



# MAPPING LOCATIONS IN TEXTS

Supervised by Chris B. Jones | Moderated by Usashi  
Chatterjee

CM3203 – Individual Project. School of Computer Science, Cardiff University

Charlie Webb | 14/05/2021  
webbcj1@cardiff.ac.uk

## Abstract

In modern times we often find that, while the data of today is standardised and helpful, older data is less structured and, as such, may only contain small bits of information pertaining to the subject matter. This issue presents us the task of filling in this missing information in order to ensure all data is correct and up to date.

In this regard, we consider the example of digitising collections for ecological documentation and preservation, in which older documents may only contain natural language descriptions of specimen locations. These older documents often lack clear coordinates for the specimen location, and as such present a challenge in mapping the spatial and temporal locations of various species.

To make an attempt at solving this problem (and others like it), this paper presents the project AIGlobe – a web application designed to employ machine learning methods for Named Entity Recognition (NER) in order to tag locations in natural language captions, then resolve those entities into definite locations and map them on an interface for users to explore.

This paper will document the development process of AIGlobe, as well as evaluating its performance and defining the successes and difficulties of this project.

## Acknowledgements

I would like to thank Chris Jones for his guidance and enthusiasm during this project, especially during the COVID-19 pandemic and across many thousands of miles.

I would also like to thank my family and friends for listening to me talk unendingly about this project.

## Table of Contents

|  |    |
|--|----|
| Abstract.....  | 1  |
| Acknowledgements.....                                  | 1  |
| Table of Contents .....                                | 2  |
| Table of Figures.....                                  | 5  |
| 1 - Introduction .....                                 | 7  |
| 2 – Background .....                                   | 8  |
| 2.1 - The Issue at Hand.....                           | 8  |
| 2.2 - Stakeholders .....                               | 8  |
| 2.3 - An Introduction to NLP and NER.....              | 8  |
| 2.4 – Statistical NLP Models.....                      | 9  |
| 2.5 - Geo-Geo- and Non-Geo-Geo-Ambiguity.....          | 9  |
| 2.6 - Relative References and Project Constraints..... | 9  |
| 2.7 – Evaluative Measures .....                        | 10 |
| 2.8 - Existing Solutions .....                         | 10 |
| 2.9 – Data Provided.....                               | 11 |
| 3 - Justification of Tools .....                       | 12 |
| 3.1 - NER Tools .....                                  | 12 |
| 3.2 - Geocoding Tools and API Choice .....             | 14 |
| 3.3 – DBMS.....  | 15 |
| 3.4 – Web Framework and Python .....                   | 16 |
| 4 - Specification and Design .....                     | 17 |
| 4.1 - Features .....                                   | 17 |
| 4.1.1 – Single Geocoding Capability and Biasing .....  | 17 |
| 4.1.2 – Database Integration.....                      | 17 |
| 4.1.3 – Search Capabilities .....                      | 17 |
| 4.1.4 – Bulk Geocoding with File Uploads.....          | 17 |
| 4.1.5 – Exporting .....                                | 17 |
| 4.2 – User Interface.....                              | 18 |
| 4.3 – Data Flow .....                                  | 21 |
| 4.4 – System Algorithms .....                          | 22 |
| 4.4.1 – Vincenty Distance.....                         | 22 |
| 4.5 – Static Architecture of the System .....          | 22 |
| 4.6 – Small Features.....                              | 25 |
| 4.6.1 – Caption Chaining .....                         | 25 |
| 4.6.2 – Header Minimisation.....                       | 26 |

|   |    |
|---|----|
| 5 – Implementation .....                                  | 27 |
| 5.1 – Geocoding and NER.....                              | 27 |
| 5.1.1 – NER.....  | 27 |
| 5.1.2 – Forward Geocoding.....                            | 27 |
| 5.1.3 – Country Biasing .....                             | 28 |
| 5.2 – DB Fetch and Commit .....                           | 29 |
| 5.2.1 – Fetch.....  | 29 |
| 5.2.2 – Commit.....                                       | 29 |
| 5.3 – Search.....   | 30 |
| 5.3.1 – Standard Search .....                             | 30 |
| 5.3.2 – Radial Search.....                                | 30 |
| 5.4 – File Uploads.....                                   | 31 |
| 5.5 – Front End.....                                      | 33 |
| 5.5.1 – Add Marker and Add Marker Array .....             | 33 |
| 5.6 – Custom NLP Models .....                             | 33 |
| 5.7 – Unforeseen Issues in Development .....              | 34 |
| 5.7.1 – Rate Limiting .....                               | 34 |
| 5.7.2 – Hierarchical Addresses and Entity Condensing..... | 36 |
| 6 – Results and Evaluation .....                          | 37 |
| 6.1 – Custom NER Model Vs. Pre-trained NER Model .....    | 37 |
| 6.2 – Evaluation of Requirements.....                     | 38 |
| 6.3 – Time Performance.....                               | 38 |
| 6.3.1 – Single Geocoding.....                             | 39 |
| 6.3.2 – Custom NLP Times versus Standard NLP Times .....  | 39 |
| 6.3.3 – Bulk File Geocoding.....                          | 39 |
| 6.3.4 – Searching.....                                    | 40 |
| 6.4 – Evaluation of Tools Used.....                       | 40 |
| 6.4.1 – Python .....                                      | 41 |
| 6.4.2 – SpaCy.....  | 41 |
| 6.4.3 – GeoPy and Nominatim .....                         | 41 |
| 6.5 – Evaluation of Usability .....                       | 42 |
| 6.5.1 – Visibility of System Status .....                 | 42 |
| 6.5.2 – Match Between System and The Real World.....      | 42 |
| 6.5.3 – User Control and Freedom.....                     | 43 |
| 6.5.4 – Consistency and Standards .....                   | 43 |
| 6.5.5 – Error Prevention.....                             | 44 |



|  |    |
|--|----|
| 6.5.6 – Recognition Rather than Recall.....                          | 44 |
| 6.5.7 – Flexibility and Efficiency of Use.....                       | 44 |
| 6.5.8 – Aesthetic and Minimalist Design.....                         | 45 |
| 6.5.9 – Help Users Recognise, Diagnose, and Recover from Errors..... | 46 |
| 6.5.10 – Help and Documentation .....                                | 46 |
| 6.6 – Testing .....  | 47 |
| 6.6.1 – Single Geocoding.....  | 47 |
| 6.6.2 – Bulk Geocoding .....   | 48 |
| 6.6.3 – Export Session .....   | 48 |
| 6.6.4 – Searching .....  | 49 |
| 7 – Future Work and Alternative Solutions .....                      | 50 |
| 7.1 – Document Clustering .....                                      | 50 |
| 7.2 – Database Choice.....   | 50 |
| 7.3 – Geocoding API Choice .....                                     | 50 |
| 7.4 – Multithreading .....   | 50 |
| 7.5 – Geospatial Indexing.....                                       | 51 |
| 7.6 – Georeferencing Accuracy .....                                  | 51 |
| 7.7 – Future Opportunities .....                                     | 51 |
| 8 – Conclusions .....  | 52 |
| 9 – Reflection .....   | 53 |
| 10 – References.....   | 54 |

## Table of Figures

|   |    |
|---|----|
| Figure 1 - An example of the Landcare Data.....                                       | 11 |
| Figure 2 - An example of the NMW data .....   | 11 |
| Figure 3 - A Comparison of Flair and SpaCy .....                                      | 13 |
| Figure 4 - F1 Scores for SpaCy, Stanza, and Flair .....                               | 13 |
| Figure 5 - Comparison of the Landcare and NMW dataset schemas .....                   | 15 |
| Figure 6 - Initial UI Wireframes.....   | 18 |
| Figure 7 - Main Page Wireframe Equivalent.....  | 19 |
| Figure 8 - Geocoding Wireframe Equivalent.....  | 19 |
| Figure 9 - Search Wireframe Equivalent .....  | 20 |
| Figure 10 - File Upload Wireframe Equivalent.....                                     | 20 |
| Figure 11 - Radial Search Wireframe Equivalent .....                                  | 20 |
| Figure 12 - Level 0 DFD (Context Diagram).....  | 21 |
| Figure 13 - Level 1 DFD .....   | 21 |
| Figure 14 - Server Module Map .....   | 22 |
| Figure 15 - /geo Event.....   | 23 |
| Figure 16 - /fetch Event .....  | 23 |
| Figure 17 - Raw Return Data .....   | 23 |
| Figure 18 - All Possible Client-Server Interactions.....                              | 24 |
| Figure 19 - Caption Chaining for Multiple-Entity-Captions.....                        | 25 |
| Figure 20 - Minimised Header.....   | 26 |
| Figure 21 - Maximised Header .....  | 26 |
| Figure 22 - Loading the Model .....   | 27 |
| Figure 23 - NLP Function Call .....   | 27 |
| Figure 24 - Example of a Tagged Caption.....  | 27 |
| Figure 25 - Initialising the Geocoder .....   | 27 |
| Figure 26 - Geocoding and Structuring the Return Data .....                           | 28 |
| Figure 27 - Fuzzy Search for the Bias Country .....                                   | 28 |
| Figure 28 - Fetching 60 Captions from the DB.....                                     | 29 |
| Figure 29 - Committing the Session to DB .....  | 29 |
| Figure 30 - Flat Searching.....   | 30 |
| Figure 31 - A 500km Radial Search Window .....  | 30 |
| Figure 32 - 500KM at different latitudes.....   | 31 |
| Figure 33 - Radial Searching Algorithm.....   | 31 |
| Figure 34 - Creating a FormData Object .....  | 31 |
| Figure 35 - POSTing the Request.....  | 32 |
| Figure 36 - Adding a Marker .....   | 33 |
| Figure 37 - Adding a Marker Array.....  | 33 |
| Figure 38 - A Manually Annotated Caption.....   | 34 |
| Figure 39 - Training a Custom NLP Mode .....  | 34 |
| Figure 40 - The F1 score of the en_aiglobe NLP model .....                            | 37 |
| Figure 41 - The F1 score of the en_core_web_trf model .....                           | 37 |
| Figure 42 - Using en_aiglobe to entity condense .....                                 | 38 |
| Figure 43 - Using en_core_web_trf to parse a hierarchical caption.....                | 38 |
| Figure 44 - Unnecessary data in the graphical output when using en_core_web_trf ..... | 38 |
| Figure 45 - Time Taken for Natural Language Processing of "Pakaraka, Northland" ..... | 39 |
| Figure 46 - Time Taken to Geocode "Pakaraka, Northland" .....                         | 39 |

|  |    |
|--|----|
| Figure 47 - Time Taken for Natural Language Processing of "Pakaraka, Northland" using<br>en_core_web_trf ..... | 39 |
| Figure 48 - Input File for Bulk Geocoding Time Test.....   | 40 |
| Figure 49 - Time taken to process a 5-line file .....  | 40 |
| Figure 50 - Flat Searching with Query "England" .....  | 40 |
| Figure 51 - The location of the Radial Search Window .....   | 40 |
| Figure 52 - Radial Searching over the UK.....  | 40 |
| Figure 53 - Successful Search Notification .....   | 42 |
| Figure 54 - Geocoding Form with Exit Cross in Top Right Corner .....   | 43 |
| Figure 55 - Preventing Empty Caption Submission .....  | 44 |
| Figure 56 - Minimum Information Displayed.....   | 45 |
| Figure 57 - Maximum Information Displayed .....  | 45 |
| Figure 58 - Error Message for No File Submitted .....  | 46 |
| Figure 59 - Error Message for No New Points Found .....  | 46 |
| Figure 60 - Demonstration of Single Geocoding.....   | 47 |
| Figure 61 - Bulk Geocoding Demonstration.....  | 48 |
| Figure 62 - Demonstration of Export Feature.....   | 48 |
| Figure 63 - Search Demonstration .....   | 49 |
| Figure 64 - Radial Search Demonstration .....  | 49 |

## 1 - Introduction

The task of digitising collections is, objectively, an arduous task. Many museums (and other organisations) across the globe are in possession of incredible amounts of historic data, and tens of thousands of hours of manpower have been invested into digitising and modernising these repositories.

With the advent and popularisation of machine learning techniques, this task is becoming more and more viable to hand off to machines. Instead of manually annotating each record, it is now possible to hand huge collections of records off to a deep learning algorithm and allow it to automatically process in 10 minutes what one person could process in a day.

This project aims to process these natural language captions, tag the location entities that it reads within them, and then display those captions on a map for a user to interact with. The intended beneficiaries of such a project would be those involved with the digitisation task mentioned at the start of this section. An intended use would be to allow a digitisation professional to upload a file containing a series of natural language captions, each describing the location of a given specimen.

The application would then process this series of captions, tagging the location entities that it finds within them and displaying them on the map as well as any relevant data.

This offers two benefits. The first would be the quick and automated processing of a potentially repetitive and time-consuming task. The second would be the collection of specimen data in a central repository, with the opportunity of graphically displaying the locations of the specimen on a map, as opposed to just displaying textual descriptions of these locations. This graphical display may lead to connections being seen that possibly would not be noticed in the pure textual processing of this data.

It also presents an opportunity to assist in the current task of manual georeferencing, where a caption is too complex for the program to entirely pinpoint. The assistance would still greatly reduce the manpower needed, reducing the task from full georeferencing to simple validation.

The project will take the form of a web application, which will be able to be used as an interface to interact with the processing features. The scope of the project aims to include a graphical front-end map interface, the machine learning features of the back end, and a database to store processed data.

Extra scope includes the uploading and serving of files from user to server and vice versa, as well as potential geographical indexing of stored data.

The described project is effectively a general-purpose natural language processing and mapping website, and so in order to define clear and helpful functionality, this project has been assumed to be used for the purposes of the problem of collection digitisation.

In broad terms, this project is expected to generate a web application capable of processing and georeferencing natural language captions, as well as storing and processing data in an efficient and robust manner.

## 2 – Background

This section will provide some background on the area surrounding the project, as well as the project itself. It aims to provide familiarisation with the concepts and challenges related to this project.

### 2.1 - The Issue at Hand

Expanding on the original issue detailed in the Introduction section, this project is aimed at being used for the digitisation of historic collections. Where current specimen records are much more standardised and contain perfect latitude and longitude coordinates, older specimens tend to be described only by natural language references to location. While the scope of this project is limited (see section 2.5), it will attempt to use these natural language descriptions to infill a latitude and longitude (or multiple) for each specimen.

Below are three examples of descriptions that can be used to describe specimen locations:

- “Ngararatunua Cone, Kamo, Whangarei Co.”
- “Downlands at Gleniti, Timaru, on west side of Morgans Road, 300yd north from Timaru-Claremont Road”
- “Belmont Army Reserve, 40yds south of magazine 37”

The aim of this project is to create a system which can automatically process these natural captions and generate a given address for each of the specimen locations that it encounters.

### 2.2 - Stakeholders

Further to the section above, the specificity of this assumption of the project’s use allows us to identify key stakeholders in the use of such an application.

Clearly, this project would be most useful to those who are in possession of large historical collections, and who are interested in the digitisation and modernisation of their data.

Organisations matching that description include museums such as the National Museum of Wales, the National Museum of England, etc. The National Museum of Wales have already expressed an interest in a project of this type and have kindly provided this project with a dataset to allow us to build the project with example data in mind. Further to this, a New Zealand-based company, Landcare, have also provided a second dataset for the same purpose.

### 2.3 - An Introduction to NLP and NER

For this project, we will be using two concepts – Natural Language Processing (NLP) and Named Entity Recognition (NER).

These two concepts are integral to the function of this project, and they describe the solutions to task described in the Introduction section - NLP is a subdivision of machine learning, specifically the process of teaching a machine learning algorithm to process natural language, i.e., the language that this paper is written in. Natural language is not usually something that computers can process – human language can be vague, or subtle, or implicit, in ways that a computer cannot understand. Machines require structured, predictable, explicit language, features which unstructured natural language does not provide.

NLP aims to solve this problem – machine learning algorithms can use statistical models and language processing rules to guess the meaning of natural language, in order to bridge the communication gap between human and computer. NLP is used in many modern technologies, most

notably smart phone- and home-assistants such as Google’s Assistant (Singh, 2016) and Amazon’s Alexa (Gonfalonieri, 2018)

This project focuses on a further subdivision of NLP known as Named Entity Recognition (NER). NER can be defined as “the task of identifying and categorizing key information (entities) in text” (Marshall, 2019). A statistical model classifier is used to determine the nature of an entity based on the context of the caption around it. The classifier is expected to resolve ambiguity issues, for example the entity “India”. India is a geopolitical entity, but also a common name for a human. A good NER tool will be able to distinguish which of these natures is implied, based on the context that it reads.

## 2.4 – Statistical NLP Models

Another key concept to the functioning of this project is the use of a statistical model known as an NLP model. As NLP belongs to the domain of artificial intelligence and machine learning, it follows that the NLP algorithm needs to have a “knowledge base” to induce information and make decisions on unseen data.

For NLP, this knowledge base is known as a statistical model and describes, statistically, the rules and knowledge that the NLP algorithm uses to decide whether a candidate entity is correct (in its opinion), or whether it is a false positive.

These statistical models can vary wildly in size and efficiency, depending on the data that they were trained on, and different statistical models are suitable for different tasks.

Many NLP packages include inbuilt statistical models for “out-of-the-box” use, and a few of them have functionality for training new statistical models based on the inbuilt statistical models.

## 2.5 - Geo-Geo- and Non-Geo-Geo-Ambiguity

The “India” issue mentioned in section 2.3 is a well-known problem in the study of georeferencing. It is referred to as Non-geo-geo-ambiguity, in which a geographical entity can be confused with a non-geographical entity.

The sister issue to this is also an issue that will have to be addressed in this project. Known as geo-geo-ambiguity, it describes the issue of disambiguating a place name that occurs multiple times in a given space. For example: there are 15 instances of a town or city in the world which are named “Raleigh”. For a human, it is usually easy to disambiguate these occurrences automatically and subconsciously, based on context, but this presents a larger issue for a computer.

## 2.6 - Relative References and Project Constraints

Natural language can, at times, be vague. There are a multitude of ways to express an idea, and this case remains true when describing the location of a subject. “3 miles west of Swindon”, “Just at the end of the road”, and “the intersection of Shand and Trent Roads” are all examples of relative relations, in which the location of the subject is described in relation to the location of another, usually more well-known object. There is an incredibly large number of ways to express these so-called relative references, such that manually coding the recognition of each of these expressions is a futile task.

This issue presents a large constraint, and solving it is beyond the scope of what can be expected for this project. In that regard, we will be ignoring relative references and instead simply tagging entities that are mentioned in a caption. For example, in the example of “3 miles west of Swindon”, the tag

will be placed directly on Swindon, as opposed to 3 miles west of Swindon. While this solution is less than perfect, and will lead to a loss of accuracy, it is a good compromise to the issue.

## 2.7 – Evaluative Measures

In order to evaluate the relative performance of a given NLP tool against a dataset, a value is needed. The F1 measure is introduced, which is defined as the harmonic mean of the precision and recall measures of a dataset:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

Where precision and recall are each defined as a ratio of true positives to true positives and false positives or true positives and false negatives, respectively:

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN}$$

Where TP, FP, and FN stand for True Positives, False Positives and False Negatives, respectively.

In short, an F1 score approaching 1 is considered better performance. It is desirable to maximise the F1 score.

The application of these equations with respect to the data used for this project will be expanded upon in section 6.1

## 2.8 - Existing Solutions

The idea of automated georeferencing, especially in the field of collection digitisation, is not an entirely new concept. (Van Erp et al., 2015) attempted a reasoning-based approach to collection digitisation for the Netherlands Centre for Biodiversity Naturalis.

A 2004 paper (Murphey et Al, 2004) was published in order to evaluate four pre-existing automated georeferencing tools:

- Biogeomancer (Beaman and Conn, 2003)
- MANIS Georeferencing Calculator
- GEOLocate
- ArcView Georeferencing Extension

The above-mentioned articles are certainly not an exhaustive list of attempts to solve this issue, and other evaluative papers list more automated georeferencing tools (Melo and Martins, 2017).

It is clear, from the above, that the study of automated georeferencing is not an unsaturated field. It is worth noting, however, that the machine learning techniques used to automate georeferencing improve year upon year at an abnormal rate. With that in mind, it is felt that this project is justified in using modern techniques to improve upon the base performance of the georeferencing tools from the early 2000s.

On a wider scale, there exists further solutions to georeferencing. The examples listed above all aim to tackle the issue of georeferencing for collection digitisation, but this is only one facet of the many issues to which automated georeferencing can be applied.

Notable systems for automated geoparsing outside of the sphere of digitisation of collections include:





## 3 - Justification of Tools

This project will use the Python programming language as a server-side language. Python includes multiple libraries for NLP and NER, as well as other tools that will be necessary for the efficient implementation of this project. This section will present the range of tools to choose from and discuss the choice between them.

### 3.1 - NER Tools

There is a myriad of available NER tools online (Shen, 2019; Roldós, 2020), and so a choice needs to be made as to the choice of tool to use.

The first thing to note about NER tools is the difference between the old-style “rule-based” classifiers and the newer deep-learning algorithms that are used for NER. Today, most of the NER tools use this deep learning, but it is worth mentioning that this project will be disregarding any of the old rule-based classifiers, as a deep learning approach provides many advantages over rule-based approaches, including better scalability (Smith, 2020) and the ability to train them.

This project is, at its base, a web application. To that extent, it is a set requirement that it must be somewhat fast at processing data. The issue is that the properties of “fast” and “accurate” in the domain of NER are somewhat mutually exclusive. The more accurate tools are, on average, slower. An example of this can be seen in (Dishmon, [no date])’s comparison of NLTK and Stanford NER, where he concludes that “It seems Stanford is more accurate, but NLTK is faster”.

With that in mind, Stanford will be ruled out as an option. Other viable options are now:

- NLTK
- SpaCy
- Flair

These NER tools are the forefront tools in the domain. The author of this paper has used NLTK before and found it to be difficult to use and not particularly well documented. This project will instead consider SpaCy versus Flair.

The trade-off between the two tools is summed well by (Duffy, 2020). In short, both Flair and SpaCy have good accuracy. Flair uses slightly more “state-of-the-art” technology than SpaCy, with the trade-off that “[Flair] is known to be slow”. Duffy goes on to point out, however, that Flair can be optimised to the point where “[the] inference time can be divided by 10”

The table below was taken from this article, and sums the pros and cons of both tools:

|              | Pros   | Cons  |
|--------------|--|---|
| <b>Flair</b> | <ul style="list-style-type: none"> <li>• Open source library designed to reach the state of the art in NER</li> <li>• Flair supports a number of languages – and is always looking to add new ones</li> <li>• Comprises of popular and state-of-the-art word embeddings</li> <li>• The developers are additionally currently working on “Frame Detection” using flair</li> <li>• Great modular design</li> </ul> | <ul style="list-style-type: none"> <li>• Known to be slow</li> <li>• Fairly new and <u>alot</u> of work still needs done to improve it</li> </ul>   |
| <b>SpaCy</b> | <ul style="list-style-type: none"> <li>• Well-engineered and documented.</li> <li>• Known as the fastest NLP framework.</li> <li>• Easy to learn and use because it has one single highly optimised tool for each task.</li> <li>• Provides built-in word vectors.</li> <li>• The support is active and the development is ongoing.</li> </ul>   | <ul style="list-style-type: none"> <li>• Accuracy was too limited.</li> <li>• Doesn't support many languages, There are models only for 7 languages and “multi-language models</li> </ul> |

Figure 3 - A Comparison of Flair and SpaCy

The choice, then, falls to the above. As mentioned earlier, SpaCy trades accuracy for speed, and misses out on ticking the “state-of-the-art” checkbox. It is the author’s opinion that prioritising speed will be useful for this project, and the accuracy trade-off can be mitigated by training the SpaCy model on the input data that was provided by the mentioned stakeholders. Another benefit of SpaCy is that it is well documented, which will be helpful towards the smooth implementation of this project. The other negative of SpaCy is that it lacks language support, however the data that will be used will be in English only, and so that does not present an issue to the performance of the tool. Conversely, Flair’s relative “new-ness” and relative slowness are two issues that would hamper the performance of the tool.

There is, however, a third option to consider. While SpaCy and Flair are both valid options, another tool, called Stanza, needs to be considered. Stanza provides a Python wrapper for the Stanford NER tool mentioned earlier. The official documentation for SpaCy (SpaCy, [no date]) provides a table of F1 scores for these 3 tools, tested against 2 benchmark corpora:

| NAMED ENTITY RECOGNITION SYSTEM   | ONTONOTES | CONLL '03 |
|-----------------------------------|-----------|-----------|
| spaCy RoBERTa (2020)              | 89.7      | 91.6      |
| Stanza (StanfordNLP) <sup>1</sup> | 88.8      | 92.1      |
| Flair <sup>2</sup>                | 89.7      | 93.1      |

Figure 4 - F1 Scores for SpaCy, Stanza, and Flair

The F1 between them is almost indistinguishable, with Flair only beating the other two tools against the CONLL 2003 corpus. In consideration of the above table, the choice now comes down to Stanza vs SpaCy.

This project will still use SpaCy, due to the fact that its documentation is superior to Stanza, and the new RoBERTa en\_core\_web\_trf model included with SpaCy 3.0.0 has a very strong F1 value of 89.7-91.6 (as seen in Figure 4)

Not only does SpaCy provide inbuilt statistical models, but it also gives the option of self-training a new statistical model (using other models as a basis), using self-annotated data. This option will be explored later in this project.

### 3.2 - Geocoding Tools and API Choice

When it comes to geocoding in Python, the de facto standard is the “GeoPy” library. This library presents an interface for easy connection to multiple geocoding APIs online, including the OpenStreetMap API group.

Some research was conducted to investigate if there exist any alternate libraries for connecting to these APIs. There is another library, called “geocoding” which seems to present the same functionality as GeoPy does. To reinforce this statement, a page was found which compares the functionality of GeoPy to Geocoder (WebGeoDataVore, 2015). The page concludes that, while “geocoding” adds some extra functionality focused on command line interfacing and IP geocoding, the two libraries are effectively the same, and thus neither presents that much of an advantage over the other. With that in mind, this project will use the GeoPy library, as it is a library that the author is familiar with.

Given that, the only other decision to make is between which geocoding API will be used for the project.

As of the time of development of this project, GeoPy supports a total of 23 different APIs. As above, there exists a de facto standard for this, which can be found in the Nominatim API, which is provided by OpenStreetMaps.

The obvious benefit of Nominatim is that it is free, which means it will be what is implemented for this project.

The issue arises, however, that Nominatim is run on donated servers (OSMFoundation, [No Date]) and, as such, cannot handle a large request rate. In that regard, usage of Nominatim is rate limited to 1 request per second which, while acceptable for small datasets, will not be suitable for the large datasets that this project would eventually be used upon. Currently, this presents only a small issue. Datasets can simply be truncated to be small. This would, however, present an issue to further work on the project.

An alternative to Nominatim is the Open MapQuest API, which is also a part of the OSM group. It is effectively identical to Nominatim, with a few changes. The positive is that there is no rate limit to OMQ, which would solve the above issue that Nominatim presents. The negative is that OMQ is transaction limited by a pricing structure, and to accommodate the number of transactions that would be expected from a file such as the data this project is aimed at (320,000 captions in the largest file) would cost upwards of \$900 USD / month.

Were this project to become funded, and be able to afford the use of OMQ, multithreading could be used to drastically improve the performance of the program and send multiple requests to the API at a time.

Having checked each of the APIs offered by GeoPy, Nominatim can be concluded to be the only viable choice currently, without moving to a paid option.

There is an API, named Pelias, which would have offered a QPS (Queries per second) of 6 and a QPD (Queries per day) of 30,000, which presents a balance between Nominatim and OMQ, but the online hosting for Pelias was provided by the MapZen project, which was shut down in 2018.

With all of the above in mind, this project will proceed using the GeoPy library connecting to the Nominatim API.

### 3.3 – DBMS

When it comes to choosing a database management system for an application, there are many choices to make and many DBMSs to choose from.

The first choice to make is whether to choose a DBMS from the SQL family or the NoSQL family. The SQL family uses the rigid, schema-bound structure, using SQL for database queries. DBMSs in this family include MySQL, Oracle, and PostgreSQL. The use of SQL emphasises the ACID rules of database transactions – atomicity, consistency, isolation, and durability. Atomicity guarantees that a transaction will either completely succeed or completely fail. Consistency guarantees that all data within the database adheres to the rules defined by the database itself. Isolation guarantees that the database can concurrently process multiple transactions and the outcome of them will not affect each other. Finally, durability guarantees that, in the event of a system failure, all saved data will be preserved and protected.

As described above, the SQL family requires schema-based databases, which requires that all input tuples include the exact same data every time.

While this is suitable for some applications, the data that this project will be processing will be coming from different sources. To that extent, the input tuples will not contain the same information between separate data sets. This can be seen below:

|         |               |        |               |            |      |                      |          |          |         |          |           |          |            |
|---------|---------------|--------|---------------|------------|------|----------------------|----------|----------|---------|----------|-----------|----------|------------|
| site_id | Date observed | Region | Map series    | Map number | East | North                | Locality | Altitude | Element | Landform | Landscape | Land use | Site Notes |
| RecID   |               |        | catalogNumber |            |      | SpecimenDataLocality |          |          |         |          |           |          |            |

Figure 5 - Comparison of the Landcare and NMW dataset schemas

The two images above show the schema for two separate datasets. As can be seen, they contain different fields (i.e., represent different schemas), and so would be unsuitable for storage within an SQL database.

A database which is “schema-free”, then, is needed. This is provided by the NoSQL family of DBMSs. The NoSQL movement emphasises the idea of not being bound to SQL and having no schema. To that extent, anything can be inserted to a NoSQL database, no matter the format of the tuple.

Within NoSQL exists multiple types of DBMS. There are document-based databases, graph-based databases, key-value databases. The document family of NoSQL databases suits the needs of this project. Document databases store data tuples as “documents”, which are schema free and can store any data. Multiple documents make up collections, and multiple collections make up a database. The data that this project will be storing suits the idea of a document, as a point on the map will simply be stored as a document consisting of a caption, an address, a latitude-longitude pair, etc. The NoSQL design will also allow for extra data to be stored with each document, if necessary.

Within the document family exist many choices. The most widely deployed, however, is MongoDB (Yegulalp, 2018). MongoDB presents multiple advantages, such as very good documentation, the ability to deploy free non-local clusters, a good interface, etc.

One of the main benefits of MongoDB to this project is that it also has support for use with Python, via a library called PyMongo. Using this library will mean that DB functionality can be accessed directly from the back end, with no complicated API calls.

This offer of simplicity, combined with the ability to deploy an online cluster with metric recording and easy database management, makes MongoDB an easy choice for this project.

### 3.4 – Web Framework and Python

The goal is to deploy this application as a web application. To that extent, there will be an intermingling of languages used to realise the front and back ends of the project. The front end will be written in HTML, CSS and JavaScript, and the back-end functionality will be written in Python.

As a side note, Python has been chosen as it can be considered the leading language for AI based applications (Zola, 2018).

To build a web application in Python is becoming increasingly easy. There are several frameworks in Python that are designed to help developers easily and efficiently develop code for secure, efficient web applications.

Frameworks offer baked-in functionality for routing, serving, and managing web applications, and the obvious choice for this application is to use one of these.

It has been decided that the application will use a framework called Flask, which has been chosen due to the author's familiarity with it. The author found its documentation to be clear and concise. It is also popular with developers, which means many tutorials for Flask can be found online. Some research into the advantages of Flask found information agreeing with the above points, as well as making the points that Flask is extensible, fast, easy to deploy, secure, and flexible (Holcombe, 2020).

Flask provides a way to link between the front-end HTML script and the back-end Python code of the program. It includes something known as a templating engine, called Jinja. Jinja, among many other functionalities, allows Python-style code to be written in the HTML documents, which vastly increases the effectiveness of the HTML side of the project.

## 4 - Specification and Design

This section will define the features and requirements that the project is expected to incorporate, as well as provide a detailed view into the design and data flow of the application. It is, for the most part, a plan for the development of the project but has in some areas been retroactively edited to contain comparisons between the plans and the final design of the project.

### 4.1 - Features

The project will incorporate the following features:

#### 4.1.1 – Single Geocoding Capability and Biasing

The application takes a natural language caption, processes it to tag the geo-entities within it, finds a likely address for each entity and then returns:

- From backend:
  - A JSON object containing the caption and the entities found within it, including raw address and latitudes and longitudes for each entity
- From frontend:
  - A marker or marker chain on the map with popups for each marker containing the caption and entity address, with related entities (from the same caption) linked together by a line

The functionality also supports biasing results towards a given country, allowing for some inter-country geo-geo-ambiguity issues to be resolved

#### 4.1.2 – Database Integration

The application is linked to an external database, in which it stores previously geocoded captions.

The application allows the user to commit all current points from the session to the database, as well as fetching a set number of captions from the database.

#### 4.1.3 – Search Capabilities

The application allows the user to search the database of pre-processed captions either by address, or by an interactive graphical-based radial search.

#### 4.1.4 – Bulk Geocoding with File Uploads

The application supports the user uploading a csv file and will bulk process the file, allowing for efficient use of the features of the program

It supports optional columns which can be fed to the biasing feature of the geocoding functionality

#### 4.1.5 – Exporting

The system allows the user to export the current session, formatted to a CSV file, for use in other programs

## 4.2 – User Interface

The user interface is designed to be simple, and to facilitate all major functionality within around 3 clicks from the main screen. The website comprises of a single page, with functionality being presented from hide-show toggle forms that can be accessed from an expandable navigation bar at the top of the page.

The reason for this design is to prioritise minimalism, and to present an unbroken experience to the user. This is achieved by never taking the user away from the map page, and instead simply summoning small forms to cover a section of the page when necessary. This design facilitates a simple experience for the user, allowing the website to be used by anyone.

The below figures illustrate the wireframes that were created at the start of this project, as a rough idea for the UI. These are followed by their equivalents in the final system.

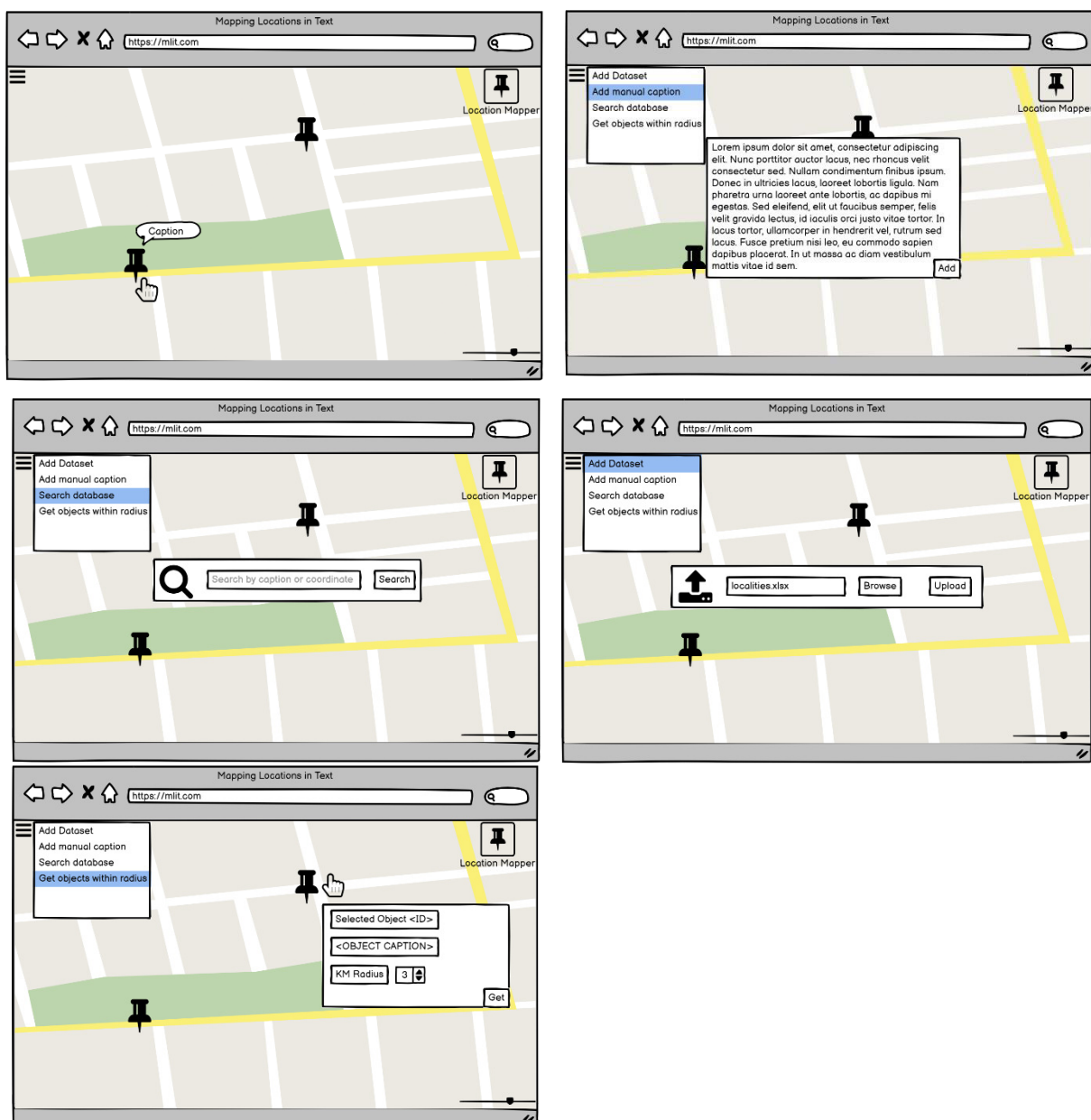


Figure 6 - Initial UI Wireframes



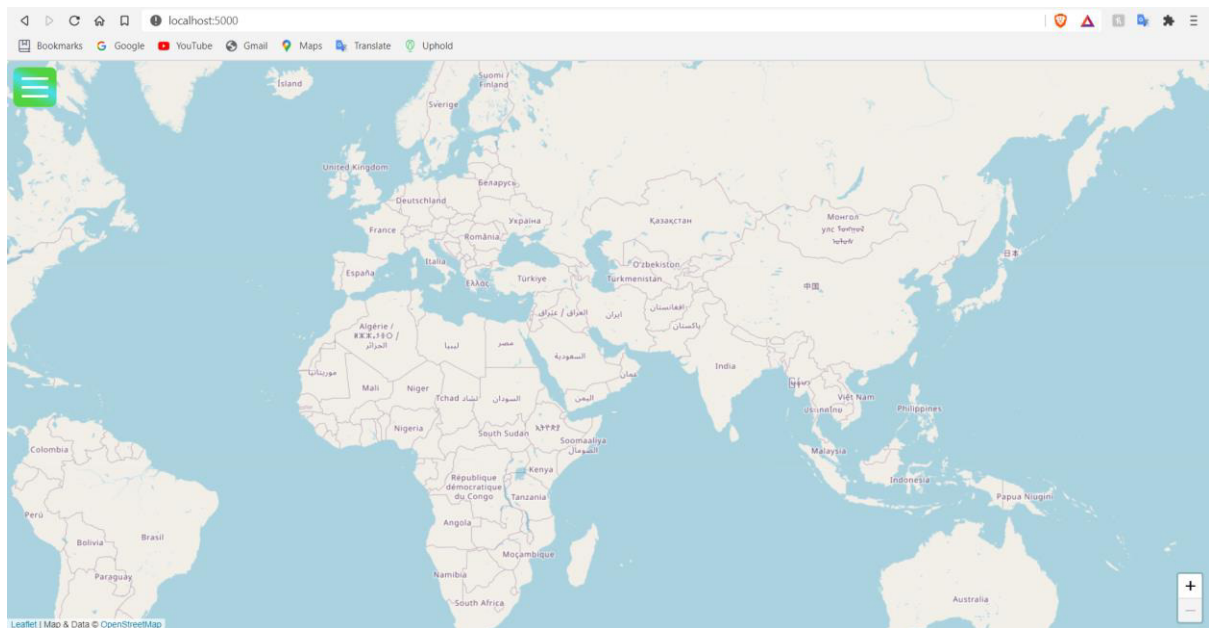


Figure 7 - Main Page Wireframe Equivalent

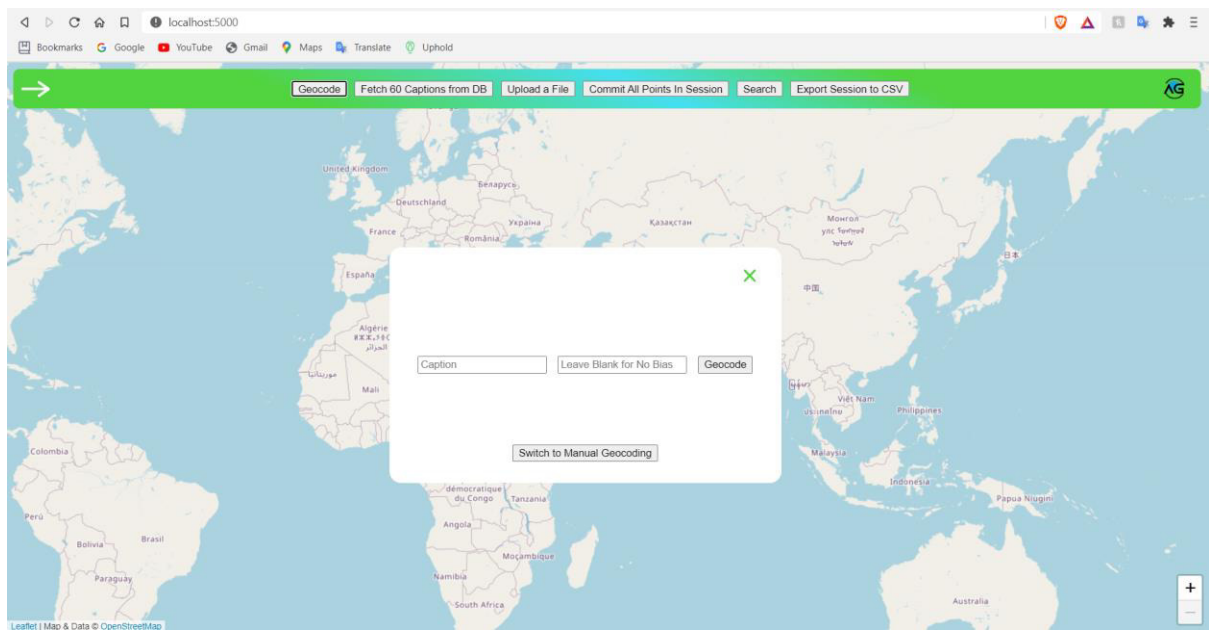


Figure 8 - Geocoding Wireframe Equivalent



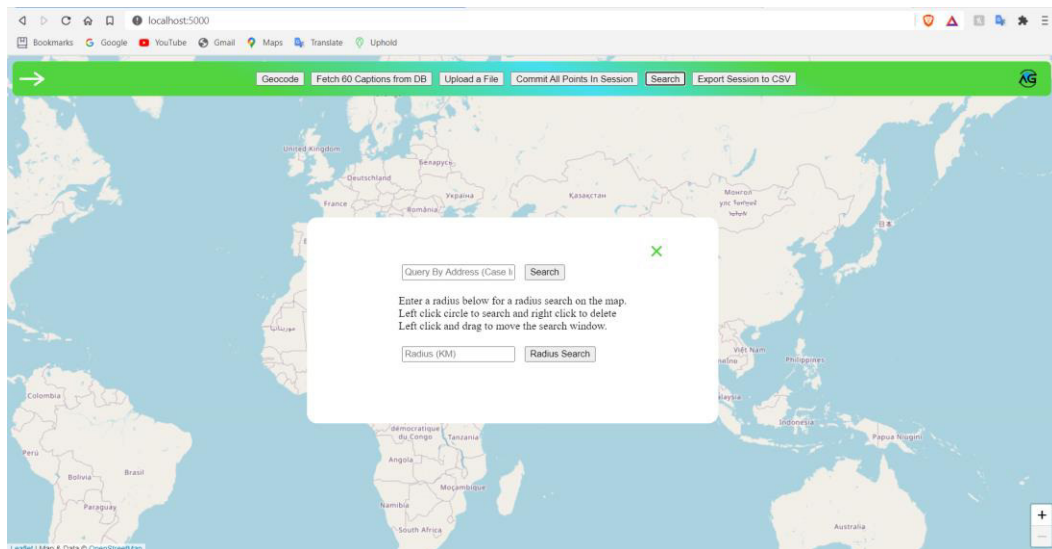


Figure 9 - Search Wireframe Equivalent

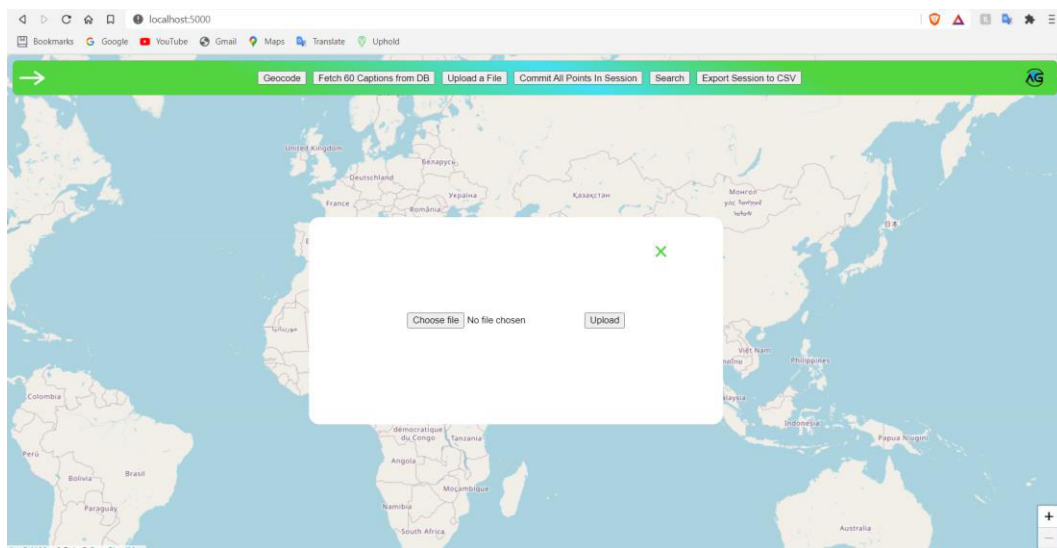


Figure 10 - File Upload Wireframe Equivalent

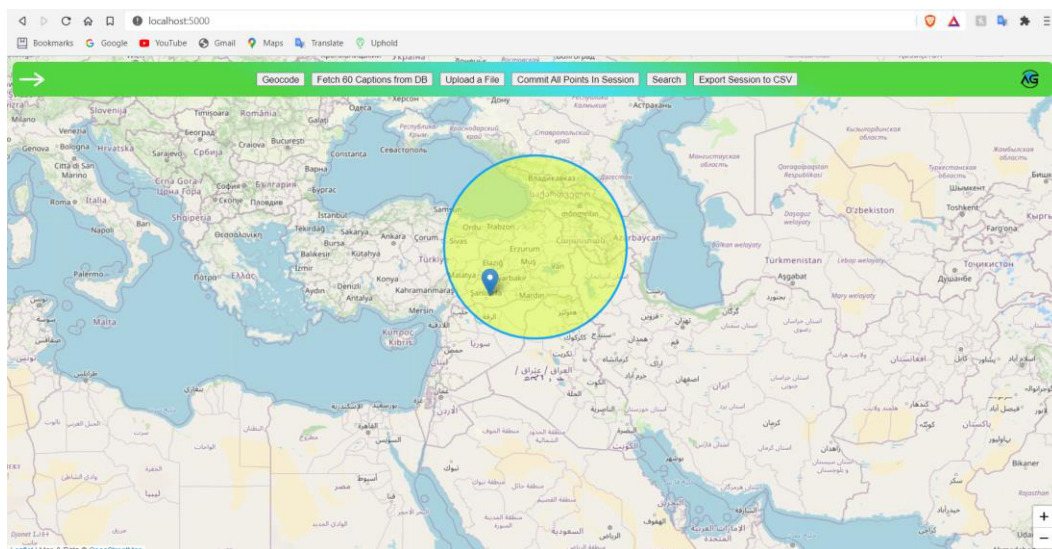


Figure 11 - Radial Search Wireframe Equivalent

### 4.3 – Data Flow

Using the Yourdon and Coad style for data flow diagrams:

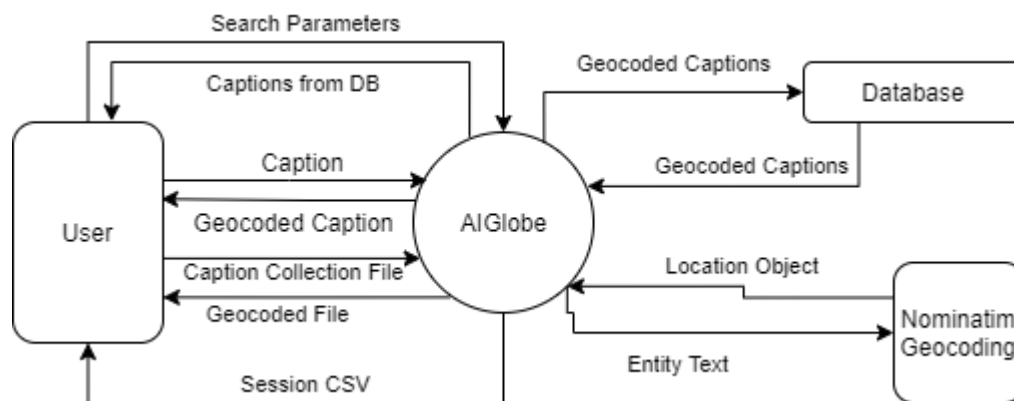


Figure 12 - Level 0 DFD (Context Diagram)

In a high-level view, the user sends a caption to the application, which processes it internally (expanded on in the Level 1 diagram), sends the entity text to Nominatim, reassembles the geocoded caption, and returns it to the user.

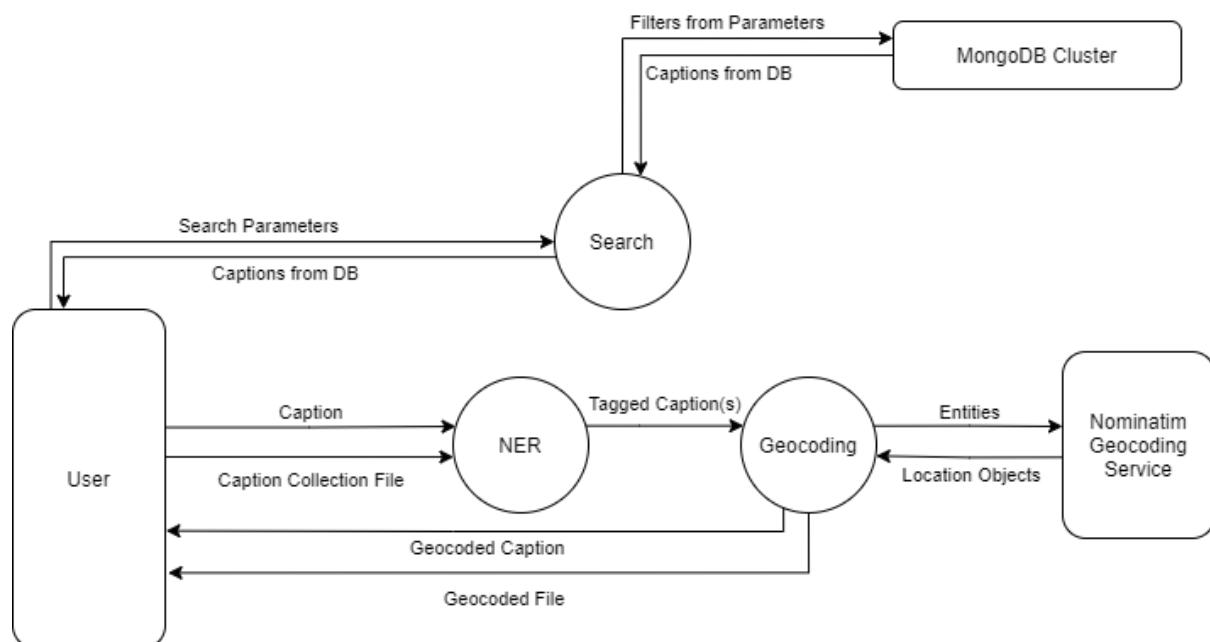


Figure 13 - Level 1 DFD

## 4.4 – System Algorithms

### 4.4.1 – Vincenty Distance

For the purpose of the radial search functionality described in section 4.1.3, the system needs to calculate the distance between two points. In a flat plane, this task is easy. It is, however, slightly more complex in the 3D plane that the world occupies. There are multiple algorithms that can be used to calculate distance, including Haversine, Great Circle, and Vincenty distance. Vincenty is the most accurate of these, with the trade-off of being the most complex.

This complexity is, however, negligible when it comes to calculation, even on large sets of data (Matan, 2017). It is for the high accuracy, then, that I have chosen the Vincenty Distance algorithm to implement.

## 4.5 – Static Architecture of the System

The system is designed to be modular. Events occur in the system by way of the client sending API requests in the form of URLs to the server. A map of the relevant server modules can be seen below:

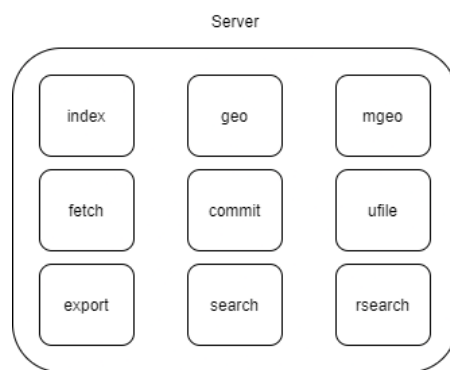


Figure 14 - Server Module Map

The functions of each module are described below:

- index
  - Serves homepage
- geo
  - Handles geocoding functionality
- mgeo
  - Handles manual geocoding functionality
- fetch
  - Fetches data from MongoDB cluster
- commit
  - Commits data to MongoDB cluster
- ufile
  - Handles file uploads
- export
  - Handles exporting session to csv and serving to user
- search
  - Handles flat search
- rsearch
  - Handles radial search

Data is returned to the client using JSON notation, with a standardised structure following the {code, message} object template.

Each module in the server is decorated with a specific URL route (as well as inputs) – for example:

- /geo/<caption>/<bias>
  - Triggers the geocoding event in the server, returns the geocoded caption

```
127.0.0.1 - - [23/Apr/2021 11:32:27] "GET /geo/Test%20Caption/None HTTP/1.1" 200 -
```

Figure 15 - /geo Event

- /fetch
  - Triggers the DB fetch functionality, returns fetched captions from database

```
127.0.0.1 - - [23/Apr/2021 11:33:01] "GET /fetch HTTP/1.1" 200 -
```

Figure 16 - /fetch Event

The below figure shows the raw JSON return data from the geo event:

```
{
  "code": 200,
  "message": {
    "caption": "Okaihau, Northland",
    "entities": [
      {
        "address": "Okaihau, Kaikohe-Hokianga Community, Far North District, Northland, New Zealand",
        "entity": "okaihau northland",
        "lat": -35.4012707,
        "long": 173.2565633
      }
    ]
  }
}
```

Figure 17 - Raw Return Data

An exhaustive diagram of all possible server interactions can be seen below:

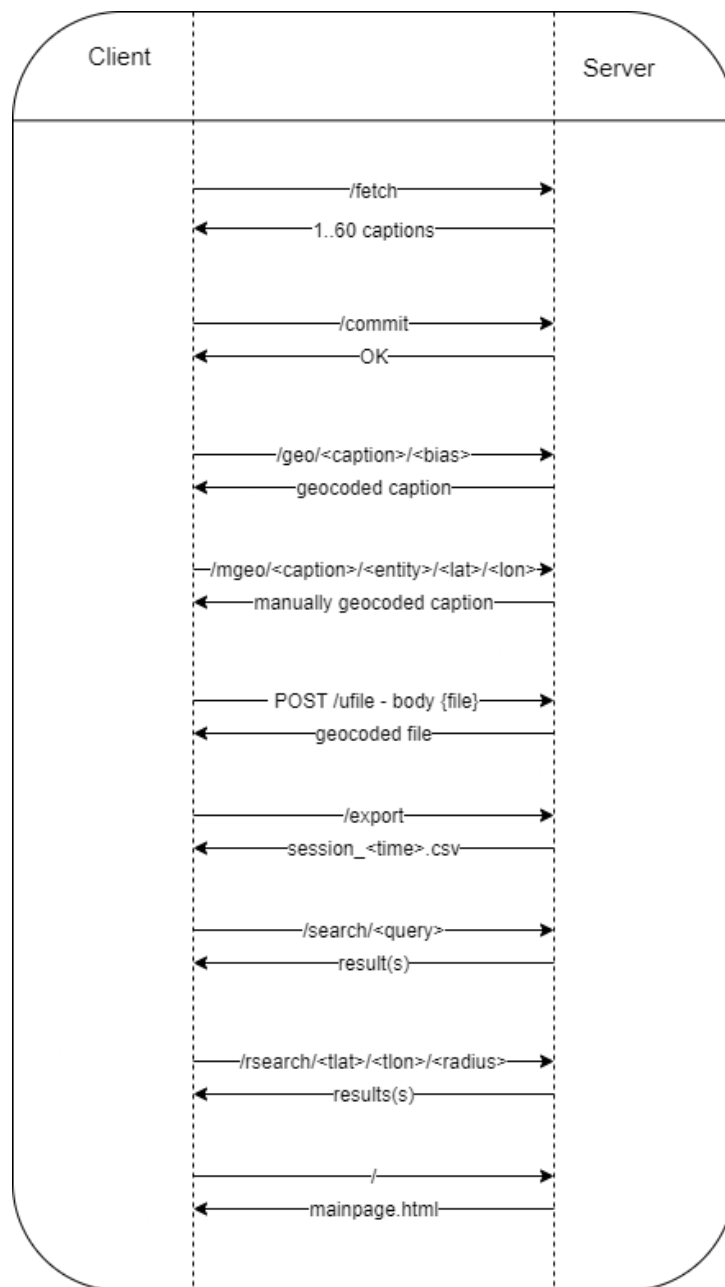


Figure 18 - All Possible Client-Server Interactions

All interactions, with the exception of the “/ufile” route, use the GET method.

The modular design offers several advantages. Firstly, events on the server can be triggered by the client, which facilitates two-way communication between the client and server, as opposed to just the server sending data once to the client.

Secondly, the front-end (client) and back-end (server) are now effectively separate. While the client code facilitates the use of these URL routes, and provides an interface for use, the back-end can be theoretically used from within another external program by sending API requests to the server. With proper documentation, this could be used by stakeholders who wish to incorporate the geocoding functionality of this project within their own applications.

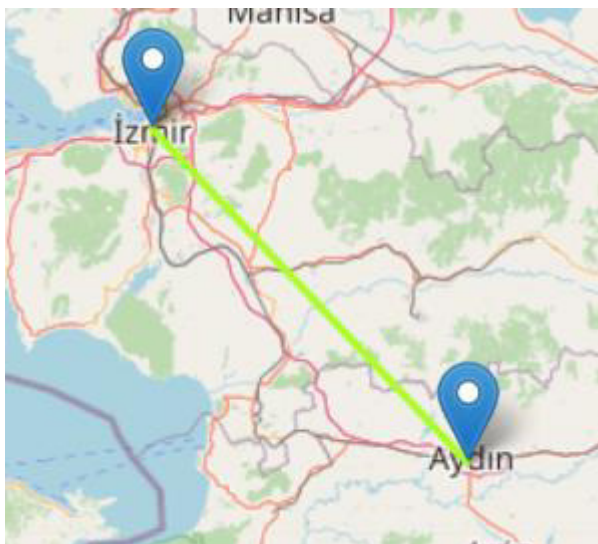
The design also prioritises simplicity and allows for modular expansion of the functionality of the application by providing a set way to add new features. A developer would simply need to add a new route, which could then be incorporated into the front-end. It facilitates easy development on this project.

#### 4.6 – Small Features

Various small features have been added in order to improve the aesthetic value of the client-facing front-end. This section will briefly explain them.

##### 4.6.1 – Caption Chaining

It is common for a caption to contain two or more entities. When these entities are returned to the user and tagged on the map, it could be possible for the user to lose track of which entities belong to which captions (without manually clicking on each pin on the map to check which caption to which the entity belongs). In order to alleviate this issue, a chaining system has been implemented, in which a line (with a random green shade) is drawn between all entities in a caption. An example of this can be seen below:



*Figure 19 - Caption Chaining for Multiple-Entity-Captions*

#### 4.6.2 – Header Minimisation

In order to present as minimal an interface as possible, the header (containing all interaction operations) has the option to be minimised, leaving just a small box in the corner of the page. This can be seen below:

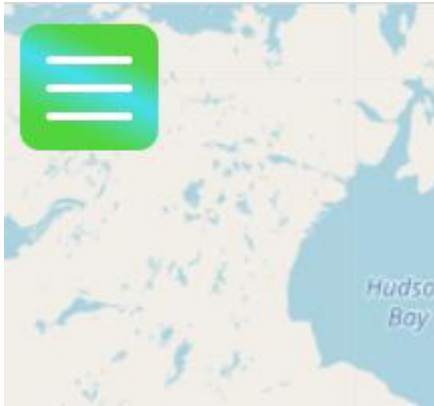


Figure 20 - Minimised Header

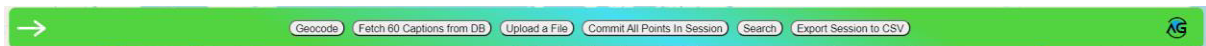


Figure 21 - Maximised Header

The arrow on the left is also the product of the animated burger menu icon.

## 5 – Implementation

This section will detail the implementation of the project on a code-level scale. It aims to introduce and explain the important code behind the main features of the project.

### 5.1 – Geocoding and NER

#### 5.1.1 – NER

The system uses a custom NLP model (explained further in section 5.6) to tokenise, parse, and tag entities within a given natural language caption.

Due to the nature of Python, callouts to external libraries are mostly simple. In order to facilitate natural language processing, the rather complex task of NLP can be simplified to a few function calls within the code:

```
nlp = spacy.load("./static/models/en_aiglobe")
```

Figure 22 - Loading the Model

Firstly, an object, called `nlp`, is assigned using the `spacy.load` function with the location of the custom NLP model as a parameter.

```
doc = nlp(caption)
```

Figure 23 - NLP Function Call

This function call then passes the caption through the “pipeline” that the NLP model implements. In the custom model that this program uses, any non-essential pipeline components have been disabled. The caption is simply tokenised, vectorised, and then tagged using the NER component of the pipeline.

An example of a tagged caption can be seen below, served for graphical inspection by SpaCy’s DisplaCy module:

Kawakawa Bay **LOC** , NW shore of Lake Taupo **LOC** , east road of Otutira Catchment **FAC** , 600 m north of lake

Figure 24 - Example of a Tagged Caption

#### 5.1.2 – Forward Geocoding

The geocoding section of the project is handled by external API calls to the Nominatim geocoding service, run by OpenStreetMaps.

The API calls are, themselves, handled by the GeoPy python library.

```
gc = RateLimiter(Nominatim(user_agent='app')).geocode, min_delay_seconds=1)
```

Figure 25 - Initialising the Geocoder

The line above shows the initialisation of a Nominatim object, which is then wrapped in a second Rate Limiter object which forces consecutive requests to adhere to a 1 second delay, so as to adhere to Nominatim’s strict 1 request per second quota.



```

address = gc(ent.text, country_codes=bias, language="en")
if address:
    loc = {
        #Construct location object
        #Convert entity to lowercase and strip punctuation
        "entity": ent.text.lower().strip().translate(str.maketrans('', '', string.punctuation)),
        "address": address.address,
        "lat": address.latitude,
        "long": address.longitude}
    points.append(loc)

```

*Figure 26 - Geocoding and Structuring the Return Data*

The above lines first send a request to Nominatim with the entity text tagged in the NER section, as well as a country bias (Seen in section 5.1.3) and forces the return language to be English so as to standardise input and output data.

A JSON object is then constructed containing the entity, its located address, and its latitude and longitude. This JSON object is the standard representation of a location for this program.

### 5.1.3 – Country Biasing

In order to resolve some higher-level geo-geo ambiguity issues (between countries), the user is given the option to bias the returned data, if they are sure that a given caption's footprint is within a country.

```

bias = pycountry.countries.search_fuzzy(bias)[0].alpha_2

```

*Figure 27 - Fuzzy Search for the Bias Country*

The contents of the bias input box on the front end are handed to the backend in the GET request, and the bias is then fed to the line above. The line uses a python module called pycountry to link the input of the box to an ISO alpha 2 code for the country, which is the input that the Nominatim URL requests in order to bias data. Note that the search is fuzzy, and so "Deutschland" and "Germany" will result in the same ISO code.

## 5.2 – DB Fetch and Commit

A MongoDB cluster has been deployed in order to provide a basic data store for the application. This cluster is accessed using the PyMongo library for python, which provides native python functions for database interactions.

### 5.2.1 – Fetch

The program fetches a limit of 60 captions from the database when the /fetch request is sent to the server.

```
global session_points
#Pull 60 points from db
points = list(db.find({}, {"_id": 0}, limit=60))
#Scrub any points that are already on the map
points = [point for point in points if point not in session_points]
[session_points.append(point) for point in points]
return jsonify({
    "code": 200,
    "message": points
})
```

Figure 28 - Fetching 60 Captions from the DB

Figure 20 shows (up to) 60 points being fetched from the database. So as to avoid data replication and unnecessary data transmission, any captions that are already on the map are scrubbed from the return data.

### 5.2.2 – Commit

```
ninserted = 0
if len(session_points) == 0:
    return jsonify({
        "code": 500,
        "message": "No Points To Commit"})

for point in session_points:
    #Unique assertion on points in DB
    #Skip any duplicates
    try:
        db.insert(point)
        ninserted += 1
    except DuplicateKeyError:
        continue
if ninserted == 0:
    return jsonify({
        "code": 502,
        "message": "DB Commit Unsuccessful"})
else:
    return jsonify({
        "code": 200,
        "message": "DB Commit Successful"})
```

Figure 29 - Committing the Session to DB

To avoid duplicate captions being stored in the database, a unique key has been placed on the captions. As can be seen in the code above, the system will try to insert a caption, which will throw a “Duplicate Key Error” if it is already present in the database. In that case, the system will skip to the next caption.

## 5.3 – Search

### 5.3.1 – Standard Search

```
flat_search = list(db.find({ "entities.address": {'$regex': '^.*'+query+'.*$', '$options': 'i'}}, {"_id": 0}))
```

Figure 30 - Flat Searching

The above line sends a fetch request to the database, filtered on the address value of each entity in a caption. The regex searches case-insensitively for any occurrence of the query in any of the addresses. If a match is found, the entire caption is returned.

### 5.3.2 – Radial Search

The radial search is slightly more complex than the standard search functionality.

On the front end of the application, the user is presented with an input box, to enter a radius (in KM) for the search. Once a radius is entered and the radial search button is clicked, a circle of the entered radius is spawned on top of the map. This circle can be dragged around. A 500km radius circle can be seen in the example below:

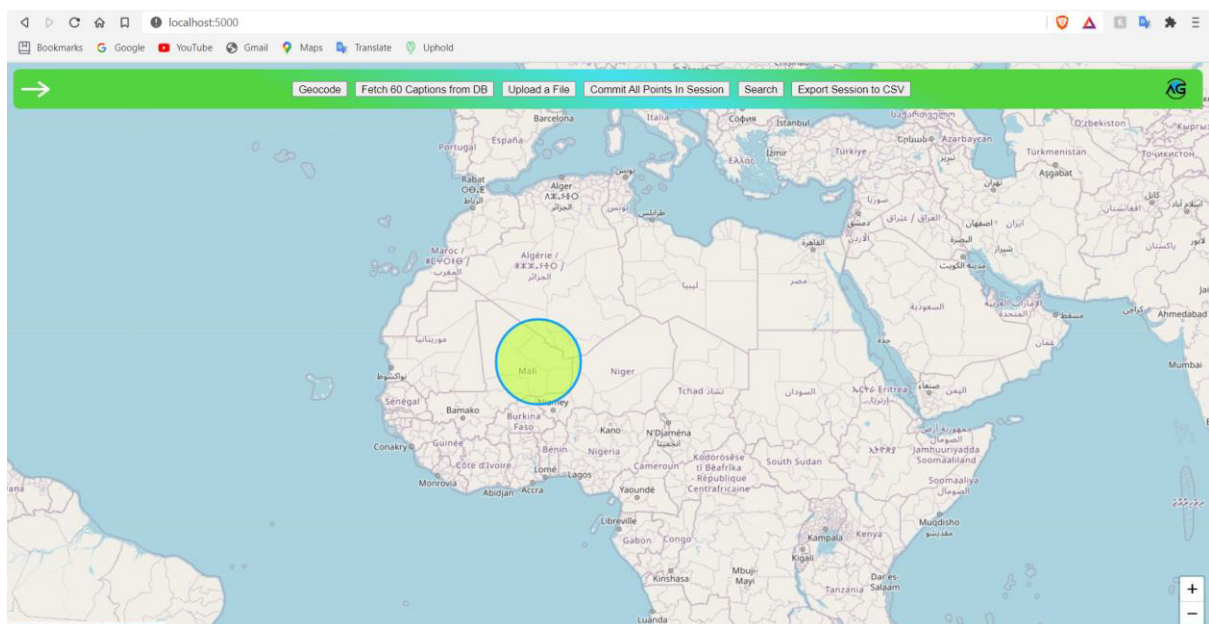


Figure 31 - A 500km Radial Search Window

A well-known issue in geodesy and other Earth-related fields is the projection of a spherical object (the Earth) onto a 2D plane (a Map). The most popular solution to this is the Mercator Projection method (Singh, 2017), which stretches the poles of the Earth. As a result of this, 500KM on the Mercator Projection is larger nearer the poles than it is at the equator.

Leaflet, the mapping extension that has been implemented, can deal with this inconsistency, and will stretch the search circle to ensure that the 500KM radius is preserved, no matter the latitude of the search window.

The figure below demonstrates this:



Figure 32 - 500KM at different latitudes

Once the user has the search window positioned over the location they wish to search, the front end gathers the radius and centre latitude and longitude of the circle and passes them via URL to the back-end.

```
#Pull all from db
rad_search = list(db.find({}, {"_id": 0}))
for point in rad_search:
    #Check all entities in pt
    for entity in point["entities"]:
        search = (entity["lat"], entity["long"])
        target = (float(tlat), float(tlong))
        #Using Vincenty Distance
        if distance.distance(search, target).km <= float(radius):
            #If point contains more than 1 entity and only 1 entity is within radius, point will still be returned
            found.append(point)
            break
```

Figure 33 - Radial Searching Algorithm

The system fetches all of the known captions from the database and checks each entity in each caption. Using the Vincenty Distance algorithm mentioned in section 4.4.1, each entity is checked to see if it is within the radius of the centre of the search window. If it is, the system passes the entire caption back to the user.

## 5.4 – File Uploads

The system, as well as being able to process single captions at a time, can also be handed an entire file to process.

This API route uses the POST method instead of the GET method in order to handle an entire file being sent. In the front end, the user is shown a standard file input HTML control. Once they have chosen a file to upload and click the file upload button, the following code is triggered:

```
var files = document.getElementById("fileUpload").files;
var form = new FormData();
form.append('file', files[0]);
```

Figure 34 - Creating a FormData Object

A JS FormData object is created to simulate a form being used, the standard method for POSTing data to a URL.

```

fetch('/ufile', {
  method: 'POST',
  body: form
}).then(response => response.json())
.then(function(response) {
  if(response.code !== 200) {
    alert(response.message)
  }
  else {
    for(var i = 0; i < response.message.length; i++) {
      addMarkerArray(response.message[i].caption, response.message[i].entities, map);
    }
  }
})

```

Figure 35 - POSTing the Request

A fetch is formulated, using the POST method and appending the FormData object to the body. Once a response is received, appropriate error handling is implemented to handle errors and, if none have occurred, each caption is added to the map.

In terms of security, file uploads usually present a strong risk to the user, as deploying potentially executable code to the server could result in major breaches and damage to the application. In order to prevent this, the application tests for a secure filename using the “secure\_filename” function provided by Werkzeug Utils. It also prevents the upload of any file with an extension that is not in the “allowed\_extensions” list. This is currently limited to CSV files only.

## 5.5 – Front End

### 5.5.1 – Add Marker and Add Marker Array

Two custom methods have been implemented to add markers to the map, as well as add arrays of markers to the map:

```
function addMarker(cap, lat, long, address, map) {  
    var marker = L.marker([lat, long]).addTo(map);  
    marker.bindPopup("<h1>" + cap + "</h1>" + address);  
}
```

Figure 36 - Adding a Marker

A marker is generated on the given latitude and longitude, and then a popup is bound to the marker with the caption, and the address of the entity.

```
function addMarkerArray(caption, array, map) {  
    var prev = null;  
    var r = Math.floor(Math.random() * 255);  
    var b = Math.floor(Math.random() * 255);  
    for(var i=0; i < array.length; i++) {  
        addMarker(caption, array[i].lat, array[i].long, array[i].address, map);  
        if(prev != null) {  
            var points = [[prev.lat, prev.long], [array[i].lat, array[i].long]];  
            new L.polyline(points, {  
                color: "rgb("+r+",255,"+b+")"  
            }).addTo(map).bindPopup("<h1>" + caption + "</h1>");  
        }  
        prev = array[i];  
    }  
}
```

Figure 37 - Adding a Marker Array

In order to indicate that certain entities belong to the same caption, the above code chooses a random green colour, and pins a line between entities in the same caption.

The program currently requires a CSV file with the captions to be stored under a column named “caption”. There is support for an optional column called “country” which will be fed to the bias functionality. Any other columns will be ignored.

## 5.6 – Custom NLP Models

As mentioned in sections 2.4 and 3.1, the option provided by SpaCy to train a new statistical model was an option that was taken advantage of with this project.

To that extent, a new model was created with the name “en\_aiglobe” in order to make an attempt at improving upon the accuracy of the inbuilt models that SpaCy provides, as well as solving another issue (detailed in section 5.7.2). This section will explain the process of training the new model.

In order to train a new statistical model, training data must be “annotated”. This process is carried out by selecting data (in this case, natural language captions), and manually classifying each of the entities within the caption. A large number of these captions must be annotated in order to gain any

significant increase in performance against the data (Shrivarsheni, [No Date]). These large datasets are also often referred to as corpora.

For SpaCy, the figure below illustrates the format in which data must be annotated:

```
("Dunedin , New Zealand'", {"entities": [(0, 22, "GPE")]}),
```

Figure 38 - A Manually Annotated Caption

The caption must be listed, followed by a list of (again, in this case) the entities that can be found in the caption. The entity is described by its start index, its end index, and the classification that it belongs to.

Two of these datasets are required, one for training the data and one (unseen) for evaluating the data.

In total, 262 captions were annotated for this project, 162 for training and 100 for evaluation.

Once these datasets were created, they were converted to the SPACY training format, and passed to SpaCy's inbuilt training process, the output of which can be seen below:

```
c:\Users\cjs\Documents\Uni Modules\Year 3\CM3203 - Individual Project\Data\Spacy Training Data>py -m spacy train ../"Spacy Config"/config.cfg --output ./output --paths.tr
ain ./train.spacy --paths.dev ./eval.spacy
Created output directory: output
Using CPU
[On]
===== Initializing pipeline =====[On]
Set up nlp object from config
Pipelines: ['tok2vec', 'ner']
Created vocabulary
Added vectors: en_core_web_lg
Finished initializing nlp object
Initialized pipeline components: ['tok2vec', 'ner']
Initialized pipeline

===== Training pipeline =====
Pipeline: ['tok2vec', 'ner']
Initial learn rate: 0.001
#      LOSS TOK2VEC      LOSS NER      ENTS_F      ENTS_P      ENTS_R      SCORE
0       0          0.00      52.44      0.00      0.00      0.00      0.00
8      200          97.09     3521.95     66.67     59.93     75.11     0.67
19     400          43.84     468.93     66.27     59.01     75.57     0.66
32     600          21.83     86.55     70.16     63.27     78.73     0.70
49     800           3.31      8.81     68.05     62.84     74.21     0.68
69    1000          13.16     26.16     65.07     58.21     73.76     0.65
93    1200           5.76     79.84     68.32     62.98     74.66     0.68
124   1400          75.61    108.58     68.55     61.82     76.92     0.69
160   1600          41.95     25.35     69.08     62.09     77.83     0.69
204   1800           9.53      7.05     68.13     60.85     77.38     0.68
256   2000          22.61     16.37     67.22     62.40     72.85     0.67
322   2200           4.60      5.55     66.80     59.57     76.02     0.67
Saved pipeline to output directory
output\model-last
```

Figure 39 - Training a Custom NLP Mode

Once this process has completed, the output is a custom NLP model that can be used in lieu of the inbuilt models.

## 5.7 – Unforeseen Issues in Development

The development for this project was, for the most part, relatively issue-free. That is not to say that no issues were discovered, as those will be detailed below, but this project came together with less issues than the author has experienced in previous projects.

The author believes this ease of development can be attributed to the choice of tools at the beginning of this project. Libraries with good documentation and large communities were prioritised, and the author believes these choices led to tools that have been well-used and well-documented, which meant that any issues with functionality were resolved with minimal research and effort.

### 5.7.1 – Rate Limiting

One of the major issues that was discovered over the course of developing this project is the fact that it is not possible to send uncapped consecutive requests to the Nominatim Geocoding API. As Nominatim runs on donated servers, sending multiple requests in parallel or consecutively could

overload their API. To combat this, Nominatim has a strict 1 request per second quota, which needs to be adhered to.

Thankfully, this quota can be met by using a Rate Limiter, which is provided by the GeoPy library. To ensure that the application does not exceed the 1 r/s limit, the Rate Limiter was simply wrapped on to the geocoder object.



### 5.7.2 – Hierarchical Addresses and Entity Condensing

An issue that was found with georeferencing captions is that the inbuilt model often identified more entities than necessary.

To expand on this, the specific issue that is being referenced can be found in the caption “Pakaraka, Northland”. A standard NER model would, correctly, identify 2 entities in this caption: Pakaraka and Northland.

In this case, however, “Pakaraka, Northland” is really referring to one place: Pakaraka, which is located within Northland. In the vast majority of the data that this application is intended to be used for, commas actually denote a hierarchical address, where the first entity is within the second entity.

It follows, then, that there is no need to place the second entity on the map, as it conveys unnecessary information.

In order to deal with this, we introduce the idea of “entity condensing” which, when presented with an example like above, should join the two entities together.

The first idea was to code in a solution to this, but it was realised that a more elegant solution would be to ‘teach’ an NLP model to condense these entities.

As training data was already being annotated to improve the results of the NLP model in use at the time, examples were added showing that “Pakaraka, Northland” should be considered not two entities but one.

## 6 – Results and Evaluation

This section will evaluate and explain the results that have been achieved by the project. It will justify the choices made in relation to the results achieved and will evaluate the choices made over the development cycle of the project. It will give metrics, where possible, relating to the efficiency and performance of the project, and will explain how those metrics have been gathered.

### 6.1 – Custom NER Model Vs. Pre-trained NER Model

As outlined in section 2.7, the efficacy of an NLP classifying model can be demonstrated using the F1 evaluative measure.

In order to do this, the data must be defined with respect to the equations for precision, recall, and F1 defined in section 2.7.

True Positives, False Positives, and False Negatives (with respect to the annotated captions shown in section 5.6) are defined as follows:

- A True Positive is counted when the model identifies an entity as belonging to the correct class (as defined with the ground truth found in the evaluation data)
- A False Positive is counted when the model correctly identifies that an entity is a toponym but does not tag it as belonging to the correct class.
- A False Negative is counted when the model incorrectly identifies an entity

Note that for this task that if a false positive is counted, a false negative is also counted.

Seen below is a comparison of the F1 values of the custom “en\_aiglobe” model versus the “en\_core\_web\_trf” model, provided by SpaCy v3.0:

```
"ents_f":0.7016129032,  
"ents_p":0.6327272727,  
"ents_r":0.7873303167,
```

Figure 40 - The F1 score of the en\_aiglobe NLP model

```
"ents_p":0.9051983097,  
"ents_r":0.8903745994,  
"ents_f":0.8977252645,
```

Figure 41 - The F1 score of the en\_core\_web\_trf model

The results above were taken from the meta.json file that is packaged with each model.

As can be seen, the stock en\_core\_web\_trf model has an F1 of 0.898, while the en\_aiglobe model has an F1 of 0.702. Mathematically speaking, the custom NLP model is wildly outperformed by the trf model. While it may seem, then, illogical to use the en\_aiglobe model instead of the trf model, it must be noted that the en\_aiglobe model implements the entity condensing technique outlined in section 5.7.2. It is also worth noting that the F1 score also considers the classification of each entity (geopolitical entity, facility, location, organisation), when this information is not utilised by the application. The application only needs to know that an entity has been found, as opposed to its classification.

To that extent, the performance difference between the two becomes acceptable, and justifies the use of the custom en\_aiglobe model.

To demonstrate the efficacy of the entity condensing technique, below is a comparison of the entities identified by each model when passed the caption “Pakaraka, Northland”:

```
Pakaraka, Northland
127.0.0.1 - - [07/May/2021 14:43:25] "GET /geo/Pakaraka,%20Northland/None HTTP/1.1" 200 -
```

Figure 42 - Using en\_aiglobe to entity condense

```
Pakaraka
Northland
127.0.0.1 - - [07/May/2021 14:45:21] "GET /geo/Pakaraka,%20Northland/None HTTP/1.1" 200 -
```

Figure 43 - Using en\_core\_web\_trf to parse a hierarchical caption

Further to that, below is the graphical output of parsing this caption with the trf model:

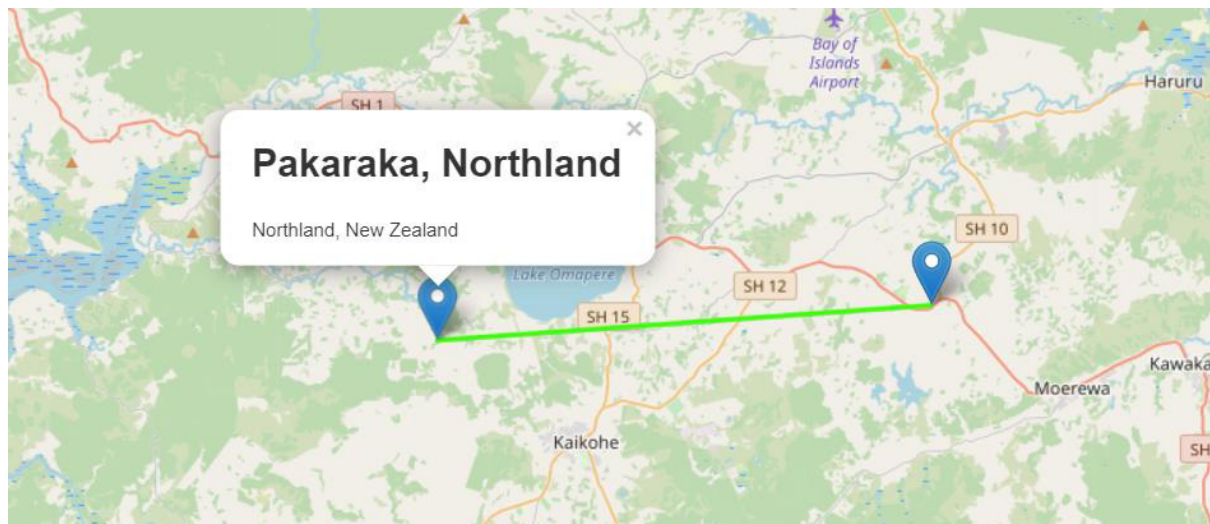


Figure 44 - Unnecessary data in the graphical output when using en\_core\_web\_trf

As can be seen, the extra point for Northland has been added, which is unnecessary for the user.

## 6.2 – Evaluation of Requirements

With respect to the original requirements of this project, as defined in section 4.1, this project can be considered a success. The author has succeeded in implementing the functionality defined in the design section, as well as implementing extra functionality such as the custom en\_aiglobe model.

It is the author’s opinion that, given more time, the application could be shaped into a full business-scale model, with potential to satisfy the needs of a client

## 6.3 – Time Performance

Evaluating the time performance of the project is somewhat difficult due to the inconsistent nature of internet download and upload speeds, which the project utilises for the external API calls to the MongoDB cluster for storage and the Nominatim API for geocoding.

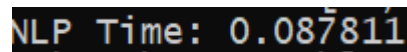
The following metrics were obtained using the Brave browser, with a quoted internet speed of 20ms ping, 220.44 mbps download, and 21.06 mbps upload. These metrics are correct for the machine

that the application was tested on but would vary for any other environment that the application could be used on.

### 6.3.1 – Single Geocoding

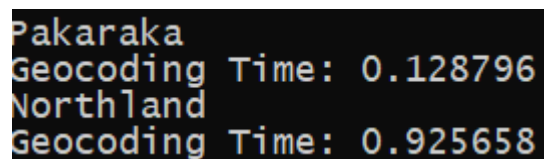
To provide transparency, two figures will be provided for this time: the time taken for NLP processing to occur and the time taken for geocoding to finalise.

The sum of these two times will be the total time taken for geocoding in the application, excluding the time taken for front end processing, as this can be considered negligible and client-specific. The times are quoted in seconds.



NLP Time: 0.087811

Figure 45 - Time Taken for Natural Language Processing of "Pakaraka, Northland"



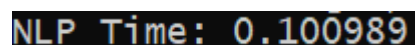
Pakaraka  
Geocoding Time: 0.128796  
Northland  
Geocoding Time: 0.925658

Figure 46 - Time Taken to Geocode "Pakaraka, Northland"

As can be seen, the geocoding functionality presents somewhat of a bottleneck to the efficiency of the application, taking almost ten times more time to process than the NLP section. This is, however, due to the external nature of the geocoding functionality, and so is not unexpected.

### 6.3.2 – Custom NLP Times versus Standard NLP Times

The time efficiency of NLP models is not standard and tends to be inversely proportional to the accuracy of the model itself. With the time quoted for natural language processing in Figure 45 in mind, the below figure records the time taken on the same caption but using the `en_core_web_trf` model as opposed to the custom `en_aiglobe` model used in Figure 45.



NLP Time: 0.100989

Figure 47 - Time Taken for Natural Language Processing of "Pakaraka, Northland" using `en_core_web_trf`

It is shown, then, that the custom `en_aiglobe` model is slightly faster than the `en_core_web_trf` model. This is likely because the `en_aiglobe` model was trained using the `en_core_web_sm` model as a basis, which is the smallest and fastest model available.

### 6.3.3 – Bulk File Geocoding

The time taken for the bulk file geocoding is mostly proportional to the time taken for single geocoding, as the bulk file method simply employs the repeated use of the single geocoding functionality. There are also time overheads associated with the opening and reading of the file, as well as the "memory method" employed to prevent double-geocoding.

| caption  | country     |  |  |  |  |  |
|--|-------------|--|--|--|--|--|
| Mangawel   | New Zealand |  |  |  |  |  |
| Waimangu Rd., 0.4 km west along road from tearooms, east side of road reserve. |             |  |  |  |  |  |
| Pakaraka, Northland. South side main north highway , opposite Pakaraka School. |             |  |  |  |  |  |
| 10.4 KM N  | New Zealand |  |  |  |  |  |
| Judgeford. Abbotts farm, 1.5 miles east of Pauatahanui.                        |             |  |  |  |  |  |

Figure 48 - Input File for Bulk Geocoding Time Test

**File Geocoding Time: 9.378259**

Figure 49 - Time taken to process a 5-line file

Roughly 10 seconds taken to process a 5-line file, which is within acceptable limits for bulk geocoding.

#### 6.3.4 – Searching

The below times describe the time taken for the flat search and the radial search functionality of the application.

Firstly, the flat search will be timed, using the search query “England”:

**Flat Search Time: 0.232379**

Figure 50 - Flat Searching with Query "England"

To compare, the radial search window will be placed over the UK on the map in order to return the same results as the flat search functionality:



Figure 51 - The location of the Radial Search Window

**Radial Search Time: 0.303143**

Figure 52 - Radial Searching over the UK

The time taken to perform a radial search is significantly longer than the flat search because the flat search employs a native indexing method in the database instance, while the radial search employs a non-indexed “naïve” search algorithm.

#### 6.4 – Evaluation of Tools Used

The tools that I chose to implement this project were, in my opinion, the correct tools to use for this application. Specifically, the presence of clear documentation was a strong theme across each tool, and I believe this documentation gave far more benefit to the development of the project than, for example, the small differences in quoted F1 values mentioned in section 3.1.

#### 6.4.1 – Python

Using Python allowed me to bridge the gap between the data processing needs of machine learning and the requirements for a web application. Due to its extensive library catalogue, implementing new features to the application meant simply searching for a library that had already been published to solve the task.

Python is an extremely well-documented language, which presented a very large benefit to this project. Due to its simplicity and sufficient error handling, errors came few and far between. When they were encountered, they were solved quickly due to the presence of clear documentation.

Python also boasts well-used libraries with similarly clear documentation, which extended the clarity of development to not only the core code, but external code as well.

#### 6.4.2 – SpaCy

Perhaps due to being built for the Python platform, SpaCy matches Python's reputation for clear documentation.

With machine learning being a new topic to the author, implementing a machine learning classifier for NLP could have been a very difficult task, were it not for SpaCy's clear documentation and tutorials for the implementation of common uses for their library.

The implementation of the features needed for this project were made very simple due to this, and as such, no major issues with the implementation of the major functionality of the program were found.

The documentation was, however, somewhat complex when it came to training a custom model. The upgrade from SpaCy v2.x to v3.0 changed the training process dramatically, and it was found that the documentation to be lacking a clear explanation on training a new model in the context that I needed.

SpaCy, however, not only has clear documentation but a mature community of users. By following a tutorial (Lim, 2021), this issue was solved.

#### 6.4.3 – GeoPy and Nominatim

GeoPy was overall satisfactory in meeting the requirements for the project. Its connection to multiple geocoding APIs meant that the library was relevant to the project no matter the choice of API

Nominatim was also satisfactory to the project at this scale but would be unsuitable for use at a larger scale. This will be expanded upon further in section 7.3

The author found Nominatim to be no less accurate than Google Maps, with the added benefit of being free to use.

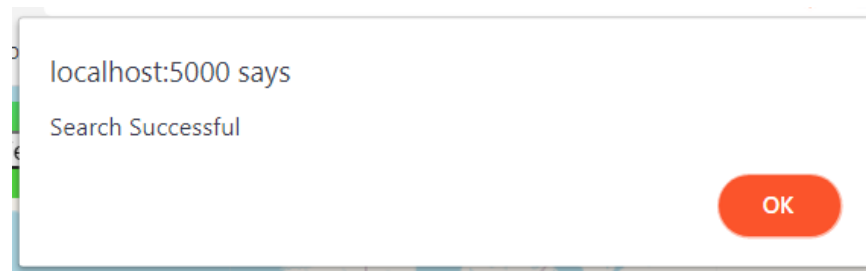
## 6.5 – Evaluation of Usability

In order to self-evaluate the usability of a system, there exist various sets of “heuristic” guidelines that can be used to judge the system. This paper will evaluate the project using perhaps the most famous of these heuristics, Jakob Nielsen’s 10 heuristics (Nielsen, 1994a). The definitions for each of these heuristics have been used from (Nielsen, 1994b).

### 6.5.1 – Visibility of System Status

“The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.”

To satisfy this heuristic, the system uses the inbuilt “alert” functionality supplied with JavaScript in order to notify the user of changes to the system state, as well as notify of any errors and warnings. An example of this can be seen below:



*Figure 53 - Successful Search Notification*

### 6.5.2 – Match Between System and The Real World

“The design should speak the users’ language. Use words, phrases, and concepts familiar to the user, rather than internal jargon.”

The system is designed to be as minimal as possible but assumes that the client will understand terms such as “geocode”, due to its specialist nature.

The system quite literally matches “the real world” in that the main part of the interface is a map, something that does not take specialist training to read or understand.

All other language in the system is non-specialist and aims to introduce no unfamiliar concepts.

### 6.5.3 – User Control and Freedom

“Users often perform actions by mistake. They need a clearly marked ‘emergency exit’ to leave the unwanted action without having to go through an extended process.”

In order to minimise jarring movement and unwanted exploration, the website is limited to a single page, which allows forms to be drawn over the top of it. Each of these forms features a clear exit cross, which can be clicked on to close the form, returning the user to the original state. See below:

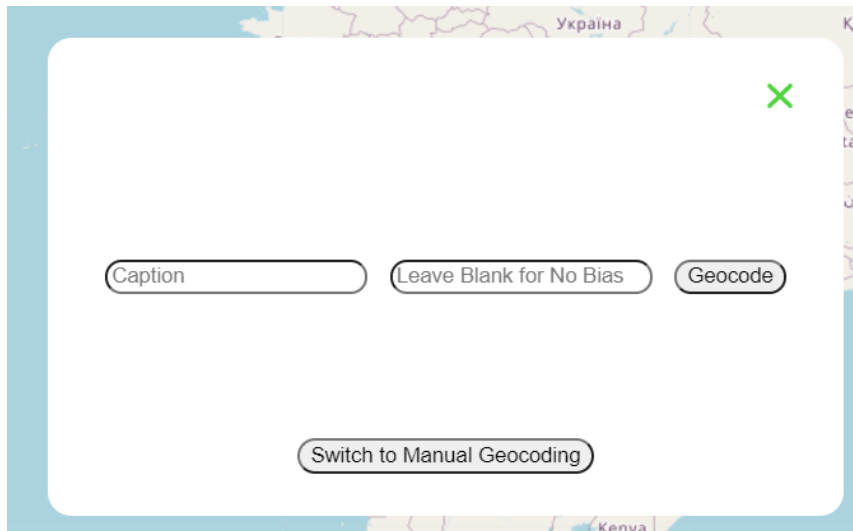


Figure 54 - Geocoding Form with Exit Cross in Top Right Corner

This cross constitutes an “emergency exit”.

### 6.5.4 – Consistency and Standards

“Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions”

In terms of design, each form on the application follows the same design standard, as well as colour theme. This means that the design and colour is standard across the entire application.

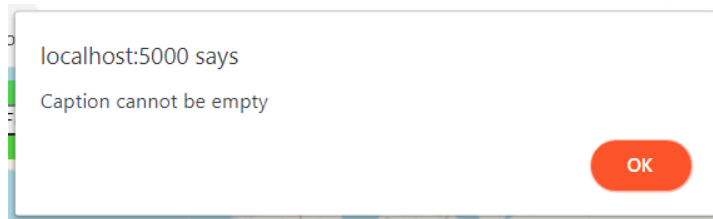
Textually, concepts are only referred to by one name. In this regard, geocoding is only ever referred to as geocoding and searching is only ever referred to as searching, etc.



### 6.5.5 – Error Prevention

“Good error messages are important, but the best designs carefully prevent problems from occurring in the first place.”

Errors are both handled and prevented in the code. In order to prevent errors, the application will prevent the user from submitting “garbage input”. An example of this is found when the user tries to submit an empty caption for geocoding, which is met with the following message:



*Figure 55 - Preventing Empty Caption Submission*

This error prevention is standard across any input in the system. All inputs must be filled before submission, and inputs that require only numbers, not text, (see radius input) disallow text input entirely.

### 6.5.6 – Recognition Rather than Recall

“Minimise the user’s memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another.”

No information is required to be remembered by the user.

### 6.5.7 – Flexibility and Efficiency of Use

“Shortcuts – hidden from novice users – may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.”

The system does not currently present the option for shortcuts to be created, but instead aims to minimise the complexity of any task in the application by minimising the number of clicks needed to perform the task.

### 6.5.8 – Aesthetic and Minimalist Design

“Interfaces should not contain information which is irrelevant or rarely needed.”

The system’s design aims to be as minimalistic as possible, taking up only a small portion of the page. There is no unnecessary information contained on the page, and the options element can be shrunk to fit only a small corner of the page.

The figure below demonstrates the page with minimum information displayed:



Figure 56 - Minimum Information Displayed

The next figure shows the page with the maximum amount of information displayed:

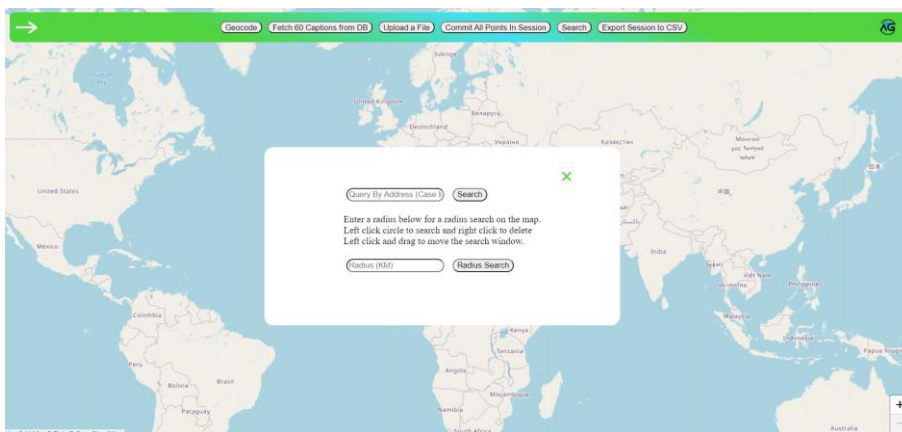
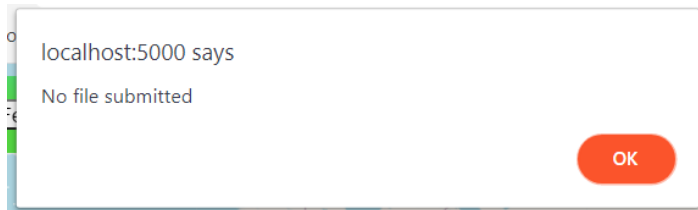


Figure 57 - Maximum Information Displayed

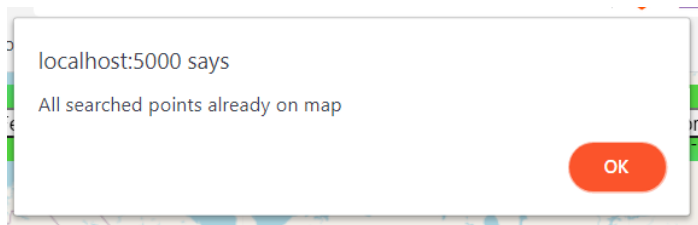
### 6.5.9 – Help Users Recognise, Diagnose, and Recover from Errors

“Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution”.

The application contains error handling for most known issues, as well as error handling for unknown issues. While it returns error codes, it also returns error messages to accompany these error codes. Some examples of these can be seen below:



*Figure 58 - Error Message for No File Submitted*



*Figure 59 - Error Message for No New Points Found*

All error messages for known errors across the system explain the issue.

### 6.5.10 – Help and Documentation

“It’s best if the system doesn’t need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks”

The system as of now does not contain a help page but could be adapted to contain a help page if users found the system to be complex or confusing. The lack of help page can be attributed to the fact that the application is a specialist application and assumes a level of knowledge of the user to interact with the minimal interface.

## 6.6 – Testing

While they have been demonstrated over the course of this paper, this section will serve to demonstrate and test each of the main functionalities of this project, in order to ensure that they work as intended.

### 6.6.1 – Single Geocoding

Using the caption “Pakaraka, Northland. South side main north highway, opposite Pakaraka School.”, taken from the Landcare dataset.

Expected result is 2 tags to be placed on the map, one on Pakaraka, Northland, and one on Pakaraka School. A line should be drawn between the two of them.

See below:

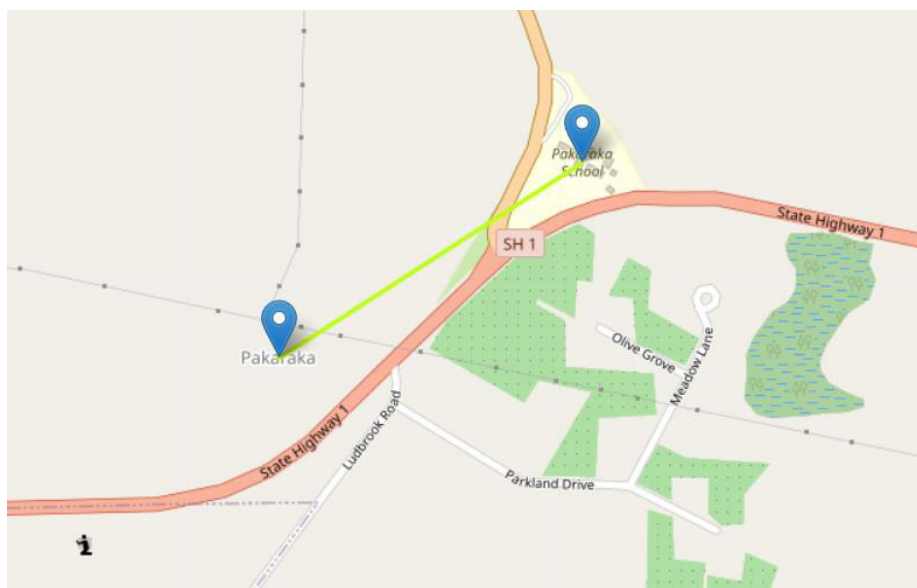


Figure 60 - Demonstration of Single Geocoding

As expected, a point has been placed each for Pakaraka, and Pakaraka School, with a line joining the two of them.

## 6.6.2 – Bulk Geocoding

Uploading the file shown in Figure 48. Expected result is 5 separate captions visible on the map.

See below:

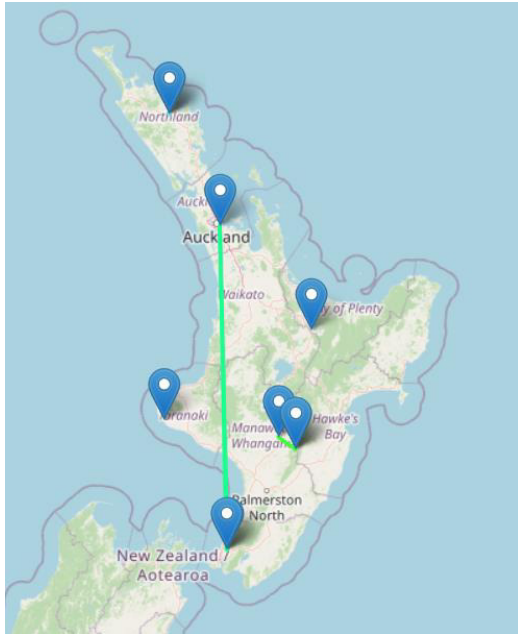


Figure 61 - Bulk Geocoding Demonstration

5 captions can be seen, all correctly geocoded.

## 6.6.3 – Export Session

Using the session persisting from section 6.6.2, expecting to receive a file containing 5 lines, 1 for each caption.

```
caption,entities
"Mangaweka Hill, 6 miles north of Mangaweka and 8 moles south of Taihape. On ""island"" formed by straightening road.",["{'entity': 'mangaweka hill', 'address': 'Mangaweka, Rangitikei District, Manawatu-Whanganui, New Zealand', 'lat': -39.8177827, 'long': 176.0854377}, {'entity': 'mangaweka', 'address': 'Mangaweka, Rangitikei District, Manawatu-Whanganui, New Zealand', 'lat': -39.8177827, 'long': 176.0854377}, {'entity': 'taihape', 'address': 'Taihape, Rangitikei District, Manawatu-Whanganui, 4720, New Zealand', 'lat': -39.6766915, 'long': 175.7983062}]"
"Waimangu Rd., 0.4 km west along road from tearooms, east side of road reserve.",["{'entity': 'waimangu rd', 'address': 'Waimangu Road, Waimangu, Rotorua, Bay of Plenty, New Zealand', 'lat': -38.2786079, 'long': 176.3585281}]"
"Pakaraka, Northland. South side main north highway , opposite Pakaraka School.",["{'entity': 'pakaraka northland', 'address': 'Pakaraka, Bay of Islands-Whangaroa Community, Far North District, Northland, 0472, New Zealand', 'lat': -35.3585645, 'long': 173.953374}, {'entity': 'pakaraka school', 'address': 'Pakaraka School, State Highway 10, Pakaraka, Bay of Islands-Whangaroa Community, Far North District, Northland, 0472, New Zealand', 'lat': -35.356221500000004, 'long': 173.9578498255819}]"
"10.4 KM NE OF OPUNAKE",["{'entity': 'opunake', 'address': 'Opunake, South Taranaki District, Taranaki, 4616, New Zealand', 'lat': -39.4546624, 'long': 173.8601616}]"
"Judgeford. Abbotts farm, 1.5 miles east of Pauatahanui.",["{'entity': 'judgeford', 'address': 'Judgeford, Porirua City, Wellington, 5381, New Zealand', 'lat': -41.1166793, 'long': 174.9513694}, {'entity': 'abbotts farm', 'address': 'Abbotts Way, Ellerslie, Ōrākei, Auckland, 1051, New Zealand', 'lat': -36.88733, 'long': 174.8191731}, {'entity': 'pauatahanui', 'address': 'Pauatahanui, Porirua City, Wellington, 5381, New Zealand', 'lat': -41.1054459, 'long': 174.9170723}]"
```

Figure 62 - Demonstration of Export Feature

#### 6.6.4 – Searching

For this test to be understood, it must be noted that as of writing this paper, the database currently contains 5 captions that are located within New Zealand. In order for the search test to be successful, when searching with New Zealand as the search query or having New Zealand within the search radius, all 5 captions must be returned.

Initially, the flat search is performed with search query “New Zealand”.

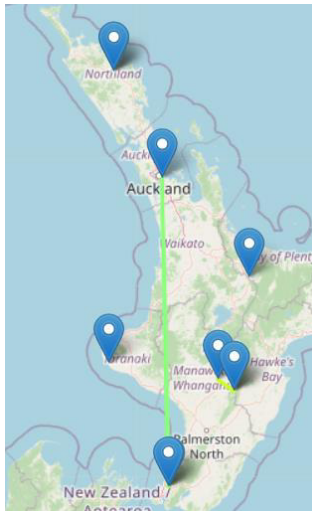


Figure 63 - Search Demonstration

The flat search can be considered successful.

Secondly, the radial search is performed:

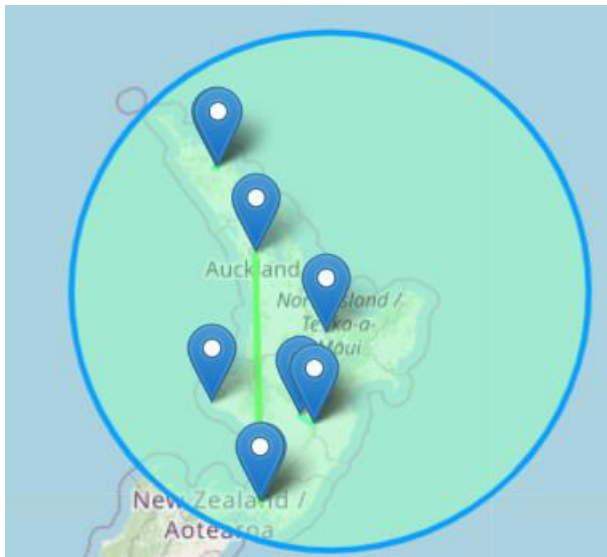


Figure 64 - Radial Search Demonstration

This search is also successful.

## 7 – Future Work and Alternative Solutions

While this project has succeeded in filling the original requirements, there is also a multitude of further work that could be applied to it. This section will detail the extra functionality and solutions that could be implemented with more time and resources, and the choices that would have been made given a business-scale deployment of this application.

### 7.1 – Document Clustering

The app currently attempts to solve geo-geo ambiguity on a somewhat basic level. The biasing feature will solve conflicts between countries, but only in cases where the bias is known.

Not only this, but geo-geo ambiguity still persists within countries when one considers common names such as street names or villages.

There are multiple proposed solutions to minimising geo-geo ambiguity, but one that shows promise is the idea of document-level clustering and minimising footprints.

The idea of this solution is to consider the list of potential candidates for a geocoded location and choose the candidate that is closest to the other already identified and geocoded point(s) in the document.

This solution would be an improvement upon the application's current solution to the issue of geo-geo ambiguity. It is not, however, without flaw. It requires at least 2 entities to be within a document to work, as well as at least 1 point to be geocoded with surety.

### 7.2 – Database Choice

The current database choice for the application is a MongoDB M0 Sandbox Replica Set Cluster, consisting of 3 nodes, located in Ireland. The cluster has a 512 Mb max storage. This option was chosen simply because it was free.

Clearly this storage space would not be feasible were this project to continue to a business deployment scale. In that regard, given a budget, this cluster would be upgraded to an M1 or above, depending on the requirements of the client.

### 7.3 – Geocoding API Choice

As with the database choice defined in section 7.2, Nominatim was chosen largely due to the fact that it was free. Nominatim is, however, limited. The limit of 1 request per second satisfies the requirements of the project in its early stages, but if it were to continue to business deployment, I would recommend utilising a different geocoding API.

To that extent, and assuming a higher budget, the author would recommend a service with a higher RPS quota, as well as a more “well-established” service such as Google's mapping API, which provides geocoding functionality.

### 7.4 – Multithreading

Assuming the implementation of a geocoding API with a higher RPS quota as outlined in section 7.3, a new bottleneck for the application would be found in sending consecutive requests.

To improve performance, the application could be modified to implement multithreading, sending parallel requests to the API. With this implemented, a drastic delay decrease would be noted, which would increase the performance of request heavy operations such as the file handling functionality that the application implements.

## 7.5 – Geospatial Indexing

As mentioned in section 5.3.2, the current method for radial searching from a point is to test each of the entities within the database and check their distances from the centre of the search window with the Vincenty distance algorithm.

While this solution is feasible for the relatively small amount of data that is currently stored within the database, this would not be recommended for larger datasets, and would present a serious performance bottleneck to the application.

Instead, the author would recommend making use of the geospatial indexing feature that is common in NoSQL databases such as MongoDB. By placing a geospatial index on each of the entities, the performance could be greatly increased when testing for presence within the search window.

This would, however, require a reformat of the current data to match the necessary data format for geospatial indexing in MongoDB.

## 7.6 – Georeferencing Accuracy

As has been seen in this report, the task of georeferencing natural language to a high degree of accuracy is a difficult challenge. The application currently works around this by ignoring the low-level adjustments to the mentioned location found in the caption “3km north of Swindon” and instead places a point directly on Swindon (as opposed to 3km north).

To this extent, there are a multitude of ways of expressing the translation of a given point from the mentioned location, as is mentioned in section 2.5.

Potentially the largest challenge to be found in future work will be solving this problem. If solved, however, it would result in an application with almost unlimited accuracy for georeferencing documents.

## 7.7 – Future Opportunities

Over the course of this project, it was expressed by the National Museum of Wales that there could be an interest in a project similar to this one.

In that regard, this project could be integrated into the systems of an interested party. Due to the split between the front end and back end of the project, the back-end alone could be treated as an API for another system to use, and so this project could be adapted to be used by any interested stakeholders for the purpose of collection digitisation.



## 8 – Conclusions

To finish a project, one must state the conclusions that have been derived over the course of the work done. With an application-based project such as this one, this can be slightly difficult as there has been no hypothesis proven nor disproven. This section will, however, attempt to sum the lessons that can be learned from this project, as well as compare the result with respect to the original introduction and requirements.

This project was undertaken with the idea of creating a web application that could map the locations mentioned in natural language.

Over time, it evolved into a project intended specifically for georeferencing natural language data for ecological documentation and preservation, with the target stakeholders of ecological organisations looking to digitise and georeference collections.

The project can, overall, be considered a success in that it met each of the initial requirements defined. It has achieved the goal of mapping the locations in texts, and all sub-goals that were chosen to meet the main objective.

The system demonstrated in this report has made an attempt at solving the issues of geo-geo ambiguity and non-geo-geo ambiguity, as well as the other known issues within the field of automated georeferencing.

It has been shown that a wide scale system designed to automate the georeferencing of texts, be it for collection digitisation or otherwise, is not only feasible but easily within reach. Combining the system that has been produced by this report with other, more complex, solutions defined in the material referenced in this report would result in a system capable of georeferencing documents to a high degree of accuracy.

This report can be considered as a proof-of-concept for a future georeferencing application that would alleviate, if not cut entirely, the hours and manpower needed for georeferencing the massive non-digital collections of which many museums around the world are currently in possession.

This report also serves to demonstrate the power of modern-day machine learning solutions such as those implemented in the SpaCy library. Their base accuracy and the ease of which they can be trained has increased many-fold over the years and is leading to their widespread use in non-specialist applications.

## 9 – Reflection

In the spirit of continual improvement, it is good practice to reflect upon the transferrable lessons and skills that have been learned over the course of a project. This section will reflect upon the skills and lessons that have been learned over this project.

I originally undertook this project with a light scepticism as to the amount of material that could be covered in the relatively long time that this project ran for.

This initial assumption was entirely wrong, as I realised how large the scope of the issue of natural language processing is.

In that regard, I have learned to avoid assuming the scale of a project before the background research has been conducted, so as to avoid misjudging the scale of the project at hand.

I have also learned the value of “choosing the correct tools for the job”. In my experience with previous projects, I took less time in choosing a tool/library for a given solution and often found the tool to be lacking or broken at a later time, when refactoring was too difficult to be considered.

For this project, I took a serious approach to the choice of tools, weighing performance and efficiency but most importantly documentation and community. I found this approach to be wildly superior to the alternative, as when issues inevitably occurred, fixing them was made easy by the documentation and community support.

On a higher scale, this project introduced me to the idea of remote work. Due the COVID-19 pandemic, this project was undertaken remotely, with a supervisor in New Zealand. While modern video conferencing technology allows such a situation to be entirely feasible, it is still not comparable to working with a supervisor in person.

I have found the challenge of self-supervision and motivation to be one of the most important lessons that I have learned over the course of this project, skills which will transfer and be very valuable towards my future work.

Another valuable, transferrable skill learned from this project is the tactic of “document-as-you-go”. Many of the sections in this project, especially section 3 and all subsections, were written as and when the decisions were being made. This made the collation of this report not only easier towards the end of the project, but the sections are more accurate to the decisions that were made at the time, as opposed to having been written at a later date when potentially the reasoning for choices has been lost.

This idea of document-as-you-go will be retained for all future projects.

In terms of my approach to the project, I believe that it was well planned. There was a potential, however, for more research into the material surrounding the issue to be conducted at an earlier phase in the project, which would have given me more time to formulate elegant solutions to problems mentioned in the surrounding material.

Overall, this project has been a success, and has taught me numerous lessons which I will keep in close consideration for my future projects.

## 10 – References

- Adams, B. et al. 2015. Frankenplace: Interactive Thematic Mapping for Ad Hoc Exploratory Search. In: Adams, B. et al. eds. *WWW '15 Proceedings of the 24th International Conference on World Wide Web*. Florence, Italy. May 2015. Florence, Italy. 10.1145/2736277.2741137
- Amitay, E. et al. 2004. Web-a-Where: Geotagging Web Content. In: Amitay, E. et al. eds. *SIGIR '04: Proceedings of the 27<sup>th</sup> annual international ACM SIGIR conference on Research and development in information retrieval*. Sheffield, UK, 25-29<sup>th</sup> July 2004. New York, NY, USA. Association for Computing Machinery. pp. 273-280. 10.1145/1008992.1009040
- Beaman, R. and Conn, B. 2003. Automated geoparsing and georeferencing of Malesian collection locality data. *Telopea* 10(1). 43-52. 10.7751/telopea20035604.
- Dishmon, C. [No date]. Testing NLTK and Stanford NER taggers for Speed. Available at: <https://pythonprogramming.net/testing-stanford-ner-taggers-for-speed/>. [Accessed: 10/02/2021]
- Duffy, S. 2020. Is Flair a suitable alternative to SpaCy?. Available at: <https://medium.com/@sapphireduffy/is-flair-a-suitable-alternative-to-spacy-6f55192bfb01>. [Accessed: 09/02/2021]
- Flask. [No Date]. *Uploading Files*. [Code Tutorial]. Available at: <https://flask.palletsprojects.com/en/1.1.x/patterns/fileuploads/>. [Accessed: 12/04/2021]
- Gonfalonieri, A. 2018. *How Amazon Alexa works? Your guide to Natural Language Processing (AI)*. Available at: <https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3>. [Accessed: 31/03/2021]
- Holcombe, J. 2020. *Why should we use Flask for web development?* Available at: <https://www.quora.com/Why-should-we-use-Flask-for-web-development>. [Accessed: 22/02/21]
- Jones, C.B. et al. 2002. Spatial information retrieval and geographical ontologies: An overview of the SPIRIT project. In: Jones, C.B. et al. eds. *SIGIR '02: Proceedings of the 25<sup>th</sup> annual international ACM SIGIR conference on Research and development in information retrieval*. Tampere, Finland, 11-15<sup>th</sup> August 2002. New York, NY, USA. Association for Computing Machinery. pp. 387-388. 10.1145/564376.564457
- Lieberman, M. et al. 2007. STEWARD: Architecture of a Spatio-Textual Search Engine. In: Lieberman, M. et al. eds. *GIS '07: Proceedings of the 15<sup>th</sup> Annual ACM International Symposium on Advances in Geographic Information System.*, Seattle, WA, USA, 7-9<sup>th</sup> November 2007. New York, NY, USA. Association for Computing Machinery. pp. 1-8. 10.1145/1341012.1341045
- Lim, Z. 2021. *Using spaCy 3.0 to build a custom NER model*. Available at: <https://towardsdatascience.com/using-spacy-3-0-to-build-a-custom-ner-model-c9256bea098>. [Accessed: 07/05/2021]
- Marshall, C. 2019. *What is named entity recognition (NER) and how can I use it?*. Available at: <https://medium.com/mysuperaai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d>. [Accessed: 12/05/2021]
- Matan, A. 2017. *Difference Between Vincenty and Great-Circle Distance Calculations?* Available at: <https://gis.stackexchange.com/questions/84885/difference-between-vincenty-and-great-circle-distance-calculations>. [Accessed: 29/04/2021]

- Melo, F. and Martins, B., 2017, Automatic Geocoding of Textual Documents: A Survey of Current Approaches, *Transactions in GIS* 21(1), 3-38
- Murphey, P.C, et al., 2004, Georeferencing of museum collections: A review of problems and automated tools, and the methodology developed by the Mountain and Plains Spatio-Temporal Database Informatics Initiative (Mapstedi), *Phyloinformatics* 3, 1-29
- Nielsen, J. 1994a. Enhancing the Explanatory Power of Usability Heuristics. In: Nielson, J. eds. *CHI '94: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Boston, MA, USA, 24-28<sup>th</sup> April 1994. New York, NY, USA. Association for Computing Machinery. Pp. 152 – 158.
- Nielsen, J. 1994b. *10 Usability Heuristics for User Interface Design*. Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/>. [Accessed: 13/05/2021]
- OSM Foundation. No Date. *Nominatim Usage Policy*. Available at: <https://operations.osmfoundation.org/policies/nominatim/>. [Accessed: 09/02/2021]
- Roldós, I. 2020. *Named Entity Recognition: Concept, Guide and Tools*. Available at: <https://monkeylearn.com/blog/named-entity-recognition/>. [Accessed: 09/02/2021]
- Shen, P. 2019. *Best named entities recognition and entities extraction tools for NLP AI: A 2019 Comparison*. Available at: <https://medium.com/analytics-vidhya/best-named-entities-recognition-and-entities-extraction-tools-for-ai-nlp-a-2019-comparison-b30b20cf690e>. [Accessed: 09/02/2021]
- Shrivarsheni. [No Date]. *How to Train spaCy to Autodetect New Entities (NER) [Complete Guide]*. Available at: <https://www.machinelearningplus.com/nlp/training-custom-ner-model-in-spacy/>. [Accessed: 12/05/2021]
- Smith, R. 2020. *The Key Differences Between Rule-Based AI And Machine Learning*. Available at: <https://becominghuman.ai/the-key-differences-between-rule-based-ai-and-machine-learning-8792e545e6>. [Accessed: 06/04/2021]
- SpaCy. [No Date]. *Facts and Figures*. Available at: <https://spacy.io/usage/facts-figures> . [Accessed: 12/02/2021]
- Singh, I. 2016. Google puts artificial intelligence center stage. *Geospatial World*. Available at: <https://www.geospatialworld.net/blogs/how-is-google-assistant-using-artificial-intelligence-ai/>. [Accessed: 31/03/2021]
- Singh, I. 2017. *Which is the best map projection?* Available at: <https://geoawesomeness.com/best-map-projection/>. Accessed [05/05/2021]
- Van Erp, M., et al., 2015, Georeferencing Animal Specimen Datasets, *Transactions in GIS* 19(4), 563 – 581
- W3Schools. [No Date]. *How To Create A Menu Icon*. [Code Tutorial]. Available at: [https://www.w3schools.com/howto/howto\\_css\\_menu\\_icon.asp](https://www.w3schools.com/howto/howto_css_menu_icon.asp). [Accessed: 10/03/2021]
- WebGeoDataVore. 2015. *Python Geocoders Clients Comparison*. Available at: <https://webgeodatavore.com/python-geocoders-clients-comparison.html>. [Accessed: 18/02/2021]
- Yegulalp, S. 2018. *NoSQL standouts: The best document databases*. Available at: <https://www.infoworld.com/article/3201884/nosql-standouts-the-best-document-databases.html?page=2>. [Accessed: 01/03/2021]

Zola, A. 2018. *The 5 Best Programming Languages for AI*. Available at:  
<https://www.springboard.com/blog/best-programming-language-for-ai/>. [Accessed: 22/02/21]