Functional Analysis: User Migration Platform

The migration system is designed to securely and controllably manage the transfer of user data from a Legacy application (OLD) to a new platform (NEW). This analysis describes the functionality exposed to the two main actors: the **Legacy User** and the **Administrator**.

1. Functionality Exposed to the Legacy User (OLD)

Users who still access the Legacy system interact with the migration platform through the protected /api/v1/UserOld endpoint.

Functional Requirement	Endpoint / Method	Functional Description	Gating Details
Migration Status Display	GET /	The user retrieves their personal data (e.g., FirstName, LastName) and verifies: a) If they are already migrated (IsMigrated). b) If migration is currently available (CanBeMigrated).	Migration is offered only if: 1) The user exists in the OLD DB. 2) There is no migration log entry. 3) There is at least one free slot (SlotManager).
Migration Acceptance	POST /migration/accept	The user voluntarily initiates the process of migrating their data to the NEW platform.	**Throttle Control:** The request is accepted (202 Accepted) only if the SlotManager reports available slots. Otherwise, a 429 Too Many Requests status is returned.
Process Initialization	N/A (Core)	Upon acceptance, the system creates	The operation is asynchronous; the

		a PENDING entry in the audit log and sends an asynchronous message to the Broker (MessageProducer) for Worker processing.	user does not wait for completion.

2. Administrative and Monitoring Functionality

Administrators interact with the platform through the Admin role protected endpoint (/api/v1/Admin).

2.1 Concurrency Monitoring and Control

Administrators can view the overall health and capacity of the migration system.

Functional Requirement	Endpoint / Method	Functional Description	Core Components Involved
Concurrency Slot Status	GET /slot/status	Displays the maximum number of slots (MaxSlots), available slots, and slots currently in use (SlotsInUse).	SlotManager (SemaphoreSlim)
Global Migration Status	GET /migration/status	Provides an overview of the overall migration progress, including: - Percentage of users migrated. - Total successful and failed migrations. - Migrations currently in progress.	MigrationManager (aggregates data from MigrationStatusRepository)

2.2 Forced Migration Management

This feature allows the administrator to override the user's consent for operational reasons (e.g., mandated migration).

Functional Requirement	Endpoint / Method	Functional Description	Audit Details
Forced Migration	POST /migration/{legacyUserId}	Initiates the migration process for a specific user, regardless of their interaction.	The audit entry (MigrationStatus) is marked with the AdminActionBy field, containing the identifier of the administrator who executed the action.

3. Worker and Asynchronous Logic

The Migration.Worker is the engine of the system and operates in the background by consuming messages from the Broker.

Process Phase	Logical Component	Detailed Description
1. Slot Acquisition & Pre-Check	MigrationProcessor	Before spending resources, the Worker verifies the existence and validity of Legacy data. If the data is dirty (DataTransformer throws an exception), the process stops immediately **without acquiring a slot**.
2. Transformation and Validation	DataTransformer	Performs data normalization (e.g., email) and field mapping (e.g., DocumentType from OLD to NEW).

3. SAGA and Write (Critical Phase)	MigrationProcessor	Executes the creation of the NEW user and the update of the Audit Log.
4. Compensation (Rollback)	MigrationProcessor (catch block)	In case of Phase 3 failure, system consistency is ensured: if the NEW user was created, they are immediately deleted (UserNewRepository.DeleteNewUser), and the final state in the log is **FAILED**.
5. Finalization	UserOldRepository	After the SAGA success (writing NEW and marking the Log), the user in the Legacy database is marked as migrated (IsMigrated = true).

4. Security and Traceability Requirements

Requirement	Implementation	Notes
API Authentication	JWT Bearer Authentication	All APIs are protected. Program.cs configures JwtBearerDefaults to validate incoming tokens.
Authorization	[Authorize] and [Authorize(Roles = "Admin")]	The .NET Core framework ensures that only users with the Admin Claim can access the AdminController.
Audit Logging	MigrationStatus Table	Every migration attempt (standard or forced) creates an entry in the Audit DB. Administrative operations are tracked via the AdminActionBy field.