**Backend Design: Unified Microservice Authorization Architecture**

**Purpose**

This design introduces a standardized and maintainable approach to handling authorization for outbound microservice calls in the TravelMate system. Previously, different services duplicated logic to obtain authorization tokens for either OAuth2-based APIs or SecureM2M-protected endpoints. The lack of consistency led to maintenance issues, redundancy, and risk of misconfiguration.

The need for this design became especially clear when the Work Force Identity Database (WFIDB) introduced an API that required a different authorization mechanism (OAuth2 with JWT) in contrast to the SecureM2M approach used by other internal APIs such as the FeatureFlag API. This divergence introduced complexity and made it harder to maintain consistent practices across services.

The new approach consolidates this logic into a shared and extensible authorization layer that supports both schemes and includes built-in token caching to minimize unnecessary external calls.

**Supported Authorization Strategies**

* **OAuth2 JWT**: For APIs protected by the Swisscom Authorization API, such as WFIDB.
* **Secure Machine-to-Machine (SecureM2M)**: For internal APIs requiring SecureM2M credentials.

Both strategies are implemented under a shared abstraction and can be used transparently by business services.

**Architecture Overview**

The authorization flow is structured around the following components:

**AbstractAuthorizationService**

* Defines shared behavior for all authorization mechanisms.
* Provides:
  + getToken(AuthTokenConfig): Main entry point for token acquisition.
  + fetchNewToken(AuthTokenConfig): Internally used to retrieve new tokens when needed.
  + shouldExtractClaims(): Optional hook for claim-based logic.
* Includes built-in caching of tokens per configuration (e.g., per client ID and audience).

**OAuth2JwtService / SecureM2MJwtService**

* Implement the logic for their respective token schemes.
* Internally delegate the actual token retrieval to:
  + OAuth2JwtClient (for OAuth2)
  + SecureM2MJwtClient (for SecureM2M)
* Automatically check the token cache before making external calls.

**OAuth2JwtClient / SecureM2MJwtClient**

* Handle the actual HTTP requests to the authorization endpoints.
* Encapsulate the protocol-specific logic for interacting with external systems.

**WFIService and FeatureFlagService**

* Business-level services that require secure access to external APIs.
* Use the appropriate authorization service to obtain tokens.
* Do not manage token state, cache, or protocol logic themselves.

**WFIClient**

* A generic client for dispatching authenticated HTTP calls to the Work Force Identity API or FeatureFlag API.
* Receives tokens from the services and attaches them to outbound requests.

**Token Caching Behaviour**

* Each token is cached based on its configuration.
* Expiration is checked before reuse; new tokens are fetched only when needed.
* This significantly reduces:
  + Repeated network calls to upstream token providers
  + Latency in microservice-to-microservice communication
  + Load on external authentication systems

**Benefits**

* **Centralized Authorization Logic**: Eliminates duplicated and scattered token management code.
* **Reusable and Extensible**: New token strategies can be introduced without modifying service logic.
* **Performance-Optimized**: Built-in caching reduces redundant token requests.
* **Cleaner Integration**: Services focus on their responsibilities without concern for authentication protocol details.
* **Testable and Isolated**: Each component, including token expiration handling, can be tested independently.
* **Consistency Across Projects**: Promotes unified practices for secure service communication throughout the organization.