OPERATOR MAPPINGS TECHNICAL SPECIFICATION

1 OVERVIEW
2 PROBLEM
3 SCOPE
    3.1 InScope
    3.2 OutofScope
4 USE CASES
    4.1 Use Case 1: Read Operator Mapping Entry (OM-1.1)
    4.2 Use Case 2: Create Operator Mapping Entry (OM-1.2)
    4.3 Use Case 3: Update Operator Mapping Entry (OM-1.3)
    4.4 Use Case 4: Delete Operator Mapping Entry (OM-1.4)
    4.5 Use Case 5: Retrieve All Network Names (OM-2)
    4.6 Use Case 6: Mapping Countries to Their Operators (OM-3)
    4.7 Use Case 7: Grouping Operators by Network Size (OM-4)
    4.8 Use Case 8: Identify Potential Roaming Partners by TADIG Code (OM-5)
5 SOLUTION DESCRIPTION
    5.1 Summary
    5.2 System Architecture
    5.3 Database Design
    5.4 Read Operator Sequence Diagram
    5.5 Create Operator Sequence Diagram
    5.6 Update Operator Sequence Diagram
    5.7 Delete Operator Sequence Diagram
6
    6.1 Retrieve Network Names Sequence Diagram
    6.2 Mapping Countries to Operators Sequence Diagram
    6.3 Identify Potential Partners Sequence Diagram
7 API COMPONENT OVERVIEW
8 TECHNICAL RISKS
9 IMPORTANT TECHNICAL DECISIONS
10 FUNCTIONAL REQUIREMENTS
11 NON FUNCTIONAL REQUIREMENTS
12 FUTURE ADDITIONS

OVERVIEW

The aim of this project is to create a competent Operator Mappings API system for managing telecom data. Using the API, you can create, read, update, and delete operator information, pull up network names, automatically classify operators by their e212s into groups of small, medium, or large and instantly pinpoint the best roaming partners across neighboring countries using TADIG bordering data.

PROBLEM

- **Data Inconsistency:** Users handedit a shared JSON file, which leads to typos (e.g. “Vodafonne” vs. “Vodafone”) and inconsistent casing or code formats (“usa” vs. “USA”).
- Why it happens: manual, freeform edits lacks validation which may lead to inconsistencies.

- **Limited Query Flexibility:** Consumers must download and parse the entire JSON file to filter or group operator data by country, network size, IMSI, or TADIG, since the file offers no parametrized endpoints or builtin filtering capabilities.

- **Inefficient RoamingPartner Discovery:** Identifying roaming partners requires manual crossreferencing of TADIG codes against a separate borders dataset, which often results in stale or incomplete partner lists. Why it happens: there is no unified lookup service that maps “TADIG country neighboring countries operators.”

The aim of this project to create a restful API solution with automatic validation and normalization that provides dedicated typed API endpoints for filtering, grouping and automates the full TADIGcountryborder lookup to keep roamingpartner data accurate and uptodate.

SCOPE

InScope

API Endpoints

- CRUD Operations: Reliable Create, Read, Update, Delete actions with validation and logging.
- Retrieving all network names
- Mapping countries to operators
- Group operators by network size (E.212 codes)
- Identify roaming partners via TADIG code

OutofScope

- **External Data Integration**
  Integration with external thirdparty systems beyond initial data import.
- **User Management**
  Comprehensive user management & interface.

USE CASES

Use Case 1: Read Operator Mapping Entry (OM-1.1)

Use Case	Read Operator by E164/E212 or TADIG
Description	This use case enables the retrieval of operator mapping records from the database. It provides the foundation for viewing detailed operator information, including names, TADIG codes, e212 codes, and associated countries, while ensuring data consistency and accurate validation.
User Story	As a customer, I want to view the details of an existing operator entry so that I can verify the accuracy of operator data.

Use Case 2: Create Operator Mapping Entry (OM-1.2)

Use Case	Create new operator mapping entry
Description	**This use case focuses on the creation of a new operator entry within the mapping file. It allows adding essential detail s such as all essential fields and country information.** **The system validates inputs and prevents duplicate entries.**
User Story	As a customer, I want to add a new operator entry so that the mapping file accurately reflects the current operator roster.

Use Case 3: Update Operator Mapping Entry (OM-1.3)

Use Case	Updating existing operator data with validation.
Description	Focused on modifying an existing operator entry, this use case allows updating information such as operator names, TADIG codes, e212 associations, and country details.

User Story	As a customer, I want to update an operator’s information so that any changes are accurately reflected in the mapping file.

Use Case 4: Delete Operator Mapping Entry (OM-1.4)

Use Case	Removing an operator entry
Description	This use case enables the deletion of an operator mapping record from the system by their TADIG. It ensures that obsolete or inactive operator data can be removed safely, with confirmation of the deletion action and proper logging for traceability.
User Story	As a customer, I want to remove an operator’s record when it is no longer valid so that the mapping file contains only relevant and up-to-date data.

Use Case 5: Retrieve All Network Names (OM-2)

Use Case	List all network names
Description	This functionality provides an endpoint that queries and returns a complete list of network names available within the system. It allows users to quickly review and select from the available networks without navigating through the entire operator dataset.
User Story	As a customer, I want to view all network names so that I can easily identify the available networks within the system.

Use Case 6: Mapping Countries to Their Operators (OM-3)

Use Case	Mapping countries to their regional operators.
Description	In this use case, an endpoint is created to provide a mapping between countries and the operators that operate within them. This mapping offers clarity on regional operator distribution, aiding in market analysis and localized decision-making.
User Story	As a customer, I want to see a mapping of countries to their corresponding operators so that I can view information on the regional operators.

Use Case 7: Grouping Operators by Network Size (OM-4)

Use Case	Categorize operators into groups of small, medium and large by the size of their e212 codes.
Description	This use case addresses the need to group operators based on their network size, which is determined by the number of e212 codes they manage. The system will categorize operators (e.g., small, medium, large) to facilitate comparative analysis and strategic evaluations.
User Story	As a customer, I want to view operators grouped by network size so that I can compare and contrast.

Use Case 8: Identify Potential Roaming Partners by TADIG Code (OM-5)

Use Case	Find roaming partners from bordering countries via TADIG.
Description	Given a TADIG code, this use case identifies potential roaming partners by determining the operator's country and finding operators in bordering countries. It integrates geographical border data with the operator mapping to suggest viable roaming agreements.
User Story	As a customer, I want to input a TADIG code and see a list of operators in bordering countries so that I can easily discover and consider potential roaming partners.

SOLUTION DESCRIPTION

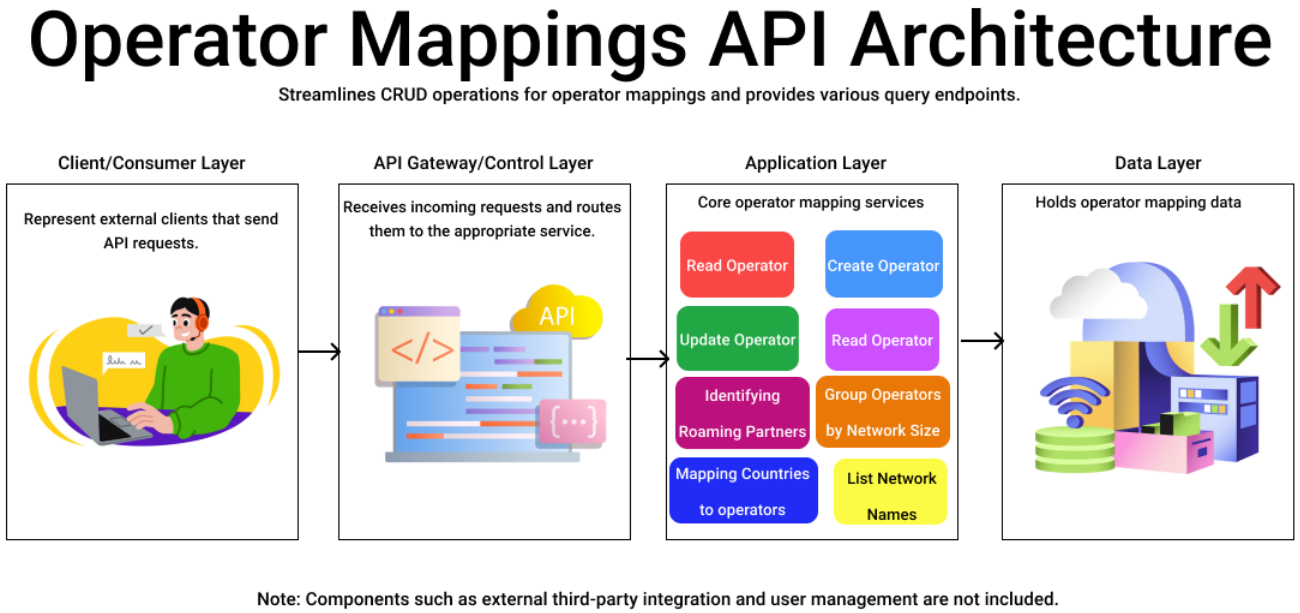
Summary

The operator mappings service is a RESTful API for managing telecom data. The Data will be loaded into memory at startup, leveraging **Tokio** for Rust to build reliable, asynchronous applications.

API Capabilities

- **CRUD Operations:**
  - *Create:* Adds new operator entries with validation, file locking, and audit logging.
  - *Read:* Retrieves existing operator details with error feedback.
  - *Update:* Modifies operator data while handling exceptions and ensuring data consistency.
  - *Delete:* Removes operator entries safely with appropriate error handling.
- **Additional Endpoints:**
  - **List Network Names:** Returns all available network names.
  - **Map Countries to Operators:** Provides a mapping of countries to their respective operators.
  - **Group by Network Size:** Categorizes operators as small, medium, or large based on the number of managed e212 codes.
  - **Identify Roaming Partners:** Discovers potential roaming partner operators using TADIG codes and bordering countries data.

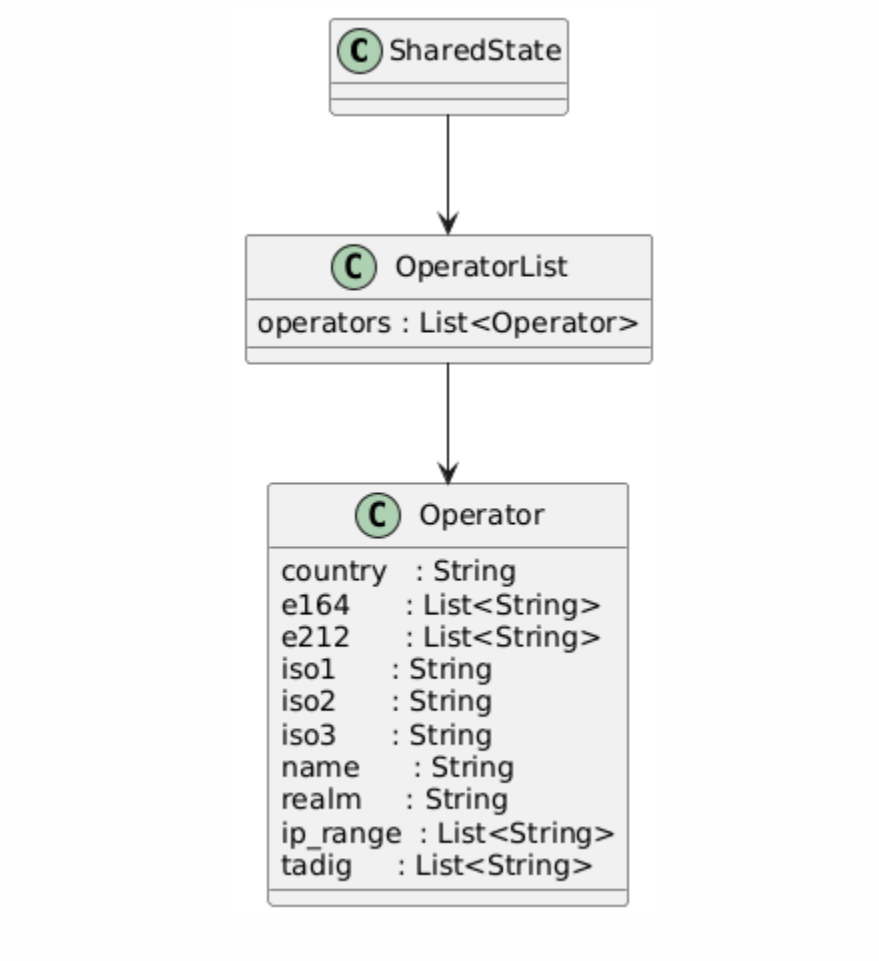
System Architecture



Requests move from the client to the API gateway, through the application layer, and finally reach the data layer.

- **Client/Consumer Layer:** External clients send requests to the API.
- **API Gateway/Control Layer:** Receives incoming requests and routes them to the right service.
- **Application Layer:**
    - Handles core CRUD operations (Create, Read, Update, Delete).
    - Offers endpoints to identify roaming partners, group operators by network size, and retrieve network names.
    - Manages validation, error handling, and business logic.
- **Data Layer:** Stores operator mapping data (operator names, TADIG codes, e212 codes, country info).

Database Design



This diagram illustrates the model for the Operator Mappings API, showing that all operator data is stored in a centralized, in-memory Operator Vector within a Shared State.

- **Shared State**: The top-level container holding all operator data for the API.
- **Operator Vector:** The in-memory list where every operator record is stored. This is the central data source used by all API endpoints.
- **Operator (Template)**: A representation of what an operator record should contain, showing the data types for each field.

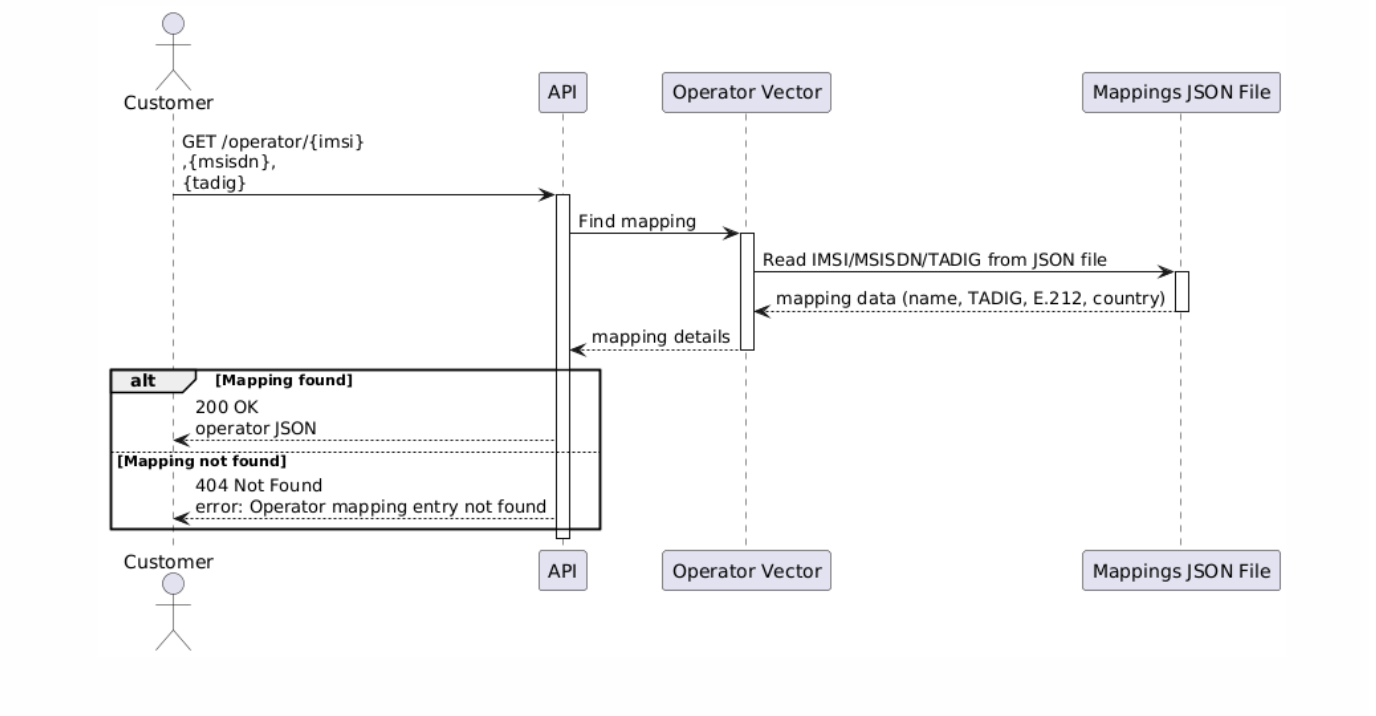
Operator Fields

- **Country**: Identifies the country where the operator is based (e.g., "Ireland").
- **E164**: A collection of E.164 formatted phone numbers or prefixes; these conform to international telephone standards.
- **E212**: Represents the mobile network codes that uniquely identify an operator in a specific region.
- **Iso2:** Represents a country code using 2 letters.
- **Iso3**: Represents a country code using 3 letters.

- **Name**: The official or commercial name of the operator.
- **realm**: A domain or logical grouping identifier, often used for routing or context differentiation in network applications.

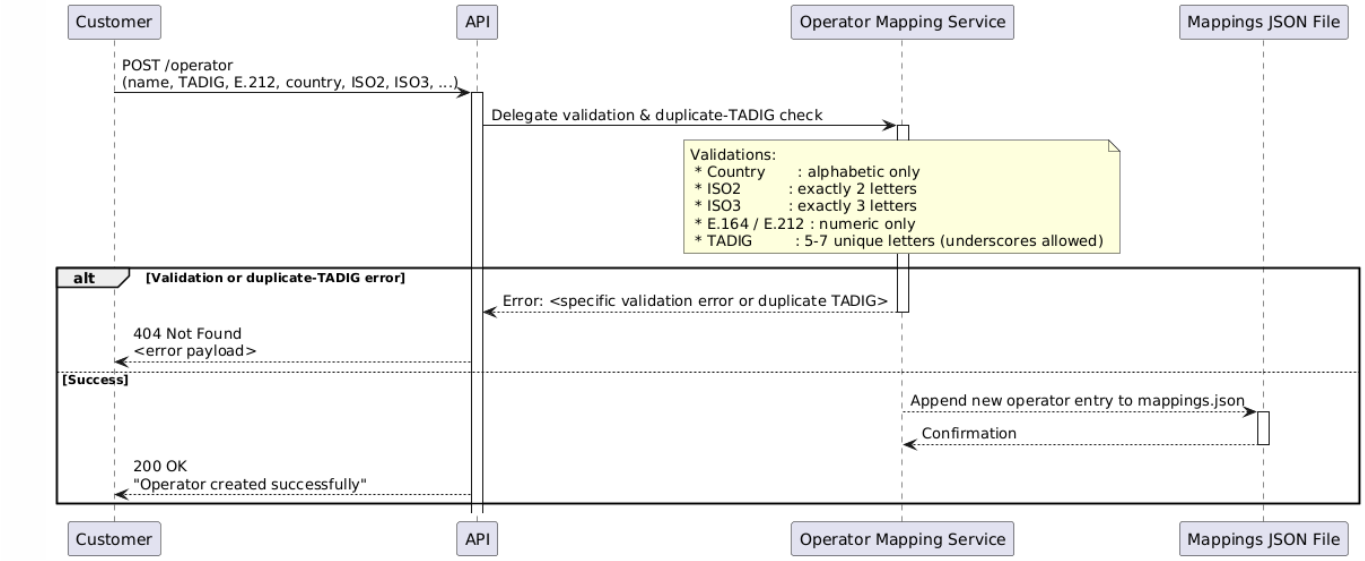
- **Ip_range**: Indicates the range of IP addresses assigned to or used by the operator.
- **TADIG**: A unique code used to identify operators

Read Operator Sequence Diagram



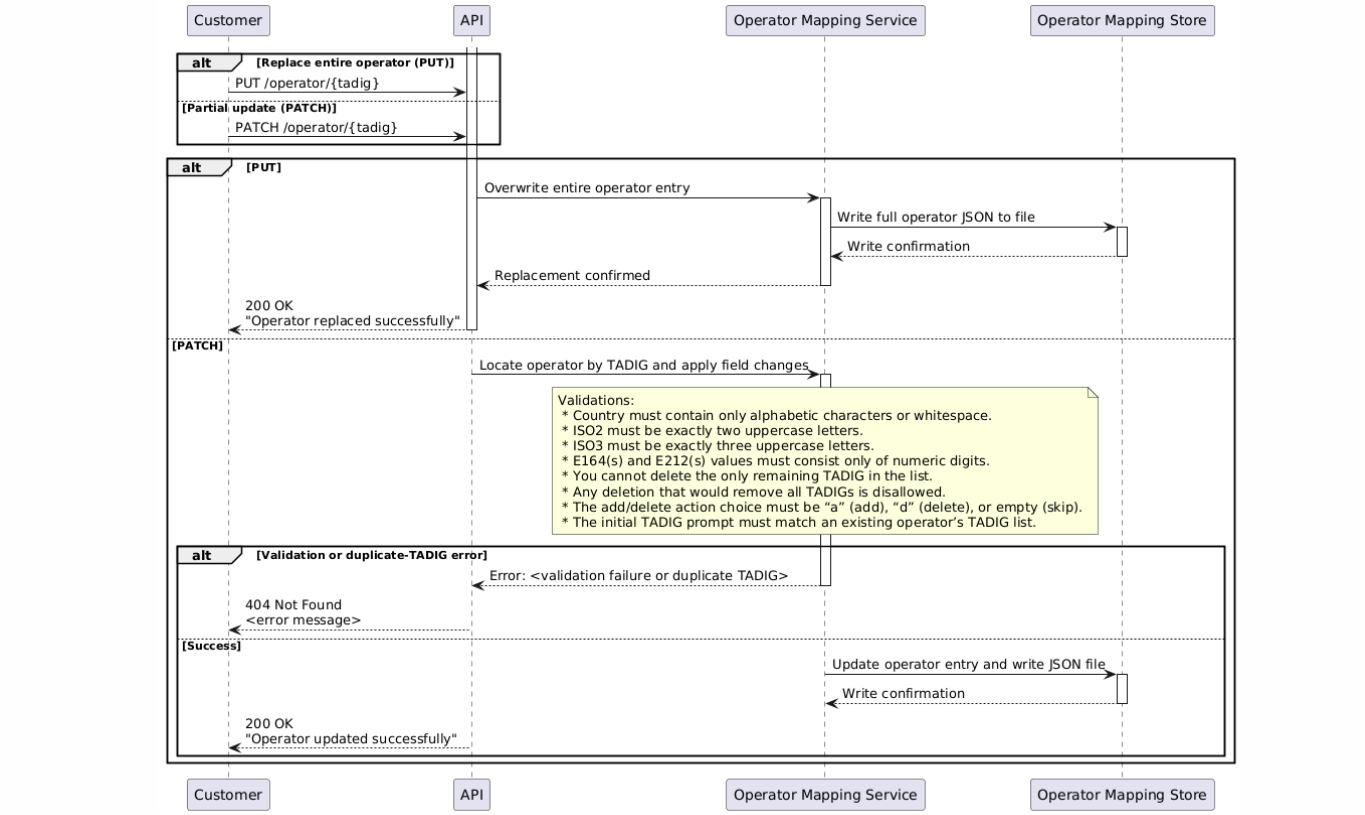
- The customer calls `GET /operator/{imsi},{msisdn},{tadig}` to look up an operator.
- The API queries its operatormapping component for a matching record.
- The component loads the IMSI, MSISDN and TADIG entries from the `mappings.json` file and performs a lookup based on those three keys.
- If a match is found, the API returns `200 OK` with the operator's JSON payload.
- If no mapping exists, the API returns `404 Not Found` with an error response.

Create Operator Sequence Diagram



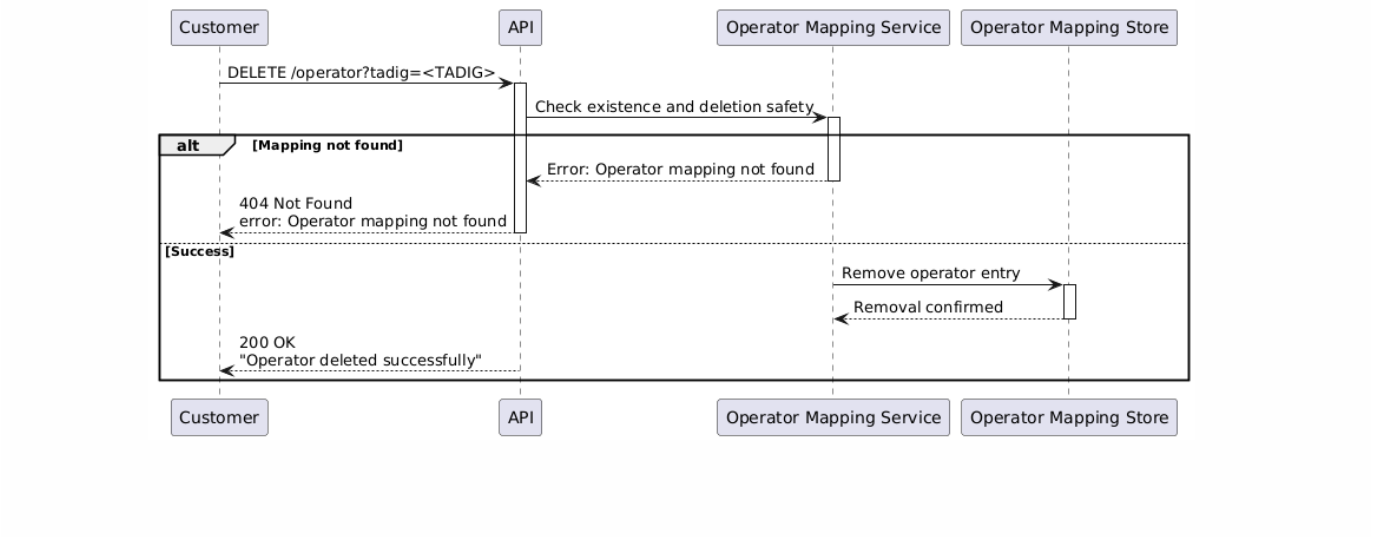
- The customer sends `POST /operator` with the new operator's details.
- The API delegates validation and duplicateTADIG checks to its operatormapping service.
- Validations include:
  Country - Can only contain alphabetical characters
  ISO2 - Contains 2 alphabetical characters
  ISO3 - Contains 3 alphabetical characters
  e164/e212 - Must only contain numerical values
  Tadig - Contains 5 - 7 Unique aphabetical characters and can also include underscores
- If validation fails or the TADIG already exists, the API returns `404 Not Found` with an appropriate error payload.
- On success, the operatormapping service creates the new entry and appends it to the `mappings.json` file.
- Finally, the API responds with `200 OK` and a "Operator created successfully" message.

Update Operator Sequence Diagram



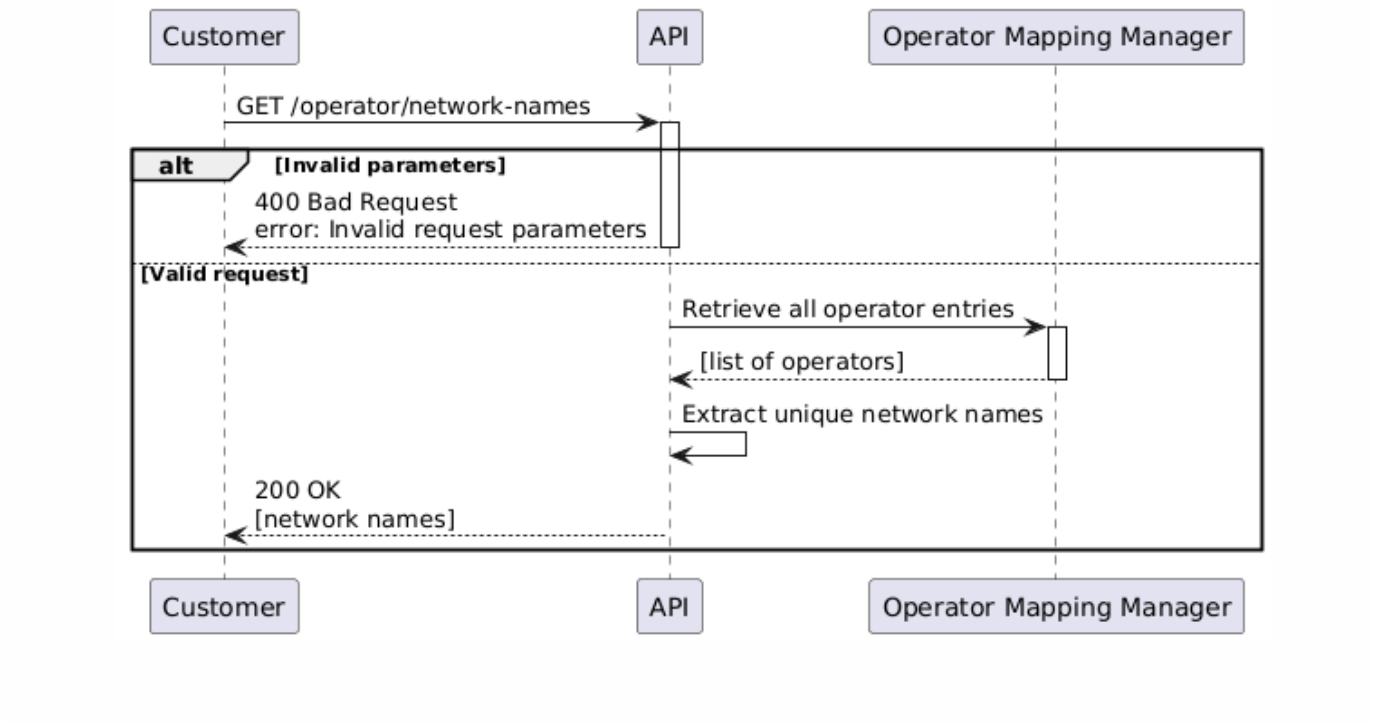
- **PUT /operator/{tadig}** completely replaces the operator record: the API instructs the mapping service to overwrite the entry, the store writes the full JSON, and the API returns **200 OK – Operator replaced successfully**.
- **PATCH /operator/{tadig}** applies only the supplied field changes: the API locates the operator by TADIG, then the mapping service runs these validations before writing:
    - Country must contain only alphabetic characters or whitespace.
    - ISO2 must be exactly two uppercase letters.
    - ISO3 must be exactly three uppercase letters.
    - E164(s) and E212(s) values must consist only of numeric digits.
    - You cannot delete the only remaining TADIG in the list.
    - Any deletion that would remove all TADIGs is disallowed.
    - The add/delete action choice must be “a” (add), “d” (delete), or empty (skip).
    - The initial TADIG prompt must match an existing operator’s TADIG list.
- If any validation or a duplicateTADIG check fails, the service returns **404 Not Found** with an error message otherwise it updates the changed fields, writes the JSON file, and the API returns **200 OK – Operator updated successfully**.

Delete Operator Sequence Diagram



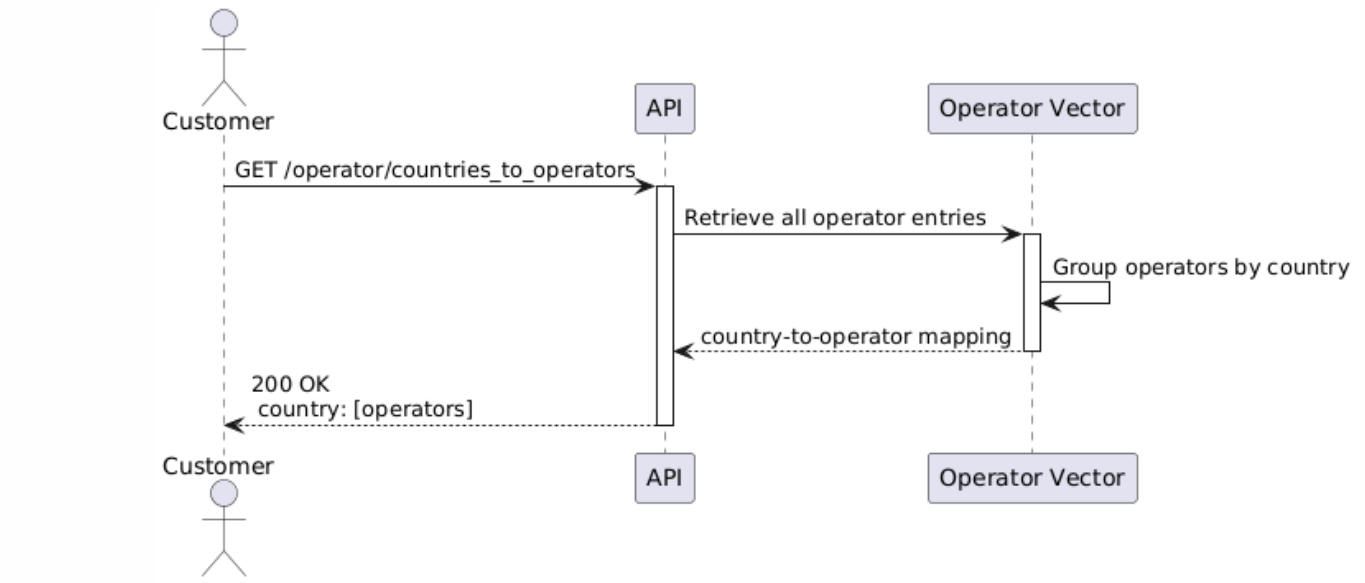
- The customer sends a `DELETE /operator` request to the API to remove a specific operator mapping.
- The API forwards this to the Operator Mapping Service, which verifies that the mapping exists and that deleting it is safe.
- If the mapping isn't found, the service returns an error and the API responds with **404 Not Found** and an error message.
- If the mapping exists, the service instructs the Operator Mapping Store to delete the entry once the store confirms removal, the API returns **200 OK** with "Operator deleted successfully."

Retrieve Network Names Sequence Diagram



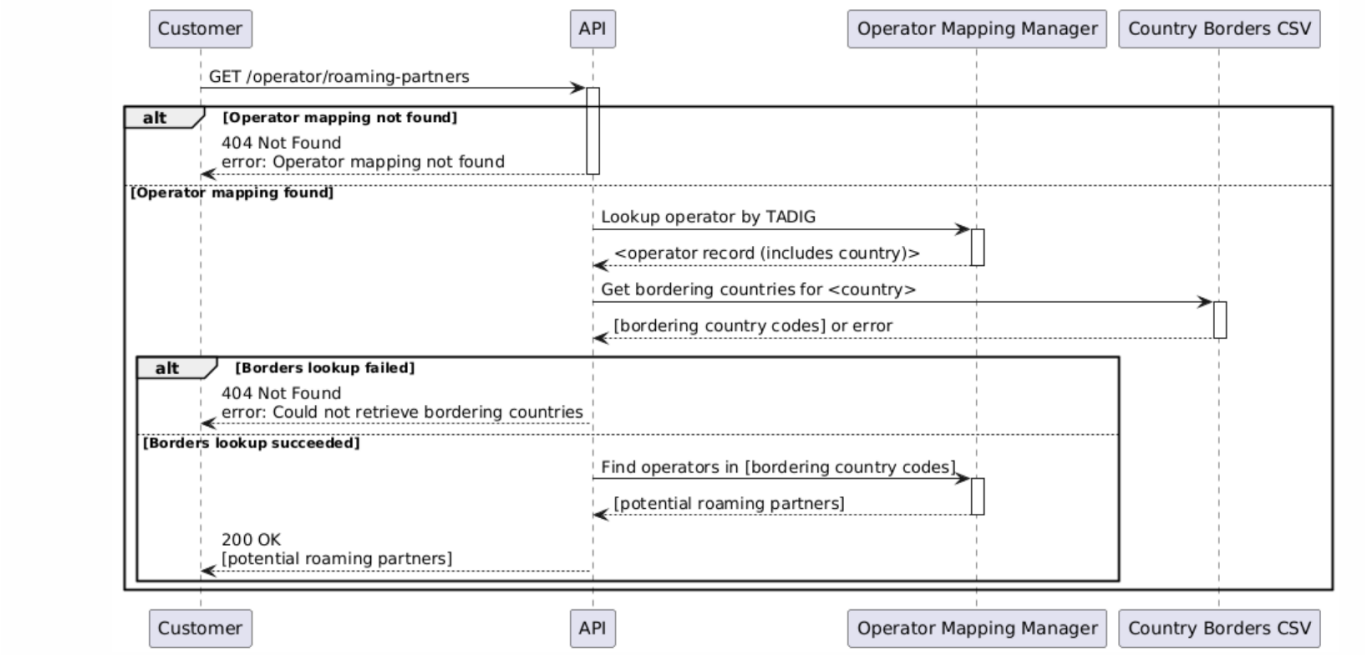
- The customer sends a `GET /operator/network-names` request to the API.
- The API first checks the request parameters and returns **400 Bad Request** if they're invalid.
- If valid, the API asks its internal Operator Mapping Manager for all operator entries.
- The manager returns the full list of operators, and the API extracts the unique network names.
- Finally, the API responds with **200 OK** and the deduplicated list of network names.

Mapping Countries to Operators Sequence Diagram



- The customer calls `GET /operator/countries_to_operators` to request a countrytooperator mapping.
- The API retrieves all operator entries from its inmemory list.
- The operator vector groups each operator by its country code.
- The API returns **200 OK** with a JSON object mapping each country to its array of operators.

Identify Potential Partners Sequence Diagram



- The customer calls `GET /api/roamingpartners` to ask for potential roaming partners.

- The API asks the Operator Mapping Manager to look up the operator by TADIG; if no mapping exists, it immediately returns **404 Not Found – Operator mapping not found**.
- If the operator is found, the API reads its country and queries the Country Borders CSV for that country's neighbors; a CSV lookup error yields **404 Not Found – Could not retrieve bordering countries**.
- When the border lookup succeeds, the API asks the Operator Mapping Manager to find all operators in those neighboring countries.
- Finally, the API responds with **200 OK** and a list of the potential roamingpartner operators.

API COMPONENT OVERVIEW

```
openapi: 3.0.3
info:
  title: Operator Mappings API
  version: 1.0.0
  description: >
    A RESTful API for managing and querying mobile network operators.
    Supports lookup by IMSI/MSISDN/TADIG, full CRUD, network-name listing,
    country mapping, grouping by E.212 size, and roaming-partner discovery.
servers:
  - url: http://{OM-API-Address}/api/v1
    description: Main API endpoint
    variables:
      OM-API-Address:
        default: 0.0.0.0:8080
        description: Host and port of the API server

paths:
  /operators:
    get:
      summary: Lookup operator by IMSI, MSISDN or TADIG
      operationId: read-operator
      parameters:
        - name: imsi
          in: query
          schema:
            type: string
            pattern: "^[0-9]+$"
          description: E.212 (IMSI)
        - name: msisdn
          in: query
          schema:
            type: string
            pattern: "^[0-9]+$"
          description: E.164 (MSISDN)
        - name: tadig
          in: query
          schema:
            type: string
          description: Unique TADIG code
      responses:
```

```
        '200':
          description: One or more matching operators
          content:
            application/json:
              schema:
                oneOf:
                  - $ref: '#/components/schemas/Operator'
                  - type: array
                    items:
                      $ref: '#/components/schemas/Operator'
        '400':
          description: Must specify exactly one of `imsi`, `msisdn`, or
`tadig`
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '404':
          description: No operator found matching the provided identifier
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

    post:
      summary: Create a new operator
      operationId: create-operator
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Operator'
      responses:
        '201':
          description: Operator created successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Operator'
        '400':
          description: Invalid request payload
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '409':
          description: Operator with given TADIG already exists
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
```

```
    put:
      summary: Replace an existing operator by TADIG
      operationId: replace-operator
      parameters:
        - name: tadig
          in: query
          required: true
          schema:
            type: string
          description: TADIG code of the operator to replace
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Operator'
      responses:
        '200':
          description: Operator replaced successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Operator'
        '400':
          description: Invalid request payload
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '404':
          description: Operator not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

    patch:
      summary: Update an existing operator by TADIG
      operationId: update-operator
      parameters:
        - name: tadig
          in: query
          required: true
          schema:
            type: string
          description: TADIG code of the operator to update
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Operator'
      responses:
```

```
        '200':
          description: Operator updated successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Operator'
        '400':
          description: Validation error or invalid payload
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '404':
          description: Operator not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '409':
          description: Conflict: attempting to set a TADIG that already
exists
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

    delete:
      summary: Delete an operator by TADIG
      operationId: delete-operator
      parameters:
        - name: tadig
          in: query
          required: true
          schema:
            type: string
          description: TADIG code of the operator to delete
      responses:
        '200':
          description: Operator deleted successfully
        '404':
          description: Operator not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /operators/network-names:
    get:
      summary: List all unique network names
      operationId: list-network-names
      responses:
        '200':
          description: Array of network names
```

```
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
      '400':
        description: Invalid request parameters
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '500':
        description: Internal server error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

/operators/country-mapping:
  get:
    summary: Map ISO3 country code to operators
    operationId: get-country-mapping
    parameters:
      - name: iso3
        in: query
        required: true
        schema:
          type: string
          minLength: 3
          maxLength: 3
          pattern: "^[A-Za-z]+$"
        description: Three-letter ISO3 country code
    responses:
      '200':
        description: Mapping of country code to operator list
        content:
          application/json:
            schema:
              type: object
              additionalProperties:
                type: array
                items:
                  $ref: '#/components/schemas/Operator'
      '404':
        description: No operators found for given ISO3
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

/operators/roaming-partners:
  get:
```

```
    summary: Discover roaming partners by TADIG
    operationId: get-roaming-partners
    parameters:
      - name: tadig
        in: query
        required: true
        schema:
          type: string
        description: TADIG code for base operator
    responses:
      '200':
        description: List of potential roaming partner operators
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Operator'
      '400':
        description: Missing or invalid TADIG
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'
      '404':
        description: Operator not found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ErrorResponse'

/operators/grouped-by-e212:
  get:
    summary: Group operators by E.212 code count
    operationId: group-by-E212
    responses:
      '200':
        description: Operators grouped into small/medium/large
        content:
          application/json:
            schema:
              type: object
              properties:
                small:
                  type: array
                  items:
                    $ref: '#/components/schemas/Operator'
                medium:
                  type: array
                  items:
                    $ref: '#/components/schemas/Operator'
                large:
                  type: array
```

```
                    items:
                      $ref: '#/components/schemas/Operator'
                required:
                  - small
                  - medium
                  - large

components:
  schemas:
    Operator:
      type: object
      properties:
        country:
          type: string
          description: ISO2 or ISO3 country code
        e164:
          type: array
          items:
            type: string
          description: List of E.164 (MSISDN)
        e212:
          type: array
          items:
            type: string
          description: List of E.212 (IMSI)
        iso2:
          type: string
          description: Two-letter ISO country code
        iso3:
          type: string
          description: Three-letter ISO country code
        name:
          type: string
          description: Operator display name
        realm:
          type: array
          items:
            type: string
          description: Realm(s) for the operator
        ip_ranges:
          type: array
          items:
            type: string
          description: IP ranges for the operator
        tadig:
          type: array
          items:
            type: string
          description: One or more TADIG codes
      required:
        - country
        - iso2
        - iso3
```

```
      - tadig

  ErrorResponse:
    type: object
    properties:
      code:
        type: integer
        description: HTTP status code
      message:
        type: string
        description: Descriptive error message
    required:
      - code
      - message
```

TECHNICAL RISKS

Scalability Issues
• As the number of operator entries grows, the in-memory data structure (e.g., a Vec<Operator>) may face performance bottlenecks.
• Large datasets may challenge the efficiency of CRUD operations.

Concurrency and Synchronization Risks
• Simultaneous updates or deletions by multiple users can lead to race conditions if not managed properly.

IMPORTANT TECHNICAL DECISIONS

Tokio

Pros

- The de-facto async runtime with the widest library support
- Modular I/O primitives and scheduler options

Cons

- Steeper learning curve due to many features
- Can bloat binary size with all features on

Choice

Made a decision to use Tokio for rust as it supplies reliable asynchronous applications such async I/O, networking and scheduling etc.

FUNCTIONAL REQUIREMENTS

OM-1: Implement CRUD Operations for Operator Mappings

- **Create:** Validate inputs, perform file locking, and add a new operator record via the API.
- **Read:** Confirm operator existence and retrieve its details, returning appropriate error messages when necessary.
- **Update:** Validate and modify operator data, ensuring exceptions and edge cases are handled through the API.
- **Delete:** Verify the operator exists and remove the record, implementing robust error handling.

OM-2: Return All Network Names

- Load and parse the operator mapping data, then extract and return all network names via the API while managing errors.

OM-3: Return Mapping of Country to Operators

- Build a data structure mapping each country to its associated operators, and provide this mapping as an API response.

OM-4: Return Operators Grouped by Network Size

- Group operators into small, medium, and large categories based on the number of e212 codes they manage, then return these groups through the API.

OM-5: Return Potential Roaming Partners by TADIG Code

- Parse adjacent country data, match operators in bordering countries based on the provided TADIG code, and return a list of potential roaming partners via the API.

NON FUNCTIONAL REQUIREMENTS

Maintainability: The system should be designed and documented in a way that facilitates easy updates and modifications. This includes clear, modular code, comprehensive documentation, and coding standards that ensure future enhancements and bug fixes can be performed efficiently.

Reliability: The API must perform consistently under various conditions, correctly handling unexpected inputs and gracefully recovering from errors. Comprehensive testing, robust error logging, and effective exception management are essential to maintain a trustworthy system.

Scalability: The design should support growth in data volume and user demand without significant performance degradation. The system architecture needs to allow for scaling, both in terms of adding more operator records and handling concurrent API requests, ensuring sustained performance as the load increases.

Fault Tolerance: The system should remain operational in the face of component failures or unexpected disruptions. It must include mechanisms to detect faults, automatically recover from errors, and isolate issues so that a failure in one part of the system does not affect overall service delivery.

Availability & Redundancy: High availability is critical, the API must be accessible to users around the clock with minimal downtime. This involves incorporating redundancy in critical components—such as using backup servers or clusters and establishing reliable data recovery and failover processes to ensure continuous operation.

Configurability: Key parameters such as file paths, port numbers etc must be externalized via environment variables or config files, allowing the service to adapt to different environments without code changes.

FUTURE ADDITIONS

- Containerize the API with Docker and define Kubernetes manifests for scalable deployments.
- Implement a CI/CD pipeline to automate testing, linting, and deployments to staging and production.
- Build a simple web dashboard using React and integrate Swagger UI for interactive API documentation.
- Migrate from file-based storage to a managed database.
- Provide bulk import/export endpoints and data versioning to support integrations and backward-compatible updates.