GSAM MDN Caching API

Here is the Implementation of GSAM MDN Caching API in Simple-Cache-Postgres Module.

1. **Data Layer Flow:**
   - `GSAM_MDNs.java`
     - JPA entity class representing a MDN Table
     - Uses `@Entity` annotation to mark it as a database entity
     - Maps to a database table named "rtt_gsam_mdns"
     - Contains single field 'mdn' marked as `@Id` (primary key)
     - Basic getter/setter methods for the mdn field
2. **Repository Layer Flow:**
   - `RTT_GSAM_MDNsRepository.java`
     - Extends JpaRepository for basic CRUD operations
     - Contains custom query method `findAllMDNs()`
     - Uses native SQL query to select all MDNs from rtt_gsam_mdns
     - Returns List<String> containing all MDNs
3. **Cache Implementation (Scala):**
   - `GSAMReference.scala`
     - Implements thread-safe caching using AtomicReference
     - Three main operations:
       - `DummyConstruct()`: Initializes empty cache
       - `Construct(data)`: Populates cache with provided MDN list
       - `get()`: Retrieves cached MDN list
     - Uses logging to track cache building time and operations

4. **Service Layer Flow:**
    - `ReferenceCacheService.java`
        - Provides method `fetchAllGSAM_MDNs()`
        - Simply retrieves data from GSAMReference cache
        - Acts as intermediary between controller and cache
    - `ReferenceCacheRefreshService.java`
        - Handles cache initialization and refresh operations
        - Components:
            - `Init()`: Called at startup, initializes empty cache
            - `refreshGSAMMDNs()`: Updates cache with fresh data from database
            - Error handling with logging
            - Returns boolean indicating success/failure
5. **Controller Layer Flow:**
    - `ReferenceEndPoint.java`
    - Has two endpoints:

a. `/getGSAM_MDNs` (POST):

- Flow:
    1. Logs request receipt
    2. Creates APIResponse object
    3. Calls service to fetch MDNs from cache
    4. Sets response code (200 for success)
    5. Sets data in response
    6. Logs completion with result count
    7. Returns response
- Error handling:
    - Catches exceptions

- Sets response code 300
- Logs error
- Returns null data

b. `/refreshGSAMMDNs` (POST):

- Flow:
  1. Logs refresh request
  2. Creates APIResponse object
  3. Calls cache refresh service
  4. Sets appropriate response based on refresh result
  5. Logs operation result
- Success case:
  - Response code 200
  - Success message
- Failure cases:
  - Response code 300
  - Error message with details

6. **Complete Data Flow:**

```
Database -> Repository -> Cache -> Service -> Controller -
> Client
```

- Database stores MDN data
- Repository fetches from database
- Cache maintains in-memory copy
- Service manages cache access
- Controller handles HTTP requests/responses

7. **Error Handling Flow:**

- Try-catch blocks at multiple levels
- Comprehensive error logging
- Appropriate error responses
- Stack trace preservation

8. **Performance Considerations:**
    - Uses caching to reduce database load
    - AtomicReference for thread safety
    - Logging of performance metrics
    - Batch operations for efficiency
9. **Operational Flow:**
    - Application startup:
        1. Initialize empty cache
    - Normal operation:
        1. Serve requests from cache
    - Refresh operation:
        1. Fetch fresh data from database
        2. Update cache
        3. Continue serving from updated cache
10. **Security and Validation:**
    - JSON content type enforcement
    - POST method restrictions
    - Exception handling for invalid requests
    - Null checks in cache operations

This implementation follows a clean architecture pattern with clear separation of concerns between layers, making it maintainable and scalable. The caching mechanism helps optimize performance by reducing database load.