Software Requirements Specification

Product name: Aldo Moro Digital Edition – location script

Project Manager: Marialaura Vignocchi

Requirement analyst: Erica Andreose - Pasquale Leuzzi

History of revisions:

1. Introduction

1.1. Document scope

The paper analyzes the software developed to make up for a gap present in the creation of the RDF during the processing of new documents for the National Edition of the Works of Aldo Moro. Within the RDF, created through the "generate.py" and "align.py" scripts, prefixes are called that are unused. These prefixes are: geonames (http://www.geonames.org/ontology#) and wgs84_pos (http://www.w3.org/2003/01/geo/wgs84_pos#). After a quick check within the old RDF file, it could be seen that these prefixes created descriptive triples for the places mentioned in the documents. They inserted longitude, latitude and a symbol for contextualization of the place itself (sorting between cities, states, continents, seas etc.). This part of RDF was not being created automatically through the scripts we had but was being given to an external entity for processing. To make up for this lack I worked on creating a python script named "*location.py*". The purpose of the script is to obtain the geographic coordinates of a given location using the GeoNames API. It initially normalizes the name of the location, removing spaces and superfluous text. It then looks for the coordinates in the local cache of previously obtained data, if available. If they are not in the cache, it makes a request to the GeoNames API to obtain the coordinates. If the request is successful, the coordinates are stored in the local cache for faster access in the future. If the request fails or does not return valid data, the error is logged in a separate file for future management. "Location.py" integrates with the functioning of "*generate.py*" and "*align.py*" for the creation of descriptive triples within the RDF. With "*generate.py*" you insert it by creating the triples for each place indicated in the MongoDB metadata. While with "*align.py*" it comes into operation for every place marked by researchers and included in the list of places mentioned by Aldo Moro in his speeches.

Definitions, acronyms, and abbreviations

- ENOAM: Edizione Nazionale delle Opere di Aldo Moro

- KwicKwocKwac: web application for labeling documents generating RDFa and TEI output, developed for the digital edition of Aldo Moro opera omnia.

1.2. References

- Showcase website: https://site.unibo.it/edizione-nazionale-moro/it
- ENOAM website: https://aldomorodigitale.unibo.it/
- ENOAM documentation: https://aldomorodigitale.unibo.it/about/docs/results#others-section
- GeoNames API: https://www.geonames.org/export/web-services.html
- wgs84_pos: https://www.w3.org/2003/01/geo/wgs84_pos

2. General description

2.1. Product scope

The software serves as a geographic coordinates retrieval tool, aimed at analyzing the place names tagged by researchers within Aldo Moro's documents. The places contained within the tags are identified through parsing within the "generate.py" and "align.py" generating functions, the location software takes care of efficiently obtaining precise geographical coordinates (and other information) for all locations. This final set of information that is inserted into the RDF file allows correct functioning and display of the interactive map present in the advanced search indices of the site dedicated to the National Edition of Aldo Moro's Works.

2.2. Product Features

Location Normalization: Automatically standardizes input locations to ensure uniformity and accuracy in coordinate retrieval. The "*normalize_location*" function in Python takes a location string as input and performs several normalization steps to ensure consistency and remove unnecessary details. It converts the input string to lowercase, removes leading and trailing whitespace, and checks if the length is sufficient. If there's a comma present, it extracts the primary part of the location. It also removes text within parentheses and any remaining leading or trailing whitespace. Finally, it returns the normalized location string, making it suitable for further processing or comparison.

Caching Mechanism: Stores previously queried locations and their coordinates locally in a json file to expedite subsequent requests and reduce API calls.

GeoNames API Integration: Utilizes the GeoNames API to retrieve geographic coordinates for specified locations, leveraging its extensive database and search functionalities.

Error Handling: Implements robust error handling mechanisms to manage API request failures or invalid responses, ensuring reliability and user confidence. In case of error it saves the name of the problematic location so that it can then be intervened manually.

Scalability: Designed with flexibility to accommodate future enhancements and scalability requirements, ensuring long-term viability and adaptability. The information saved in the json file is varied and allows for a possible future implementation thanks to the easy reuse of the metadata reported by geonames.

2.3. Domain models

The software operates within the domain of geospatial data processing and retrieval. Key concepts include geographic coordinates (latitude and longitude), location normalization, caching mechanisms, and integration with external geospatial data providers such as GeoNames.

2.4. System architecture

Input Processing Module: Responsible for receiving and processing provided location inputs. Includes functions such as *normalize_location(location)* to normalize input locations by standardizing their format and removing extraneous characters.

Data Management Module: Manages the storage and retrieval of location data, cache entries, and error logs. Utilizes JSON files (*geonames_data.json* and *errori.json*) for persistent storage of location data and error information.

GeoNames API Integration Module: Facilitates integration with the GeoNames API to retrieve geographic coordinates for specified locations. Constructs API requests, sends HTTP GET requests to the GeoNames API, and processes API responses to extract relevant location data. Handles interactions with the GeoNames API, including authentication using a user-provided username and encoding of request parameters.

Error Handling Module: Manages error handling and logging mechanisms to handle exceptional conditions encountered during the execution of the software. Logs errors and exceptions to the *errori.json* file for future reference and analysis. Ensures robustness and reliability by appropriately handling errors such as API request failures or invalid responses.

Caching Mechanism Module: Implements a caching mechanism to store previously queried locations and their corresponding coordinates locally. Reduces reliance on external API calls by facilitating quicker retrieval of location data from the cache for subsequent requests.

Interactions:

The Input Processing Module receives location inputs from generate.py and align.py and passes them to the Data Management Module for processing.

The Data Management Module interacts with the GeoNames API Integration Module to retrieve geographic coordinates for specified locations. It also interacts with the Error Handling Module to log errors encountered during execution.

The GeoNames API Integration Module communicates with the GeoNames API to retrieve location data, processes API responses, and extracts relevant information such as latitude, longitude, and address type.

The Error Handling Module handles exceptional conditions and logs errors encountered during the execution of the software.

The Caching Mechanism Module interacts with the Data Management Module to store and retrieve location data from the cache, reducing reliance on external API calls and improving performance.

This modular architecture promotes maintainability, scalability, and extensibility by separating concerns and encapsulating functionality within discrete components.

2.5. General constraints and assumptions

API Usage Limitations: The software assumes compliance with usage limits and terms of service imposed by external APIs, as GeoNames, regarding API usage quotas, rate limits, and data licensing.

Correctness of marked documents: generate.py and align.py are expected to provide accurate and valid location inputs for successful coordinate retrieval, and cooperation with API usage guidelines is presumed for uninterrupted service availability.

3. Requirements list

3.1. Functional requirements

3.1.1 Req 1: Location Normalization

The system shall normalize provided location inputs to ensure uniformity and consistency. The normalization process shall convert location inputs to lowercase and remove leading/trailing whitespace. Location inputs containing commas shall be split, and only the first part shall be considered for normalization. Text within parentheses in location inputs shall be removed during normalization.

3.1.2 Req 2: Coordinate Retrieval

The system shall retrieve geographic coordinates (latitude and longitude) for specified locations using the GeoNames API. Location coordinates shall be obtained by sending

HTTP GET requests to the GeoNames API with the normalized location as a query parameter. The system shall parse the JSON response from the GeoNames API and extract the latitude, longitude, address type, country code, and toponym name.

3.1.3 Req 3: Filtering Feature Classes

The system should iterate through the list of geographic names, filter them based on their feature class, handle historical features by changing their classification, and select the first suitable item as the result. The software should check if the feature class is one of the following: 'L' (administrative boundaries), 'A' (administrative areas), 'P' (populated places), or 'H' (historical features). This process helps in selecting the most relevant and appropriate geographic name based on specific criteria. This subdivision reflects the current functioning of the interactive map which distinguishes the points of the various places in different colors.

3.1.4 Req 4: Caching Mechanism

The system shall implement a caching mechanism to store previously queried location data locally. Cached location data shall be used to expedite subsequent requests for the same location, reducing reliance on external API calls.

3.1.5 Req 5: Error Handling

The system shall handle errors encountered during location normalization, coordinate retrieval, and API interactions. Error handling mechanisms shall include logging errors to files (errori.json) and providing informative error messages to users. HTTP request errors, invalid API responses, and unexpected data formats shall be handled gracefully to ensure system robustness.

3.2. Non-Functional requirements

3.2.1 Req 1: Performance

The system shall be capable of handling a high volume of location queries efficiently. Response times for location normalization and coordinate retrieval shall be optimized to minimize user wait times. The caching mechanism shall improve system performance by reducing the number of external API calls and improving response times.

3.2.2 Req 2: Reliability

The system shall exhibit high reliability in retrieving accurate geographic coordinates for specified locations. Error handling mechanisms shall ensure that errors are captured and managed effectively, minimizing disruptions to system functionality.

4. Verification and Validation

The script was tested and used for the current upload present on the Aldo Moro Digitale website, which has 493 locations used within the interactive map. The functioning of the map is put into operation by parsing the rdf file, containing all the information necessary for the placement of the points on the map, their classification (displayed in the color choice) and recursion count. The "geonames_data.json" and "errori.json" files contain respectively: all successful calls with related information useful for data research and quality control, all inputs that generate errors and which must be reviewed at a later time.

Some control tests were also developed and executed manually by the user by running the script from the command line. The script contains multiple test cases, each representing a scenario that the get_coordinates function might encounter. These scenarios include:

Testing with a valid location.

Testing with an invalid location.

Testing with a location that is already cached.

Testing with a location that resulted in an error previously.

Each test case is executed sequentially. The get_coordinates function is called with the corresponding test input, and the result (either coordinates or an error message) is printed to the console. The user observes the printed output to verify if the behavior matches the expected behavior. If the coordinates are returned for a valid location, it indicates that the function is functioning correctly. If an error message is returned for an invalid or previously errored location, it indicates that the function is handling errors appropriately. The user validates the results of the tests manually. This involves comparing the actual output with the expected output based on the scenarios being tested. Any discrepancies between the actual and expected behavior are noted for further investigation.