- Spring Data JPA
- Hypersistence Utils
- Liquibase
- PostgreSQL Driver
- Micrometer Prometheus
- SpringDoc OpenAPI

Key Design Patterns

1. **Session Management**: State machine pattern for KYC session lifecycle
2. **Pessimistic Locking**: Prevents concurrent processing of same session
3. **Repository Pattern**: Data access abstraction
4. **Scheduler Pattern**: Background processing for session management

KYC Workflow

1. **Session Creation**: New KYC session created with status NEW, stage INIT
2. **Identity Initiation**: Session moves to KYC_IDENTITY_INITIATED stage
3. **Email KYC Link**: Email sent to signers (EMAIL_KYC_LINK_FOR_SIGNERS_SENT)
4. **Document Upload**: Contract documents uploaded (CONTRACT_DOCUMENT_UPLOADED)
5. **Processing**: Schedulers pick up sessions for processing
6. **Retry Logic**: Failed sessions move to RETRY status

Related Services

- **lt-aml-service**: AML checks (may be called during KYC)
- **lt-credit-service**: Credit checks (may be called during KYC)
- **lt-company-registry-check-service**: Company verification