

# MOSIP DEMOGRAPHIC DEDUPLICATION ENGINE

- 1. INTRODUCTION
- 2. MOTIVATION
- 3. CURRENT ARCHITECTURE
- 4. REQUIREMENT & SOLUTION APPROACH
  - 4.1 Requirement
    - 4.1.1 Identify suitable text fuzzy match algorithm preferably based on phonetics
    - 4.1.2 Short list most suitable 2 approaches for development, review with mentors and finalise the approach
    - 4.1.3 Develop the software as per the selected approach(s)
  - 4.2 Proposed solution approach
  - 4.3 Development Environment
  - 4.4 Deployment, Testing & Submission
- 5. DELIVERABLES
- 6. APPENDIX
  - 6.1 Reference Repos & URLs
    - Documents Reference
    - Github

## 1. INTRODUCTION

One of the key process in MOSIP registration is De Duplication. De Duplication is a solution to uniquely identify a person for. given set of attributes such as biometric attributes such as fingerprint, Iris, voice etc. or using combinations of demographic attributes such as name, gender, date of birth , location etc.

MOSIP supports two types of de duplications - Demographic and Biometric. It provides configurable process flow where one could configure multi-staged deduplication. MOSIP platform has is a reference implementation of Demographic Deduplication available at [github](#) which uses hashing technique to achieve the fuzzy matching functionality.

Key Information	Detail
Complexity level	medium
Team size	3-4
Timeline	Not more than 45 days
Tech Stack	Java/spring boot/vertex/REST/JSON various fuzzy text match techniques, phonetics
MOSIP target version	Branch 1.2.x

## 2. MOTIVATION

The reference implementation uses a more generic hashing technique which may not be very efficient and languages add a new dimension to the technique. There are several algorithms and approaches being used for fuzzy text match.

Hence we need a solution which evaluates available alternate mechanisms and implement one or two of them for at least 2 languages english and either french or arabic.

### 3. CURRENT ARCHITECTURE

MOSIP Registration processor is designed on event driven architecture, SEDA and uses Apache Vertex. Please refer to this [document](#) for more information. All the modules are built as stages which receive events from vertex. Kafka is configured as event bus. There is a powerful, highly configurable workflow engine built using Camel bridge, which allows to configure several work flows out of these stages. event/request flow from one stage another as per the workflow and matching criteria. The new Demo dedupe module should be developed adhering to same architecture and should be pluggable into Registration processor workflow.

The Reference demo dedup implementation uses postgres database engine

### 4. REQUIREMENT & SOLUTION APPROACH

#### 4.1 Requirement

##### 4.1.1 Identify suitable text fuzzy match algorithm preferably based on phonetics

- Study, explore and identify pros and cons of various text matching algorithms such as double metaphone, Levenshtein distance or any other.
- Also keep in mind language requirements while doing the study.
- English language support is mandatory requirement. In addition to english, either French or Arabic need to be supported.
- We need to ensure privacy to be preserved. Personally identifiable information should not be stored in plain text format.
- We may consider following demographic attributes for de-deduplication - Name ( first, last ), address, gender, date-of-birth, location.

##### 4.1.2 Short list most suitable 2 approaches for development, review with mentors and finalise the approach

- We may consider following factors to decide - ability to support the required languages, accuracy and performance

##### 4.1.3 Develop the software as per the selected approach(s)

- Create required test data set to ensure a decent coverage in targeted languages
- Develop test scripts
- Develop the solution as detailed below
- You may propose a suitable database engine and structure

#### 4.2 Proposed solution approach

- To build and test the algorithms a stand alone Java/Spring boot application with RESTful API could be designed and developed. This may expose demo-*dedup* API.
- The demographic data will be passed in the format of ID-Json, a Json structure containing the attributes
- This API should take reference-id and ID-Json as input parameters
- Develop two methods in service class as follows -
  - **Match**
    - This method should implement the match logic
    - Takes ID-Json as input, compute the algorithm specific data and implement match logic
    - It should return confidence score, and based on configured threshold decide whether it is a duplicate or not.
    - If it is not duplicate
      - Insert the details using insert method
  - If one or more duplicates found, return the list of reference-ids of the duplicates
  - **Insert**
    - Insert the required data into demo dedup storage
      - This takes following parameters
        - Id - a unique reference ID of the persona

- Algorithm specific data derived from ID-Json

### 4.3 Development Environment

- Setup a local development for Java/Springboot, local database engine and required libraries
- Prepare test data in the targeted languages, preferably as Id-Jsons and some unique reference ids
- Build a test script to run the tests and publish the result

### 4.4 Deployment, Testing & Submission

- There is no deployment requirement
- commit all your artefacts into the forked repo and raise pull request

## 5. DELIVERABLES

- source code
- project files
- test data
- test script & test reports

## 6. APPENDIX

### 6.1 Reference Repos & URLs

#### Documents Reference

 [Registration Processor](#)

 [Metaphone](#)

 <https://github.com/slacy/double-metaphone/blob/master/README.md> [Connect your Github account](#)


Levenshtein distance

 [Levenshtein distance](#)

 [F.17. fuzzystmatch — determine string similarities and distance](#)

 [Phonetic](#)

#### Github

 <https://github.com/mosip/registration/tree/release-1.2.0.1/registration-processor/core-processor/registration-processor-demo-dedupe-sta/ge/src/main> [Connect your Github account](#)