Bilkent University

Department of Computer Engineering

Senior Design Project

Final Report

(🦊)µ-Turjee Mæn

http://138.68.136.129/senior/

Group Members
Ali Kaviş
Baraa Orabi
Çağdaş Öztekin
Mustafa Yıldız

Supervisor
Mustafa Özdal

Jury Members
Öznur Taştan Okan
R. Gökberk Cinbiş

Table of Contents

1. Abstract

Language learning and translation are two closely related concepts. When people encounter a text in a foreign language that they do not know of, they tend to approach it from a practical perspective and wonder about the “meaning”. However, language learning process should be concerned with deeper understanding of how sentence components from two languages map onto each other. Turjeeman is a platform that gives students, language educators, and translation hobbyists, and NLP corpus creators the interface and the tools to create feature-rich translations for their various needs and purposes.

2. Final Architecture and Design

Turjeeman has a three-tier architecture that consists of following three major components: frontend client as the presentation tier, backend Python server as the application tier, and backend database servers as the data tier. Each of these components runs independently of the others, and communicate with the other components via well-defined API protocols.

2.1. Subsystem Decomposition

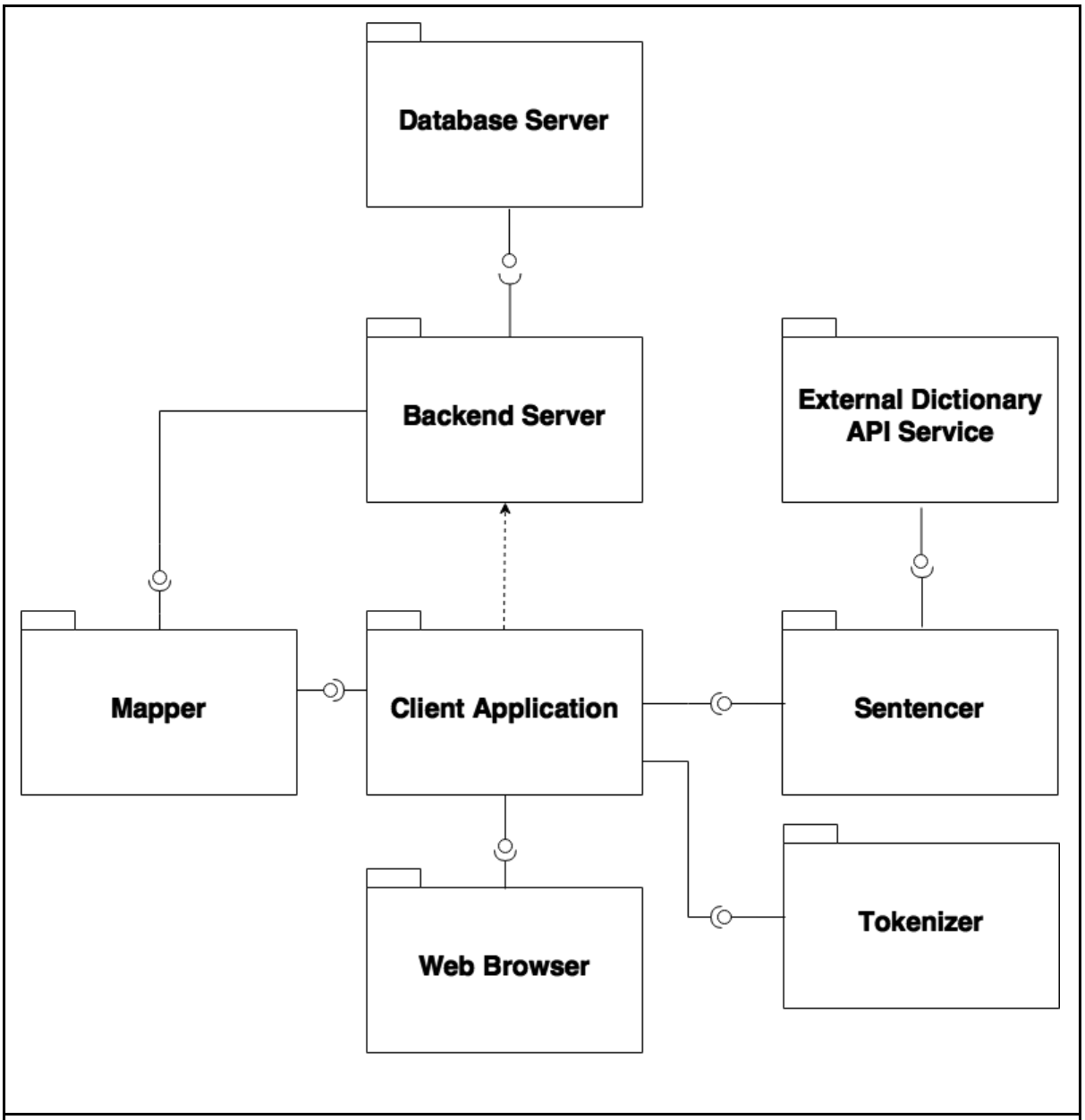

Figure 2.1.a Overall Subsystem Decomposition

2.1.1. Client Application

Client application package is a Django server that the client connect to using a typical web browser. Simple functionalities are embedded inside the client application such as manual tokenization, API for dictionary lookups and so on. Client application by itself provides an easy-to-use interface for creating translations. We aimed to design a plain view with core functionalities that will expedite the translation preparation process. The client application has the following design flow:

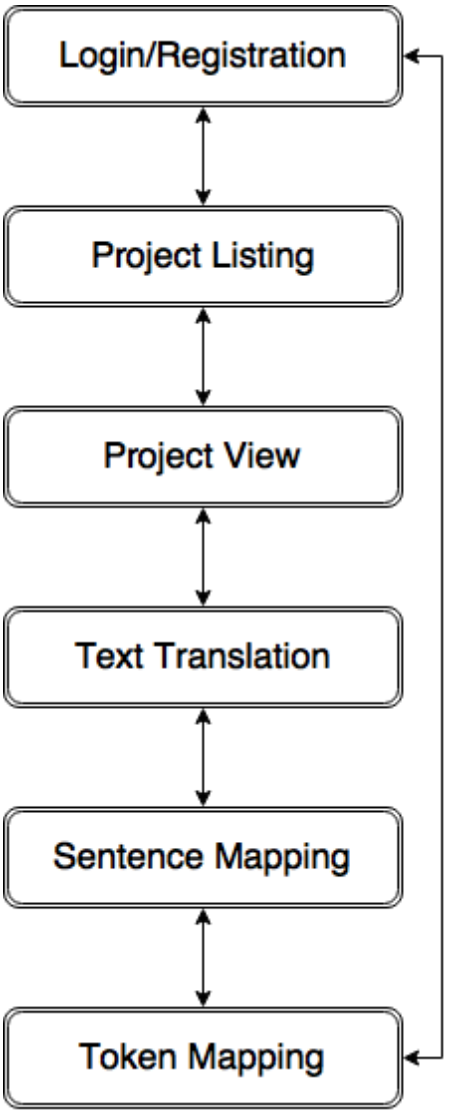

Figure 2.1.1.a Client Application design flow

The system user will use the client application as a GUI to access the system.

2.1.2. Sentencer

Sentencer is an automatic sentence-to-sentence mapper. It takes in two texts in two different languages and returns a list of pair of sentences. Our definition of sentence is beyond a regular sentence that begins with a capital letter and ends with a dot, exclamation mark, question mark or three dots. Sentences are treated as a sequence of phrases that are separated by conjunctions and punctuations. The Sentencer matches contextually matching parts of corresponding sentences to each other. It utilizes an external internet-based dictionary lookup service, which supplies word definitions and synonyms. Sentencer is essentially a Python server application.

2.1.3. Backend Server

Backend server is implementing Django web framework that interacts with client application and other python components for managing requests. Django is adopting the MVC (Model, View, Controller) pattern. For the model, Django provides an abstraction through an extra layer between its application logic and database, called Object Relational Mapper (ORM) which makes database management very portable. The model allows applications to manipulate the data without knowing the underlying database. The view is the interface of the Turjeeman, simply what users see. Django deploys client application here. The controller is controlling the information flow between the model and the view.

Django server is up at all times, even when the system is completely offline. Regular updates to Mapper model are executed offline, therefore the project file storage component must be up more often than login file storage component. Toolkits provided by django enables us to focus on developing novel solutions than implementing basic functionalities such as authentication, administration, and database connection.

2.1.4. Database Server

Turjeeman runs PostgreSQL database server. As a relational database, PostgreSQL supports JSON for document databases and the HSTORE data types for key/value pairs. With these extensions, PostgreSQL can be regarded as both SQL and NoSQL databases. Relational db functionalities of SQL is helpful to manage user authentication. NoSQL

functionalities are important because we foresaw bulk updates to the system, which we will explain later in the report. This includes users' projects (texts, sentences, mappings). Login credentials and project files are kept separately, and these files inherently have distinct structures.

2.1.5. Mapper

Mapper is an automatic token-to-token mapper. It takes in two sentences in two different languages and returns a list of mappings of the sentences' tokens. Mapper is capable of generating many-to-many token mappings, to the extent that its bipartite hypergraph model allows. Mapper model is capable of grouping and mapping tokens/token groups that it previously encountered.

Mapper relies on a hypergraph model that is based on the database of sentence mappings produced by the users. The mapper model does not guarantee a complete/comprehensive mapping, however, improves with user feedback and manual corrections. The model is subject to offline update and pruning. Mapper is essentially a Python server application.

2.2. Hardware/Software Mapping

Each of the packages of the above system decomposition can effectively be run on an separate processing node. Our system is scalable in the sense that it is modular enough to allow for easy duplication of any package on different servers to handle high demand. This would require load balancing. We do not have this designed at this stage, but we made sure to design the current system with that future in mind. Any python server and data server component can operate on a different machine without failure due to coupling. If data server needs to be changed, ORM support enables the project to be migrated easily.

2.3. Persistent Data Management

There are two main persistent data types we are keeping. The first and most important is the PostgreSQL database. The second is the hypergraph model used by the mapper package.

2.3.1. Django & PostgreSQL Servers

Django offers a high-level to keep a persistent state of user data on the server. Sessions are allowing to store users any data for their visit and have this data available for the next time the user visits the site. This structure provides convenient, robust and safe way to store the

data. These session data are stored into PostgreSQL database that in corresponding Session table, each row has a session data for a particular user hashed with sessionid. This whole process is abstracted away from the user and the developer. Django implements this via a middleware. When this session middleware is activated, each request object will have a session attribute, that is a dictionary-like object.

Django requires and stores CSRF (Cross-Site Request Forgery) tokens for any HTTP request to prevent the malicious attack that causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated. These tokens identify the both ends of requests (transmitter, receiver).

Database server does store project information in a particular format for easy access, proper management and efficient storage. Each project is saved into a separate file and stored that way unless they are removed by the user. Project files are stored as JSON files and keep a "global" state of the project with plain texts, sentence mappings and token mappings saved by the user. JSON project file has the following structure:

```
{
	"user_id" : 12345,
	"project_id" : 1,
	"title" : "The Tale of Two Cities - Translation",
	"timestamp" : 1493565851.825255,
	"language_A" : "ENG",
	"language_B" : "TR",
	"sentence_pairs" : [
		{
			"sentence_A" : "It was the best of times, it was the worst of times",
			"sentence_B" : "Zamanlarin en iyisi idi, zamanlarin en kotusu idi.",
			"mappings" : [
				{ "A" : [[0,2], [3,6]], "B" : [[20,23]]},
				{ "A" : [[7,10], [11,15]], "B" : [[11,13],[14,17]]},
				{ "A" : [[16,18], [19,24]], "B" : [[0,11]]},
				{ "A" : [[26,28], [29,32]], "B" : [[46,49]]},
				{ "A" : [[33,36], [37,42]], "B" : [[36,38], [39,43]]},
				{ "A" : [[43,45], [46,51]], "B" : [[25,35]]}
			]
		},
		{
			"sentence_A" : "It was the age of wisdom.",
			"sentence_B" : "Bilgelik cagi idi.",
			"mappings" : [
				{ "A" : [[0,2], [3,6]], "B" : [[14,17]]},
				{ "A" : [[7,10], [11,14]], "B" : [[9,13]]},
				{ "A" : [[18,24]], "B" : [[0,8]]}
```

```
                    ]
                }
        ]
}
```

Figure 2.3.1.a Example JSON Project File

The whole text is stored as sentence pairs. Initially, whole texts are treated as single sentences with empty mappings. “User_id” and “project_id” together uniquely identifies a project. “Title” is given by the user, and “timestamps” stores the time (UTF time) that the project is last saved, in seconds since epoch. “Language_A” and “language_B” are the languages of source and target texts, respectively.

“Sentence_pairs” is an array of JSON objects and each JSON object in this array stores the source and target sentences as strings together with a “mappings” array that stores token mappings.

Each mapping is a JSON object and all the mappings associated with a single sentence pair are stored in the mappings array. Each object in “mappings” array has two elements: “A” is the array tokens from source text, and “B” is array the tokens coming from target text. For compatibility with frontend, we store tokens as start-end index pairs in the sentences. Each token is a pair of integers (start, end).

2.3.2. Mapper Serialized Data

The mapper package keeps at its disposal a hypergraph module that it uses to map tokens in one sentence to tokens in another sentence. The hypergraph is created from the mappings that are stored in the PostgreSQL database. The hypergraph is loaded in memory. To avoid excessive network communication with the database, the hypergraph is stored in a serialized form on disk. Python serialization libraries will be used for this purpose.

As we will discuss in the algorithm, we use persistent data storage to update graph models, but it is a good practice to keep a copy of the most recent hypergraph for each language pairs. Therefore, we regularly store hypergraph models, offline. In order to do so, python provides a serialization library called “cPickle”, implemented in C for efficiency, that helps serialize and save any type of python variable, data type and object instance. We serialize hypergraphs, together with a timestamp and save them into a file with the name

“graphID_timestamp”. This helps a quick start up of the python servers, without the need to iterate over project files and build the system from scratch each time the system goes down.

3. Algorithms

3.1. Sentencer

Sentencer is algorithm that takes as input two texts (source and target texts), and their designated language identifiers. It outputs a list of pairs of sentences that correspond to each other in the two texts. Sentencer executes a context-aware procedure to define sentence pairs across languages.

3.1.1. Paragraph Splitting

The first step in sentencer is to break down the text into paragraphs. The assumption is that any given text pair have equal number of paragraphs. Each paragraph pair will be given to the next step iteratively, as independent input.

3.1.2. Phrasor

Second step is breaking down the paragraphs into sub-sentence units (phrase-like). This step involves phrasor, an automatic procedure for producing phrases-like units from a given text. Phrasor has two separate procedure depending on whether the text is English or Turkish:

- English: Use as delimiters standard sentence ending punctuation marks such as full stop, question mark, and exclamation point.
- Turkish: Use as delimiters not only sentence ending punctuation marks such as full stop, question mark, and exclamation point, but also conjunctions such as “ve”, “ama”, and “veya”.

This should result in more finely chopped phrases in Turkish than in English. These two phrase lists will be passed to sentence mapping procedure.

3.1.3. Sentence Mapping

Sentence mapping is the core of the algorithm. This procedure takes in the pair of lists of phrases outputted by phrasor. The algorithm then will create a set associated with each phrase. The sets will be as follows:

- For each English phrase, take each space delimited word and get a list of the top 5 Turkish translations for each word. Then take the top 5 translations of each of these Turkish words in English. This final set of English words will be the set associated with this phrase.
- For each Turkish phrase, take each space delimited word and get a list of the top 5 English translations for each word. This final set of English words will be the set associated with this phrase.

Now the similarity score of pair of phrases is defined as the Jaccard similarity between their corresponding sets. Jaccard similarity is defined as follows:

$$sim(P_1, P_2) = \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|}$$

Where P_1 is the set associated with the source phrase, and P_1 is the set associated with the target phrase.

The algorithm runs by breaking each of the two paragraphs into two parts. The similarity score of the two first parts is calculated, and so for the two second parts. The algorithm tries every possible breaking points for this paragraph division, and finds the pair of breakpoints that maximizes the sum of the two similarity scores. This pair of breakpoints is fixed, and the pair of first parts of texts is now treated as an independent paragraph and so for the second pair of parts of texts. Both of the new paragraphs are processed with the same procedure we just mentioned until on the paragraphs has only a single phrase.

The resulting output will be a list of pairs of sentences that consist of one or more contiguous phrases. The algorithm will use lazy evaluation where each similarity score that was once computed will be saved. Each step in the algorithm will take order $O(n^2)$ time where n is the number of phrases in a paragraph (to simplify analysis we assume both paragraphs have the same number of phrases). Each new step will have half of its set similarity scores precomputed from the previous step. Each step also doubles the number of problems. Together, these two maintains the computation per step at $O(n^2)$. Since we stop when a current paragraph has only a single phrase, this means that we will have at most n steps in the algorithm. This makes the overall run time $O(n^3)$.

3.2. Token Mapper

3.2.1. Hypergraph

In the most basic sense, our automatic mapper system is built upon a graph model that identifies association between tokens across languages. Specifically, we have a central Mapper unit that stores and maintains bipartite hypergraph models for capturing structural relations across languages.

Hypergraphs are a particular type of graph that help associate more than 2 nodes together. Hypergraphs capture complicated relationships between nodes and allow for development of more complex graph models. As another important aspect of our mapper model, bipartite graphs are a special instance of regular graphs in which there are two sets of nodes that are disjoint. Edges are defined between nodes of different sets. Nodes in the same sets do not have any edge connecting them.

A bipartite hypergraph, as we name it, combines both of these concepts together to serve our specific purposes. To define in formal terms, let $H = (V, E)$ denote an hypergraph instance, where V is a set of two disjoint sets, and E is the set of all hyperedges. We can further define $V = \{A, B\}$ such that A and B are two sets of disjoint nodes. An hyperedge, e can be defined as $e = \{a, b\}$ where $a \subseteq A$ and $b \subseteq B$.

We define a bipartite hypergraph instance for any language pair we have in our system. So far, we limited it to 3 languages, but it could be extend to any number of languages. When users save manual/automatic mappings in one of their projects to the system, they are stored in our Django database servers. With regular offline updates to our Mapper model, new mappings are introduced to corresponding hypergraph instances. We define a mapping to be a *pair of sets*, where sets are groups of tokens of different languages that are associated with each other. We can explain how we make use of these graphs in our system through an example. Let's assume we have several user translation projects that includes English-to-Turkish or Turkish-to-English translations. Imagine the following sentence pair: "Turk askerler alanda yuruyorlar - Turkish soldiers are marching in the field". Here is a possible complete set of mappings: {({Turk}, {Turkish}), ({askerler}, {soldiers}), ({alanda}, {in, the, field}), ({yuruyorlar}, {are, marching})}. The graph building algorithm works in the following way:

- Mapping pairs are read and all of the tokens in both languages are added to respective node set of the languages, if they do not already exist
- An hyperedge instance is created by associating the tokens in source text set to tokens in target text.
- If this hyperedge does not exist, it is added to the system.
- Repeat the procedure for all project files in the system, update all graphs accordingly.

3.2.2. Tokenizing

Automatic tokenization is another feature of our system, that aims to help language learners and translators to analyze given texts. Similar to hypergraph model instances, we offer tokenization for only two languages: English and Turkish. For English tokenization, we make use of a very popular library, NLTK, that offers a reliable tokenizer for English language. The tokenizer identifies individual words and also breaks certain types of tokens down into parts further. For instance, an occurrence of the word “didn’t” should be analyzed as “did” and “n’t”, which is handled by NLTK English tokenizer.

For any other language, our idea is to use a regular expression based tokenizer. We developed one lightweight tokenizer for Turkish language that is based on whitespaces, punctuations and particular marking characters/tokens. Turkish tokenizer helps us identify individual words, and occasionally suffixes. This tokenization approach is specifically designed for automatizing the whole process of breaking translated text pairs into sentences, tokens and mappings. We, the developers of the system, are aware that languages exhibit variable grammatical rules, sentence structures and alphabets, but whitespace based tokenization breaks sentences into smaller yet meaningful chunks independent of the alphabet and sentence structures. We believe that this type of tokenization is subject to incremental improvement.

3.2.3. Set Cover

As we discussed in this section, our mapper model relies on bipartite hypergraphs that store associations between token sets in different languages. We devised an algorithms for automatic token mapping between two sentences. We specifically operate tokenization in sentence level because text-wide mapping becomes a problems that requires contextual analysis. We propose a context-agnostic mapper algorithm that is designed to perform on results of our sentencer algorithm. Given two lists of tokens, in which tokens are ordered as they appear in the sentence, one for source and one for target sentence:

- Define $list\_S$ and $list\_T$ as the tokens in source and target sentences, respectively, $G.source$ and $G.target$ as set of all tokens for source and target languages in hypergraph G, $sent\_S$ and $sent\_T$ as paired sentences from source and target language respectively.
- For each unique token T in $list\_S$ do:
    - Search for all hyperedges in which token T is involved, and add these hyperedges to the set $candidateHyperedges$
    - Repeat this for all such T
- For each hyperedge H in $candidateHyperedges$ do:
    - Let $H.source$ and $H.target$ be token sets coming from source and target languages, respectively, and .
    - If $H.source \subseteq G.source\ AND\ H.target \subseteq G.target$ then add hyperedge H into $existingH$
- For each hyperedge H in $existingH$ do:
    - Take a copy of *list_S* and *list_T*, and operate on the copies for each iteration
    - Find the first occurrences of each token in $H.source$ and $H.target$ in source and target sentences and mark those tokens as processed.
    - For each of the processed tokens, find their start and end indices in their respective sentences, and create a new hyperedge instance with indices of tokens replaced with string values of tokens, add it to $setcoverH$
    - For the same H repeat this procedure to see if there is another occurrence on the same set of tokens in the sentences.
    - Repeat until of hyperedges in $existingH$ are exhaustively processed.
- Apply greedy set cover algorithm for hyperedges in $setcoverH$
    - Replace each token in list_S and list_T with start-end indices of tokens in respective sentences as (start_ind, end_ind)
    - Merge token lists list_S and list_T.
    - Make each hyperedge in $setcoverH$ into a list of (start_ind, end_ind) pairs.
    - Sort hyperedge lists in decreasing order of list size
    - Our greedy choice of hyperedges is the number of tokens involved in an hyperedge. We try to cover the set of all tokens in merger of list_S and list_T by trying to select hyperedge with the largest number of tokens.
    - Greedily choose the hyperedge lists that covers the merged list of all tokens in the sentences pair

- Finally return mappings as the list of hyperedges that are selected to maximally cover the set of tokens in list_S and list_T.

4. Tools, Technologies, and Resources Utilized

4.1. React.js

We use React.js for the client-side application mainly because of its dynamic nature to quickly refresh the components on the website as the state of the program changes. React is a versatile user-interface library developed by Facebook and became widely popular in 2015. We use it to easily manipulate user input data and always be in touch with our server application for enhanced dynamics.

4.2. Django

Django is a high-level Python web framework that emphasized on pluggability and re-usability.. Django enables fast development, security and scalability. Django deploys MVC design pattern. It is a free and open source system.

4.3. PostgreSQL

PostgreSQL is an object-relational database with additional NoSQL features such as JSON and HSTORE key/value pairs. Postgres is ACID-compliant and transactional. It is a free and open source system.

4.4. Python

Python is used throughout the project's automatic services. Python allows for easy and smooth development since a lot of the data structures we have needed to use are already implemented and well optimized. This allows us to focus on improving our algorithm rather than spending time on code debugging.

4.5. Dictionary API

We use Çevir.com API dictionary service. The service returns JSON formatted objects of dictionary lookups. Çevir.com is used for our Turkish-English sentencer package.

4.6. JSON

JSON stands for JavaScript Object Notation. It is a language independent data format to interchange data among different platforms. JSON offers readability for humans. Also, parsing and generating JSON is easy.

4.7. Hypergraphs

Hypergraphs are special types of graphs where the edges are defined to connect multiple nodes at the same time. As far as we have researched, hypergraphs were introduced before mid 70's and they have many applications extending from bioinformatics research to parallel databases. We are specifically dealing with bipartite hypergraphs in order to be able to model token mappings across languages via hyperedges and disjoint set of nodes/tokens.

4.8. Token Mapper Algorithm (Set Cover)

The mapper algorithm was inspired by the set cover approximation approach that was utilized in genomic structural variation discovery. This approach was used by Hormozdiari et al (2010) to pick the most likely predicted structural variation events. Similarly, we use a variation of set cover approximation to tackle the choosing of the best token-to-token mappings.

4.9. Software Architecture

To construct and to document the high-level structures of Turjeeman, we searched for best practices in software architecture. These high-level structures are software elements, relations among them, and properties of both. Software architecture, in a sense, is considering the trade-offs for different choices of these structures before the implementation starts. Resources that cover these issues in a contemporary fashion and has benefited for our project's software architecture design choices.

4.9.1. Three-tier Architecture

This type of architecture is suitable for client-server type applications that has three separable physical layers that are presentation layer, data layer and application layer. Our project is a client-server type application that is following three-tier architecture for flexibility

and reusability of following layer abstractions: front-end web server as the presentation, application server as the application layer and database server as the data layer.

4.9.2. Model View Controller (MVC) Architecture

This type of architecture is suitable for user interface designs. Turjeeman is a web application that communicates with users through its web interface. MVC is out-of-box performed by Django framework. Thus, reusability among similar components and being able to perform development simultaneously are the reasons we adopted such architecture.

5. Impact of Engineering Solutions

5.1. Educationally

Turjeeman can be used as a great language learning practicing tool. When a user completes a mapping project, they are demonstrating their ability to comprehend the syntactic and semantic relations between two languages. This can be used as a way of for teachers to test their students language skills. When a language is related to another language like what we have in Turjeeman, the neglected syntactic features are exposed to light. What usually people take for granted, is now a main point of consideration, and this is the key to learning.

5.2. Societally

Turjeeman serves a good societal role as a linguistic bridge. Human language is ambiguous, and thus the best way to demonstrate the exact meaning of a sentence is by mapping it to a second language. Ambiguities in a human language are usually result of multi-functionality of strings in a given language. Thus, these possible ambiguities are mapped to a second language, the likelihood of having the same multi-functionality in the second language is reduced. This can even be extended to having more than two languages mapped onto each other, bringing the chances of ambiguity even further down.

5.3. Economically

Using Turjeeman as a language ambiguity mitigation tool has a great potential to be a way for formulating text documents that are understood intelligibly by all parties at stake. If a

document is written in two languages and mapped in a similar fashion what we have in Turjeeman, its meaning is sealed and made tight. The mapping constraint the chances for manipulative syntactic tricks that are possible in legal documents. An example of this is how a US based company lost millions due to the interpretation of an Oxford comma!

5.4. Environmentally

Our server application will be running on at least one server and the energy that is used by that server may slightly harm the environment. Moreover, any user who will use Turjeeman will visit the web application using their personal devices, which will also use energy to run. Although we currently have no estimate on how many people may come to use Turjeeman, we hope it will not be too many people that use Turjeeman and contribute to climate change in a political setting where incapable men serve as the leaders of the world's most powerful countries.

6. Related Contemporary Issues

6.1. Machine Translation

The advent of machine learning algorithms has made creating training datasets an essential goal for any contemporary machine translation project. Turjeeman allows for easy and facilitated creation of such dataset. On top of that, Turjeeman has the added benefit of creating rich translation corpus. This is because Turjeeman adds sentencing, tokenization, and token mapping on top of vanilla translation corpus that typically stores only source and target translation texts.

We think that given enough user-curated data, Turjeeman corpus can be used as a guiding skeleton for modern machine translation techniques such as neural network based machine translation. Tokenization and token mapping represent data-driven rule-based translation. This unique combination and blurring of the lines between probabilistic translation and rule-based translation is potentially a very powerful tool

6.2. Language Education Tool

Turjeeman's data can be used as practice example for people interested in language learning. The user created dataset can be easily turned into questions that a student can use

to practice their language learning skills. The student can try to map a given source and target sentences and see what the correct answer is.

6.3. Crowdsourcing

Turjeeman relies on the crowdsourced model of content creation. This model has proved its effectiveness and scalability in many different scenarios. Wikipedia, Foursquare, and even Nasa’s search for exoplanets are heavily reliant on crowdsourcing. We see that Turjeeman would be another example on this list.

7. User Manual

7.1. Installation

7.1.1 Server Side

The following GitHub repositories need to be cloned to their respective servers. Then, the README files has exact instructions on installing the package included in them. Since our repositories have different design structures and distinct requirements, we have included unique installation guides for each of them in our GitHub repositories.

- https://github.com/baraaorabi/SentenceMapper
- https://github.com/baraaorabi/Sentencer
- https://github.com/yildizm/turjeeman

7.1.2. Client Side

The repository for the client side application lies at https://github.com/cagdass/turjeeman-client. The client side needs only a modern web browser, preferably Chrome. For locally hosted servers, the user can access the GUI through localhost:8000.

7.2. Usage

7.2.1. Logging In

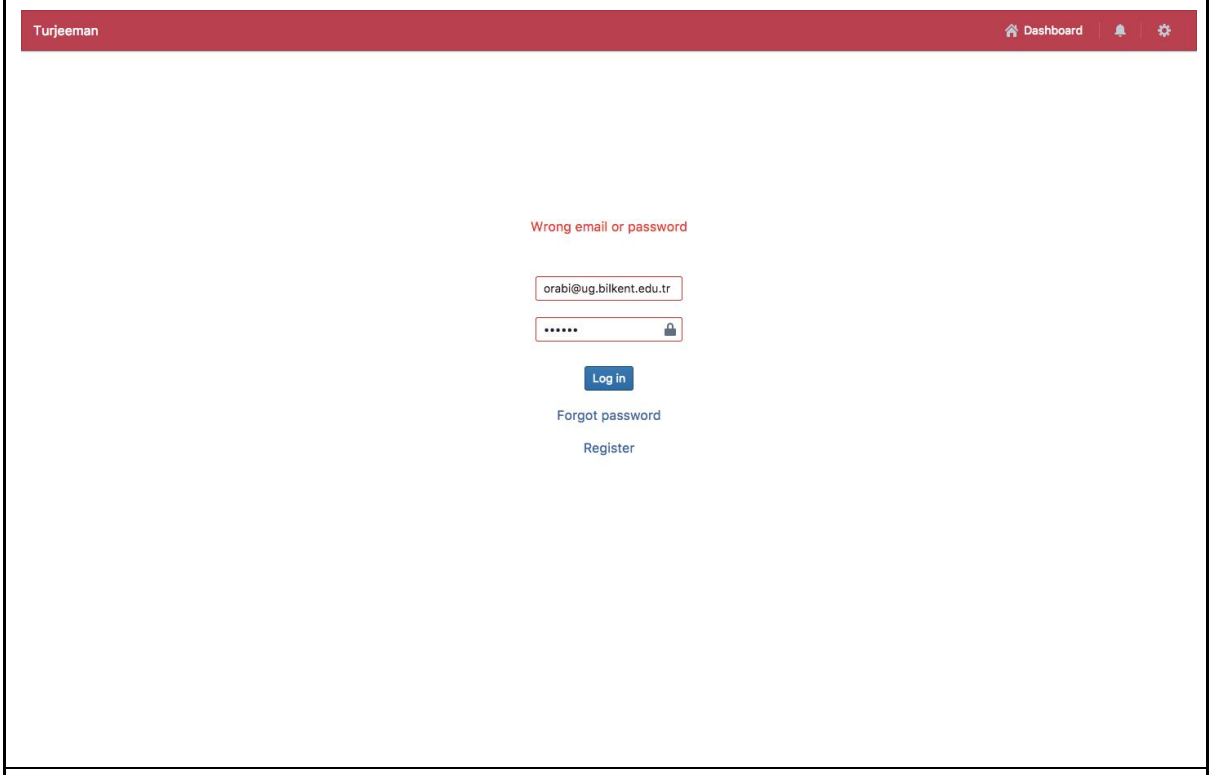

Figure 7.2.1.a A fallible attempt to log in with wrong credentials

The login page consists of input forms where the user provides their credentials, email and password. There is also an elegant button when it's clicked upon the credentials are sent to the server application for authentication. There are also two links to other pages, Forgot Password and Register where the user can respectively initiate action to recover a forgotten password or register the application.

7.2.2. Loading Project

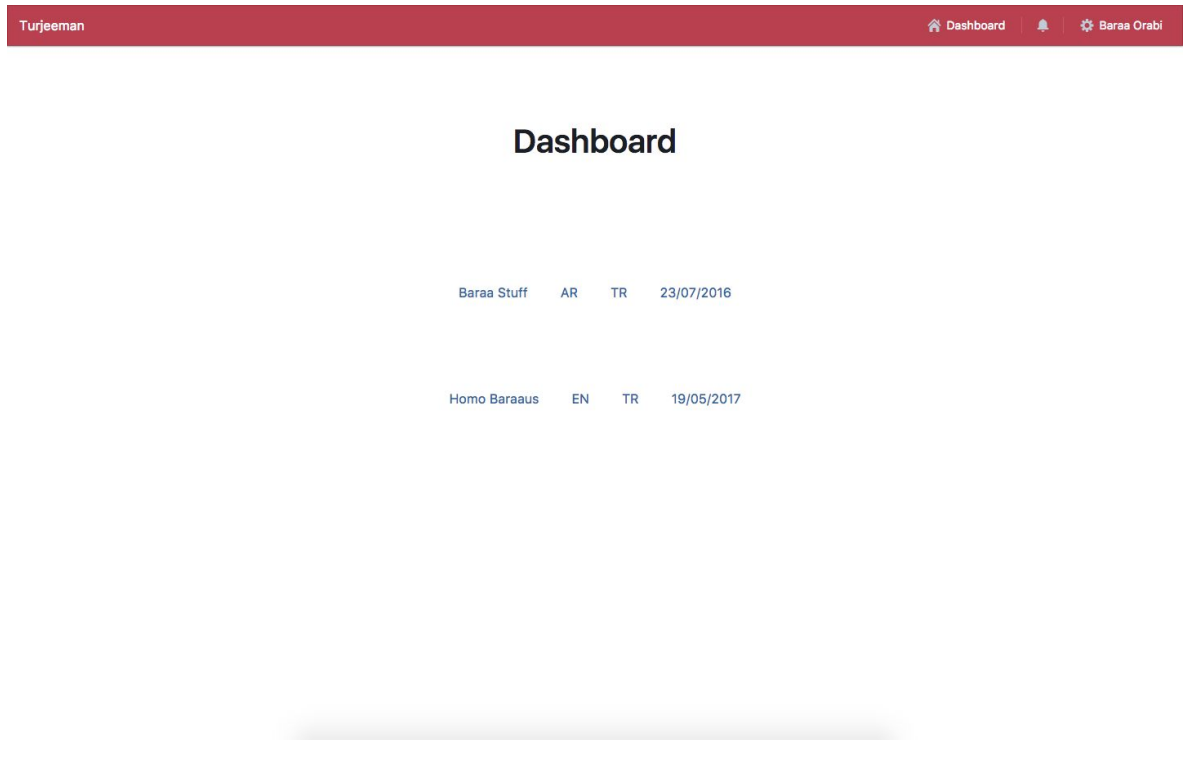

Figure 7.2.2.a Dashboard of a user on which the user's projects are displayed.

The dashboard will list a user's projects with some metadata, which being the title of the project, the source language, the target language, and the date that it was last saved. The current state of the dashboard is far from complete in terms of styling, however its function is to display each project separately in a UI component, which will redirect the user to that project's page when they are clicked.

7.2.3. Editing Project Detail

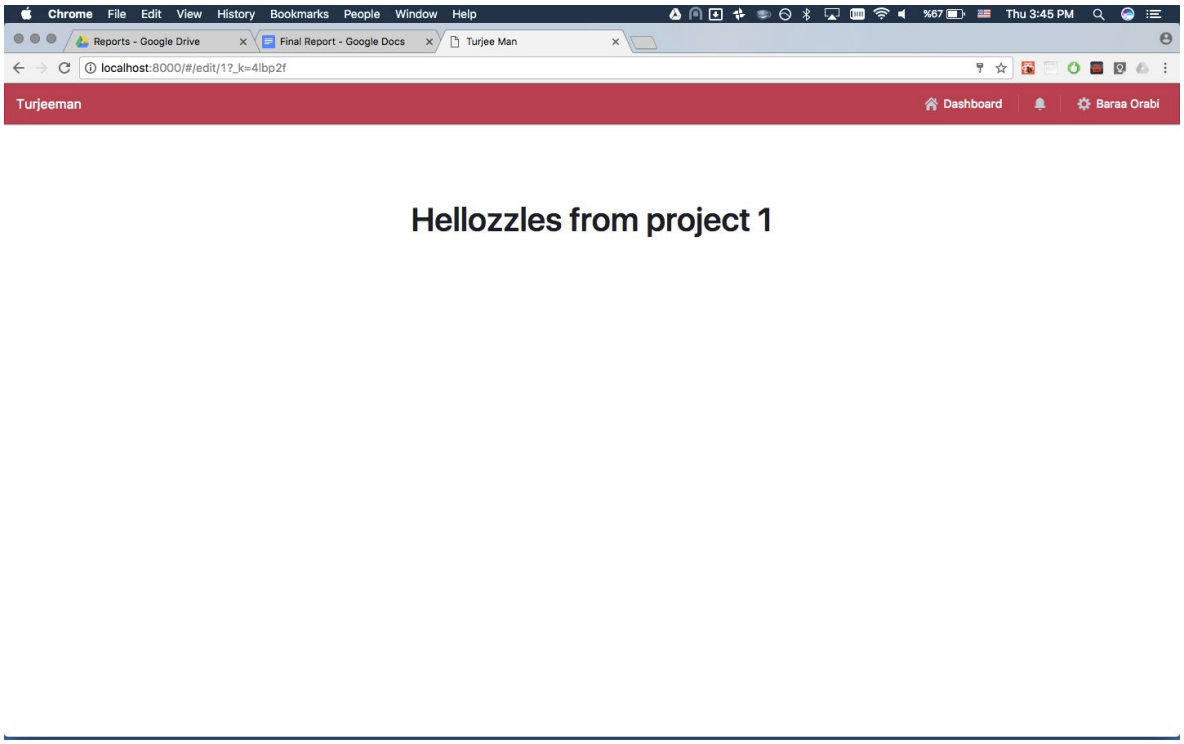

Figure 7.2.3.a A tentative screenshot from the project edit page.

The edit view consists of two text input forms where the user can edit the texts of the two languages and then talk to the server application to save the information. One important aspect of this view is that a user will have to be logged in to see this page or they will be redirected to the login page, and this page takes in the project ID number as the URL parameter when it is routed and the client application will cross-check with the server application to see if the user has permissions to display and edit this project before the project is retrieved.

7.2.4. Sentence-to-Sentence Mapping

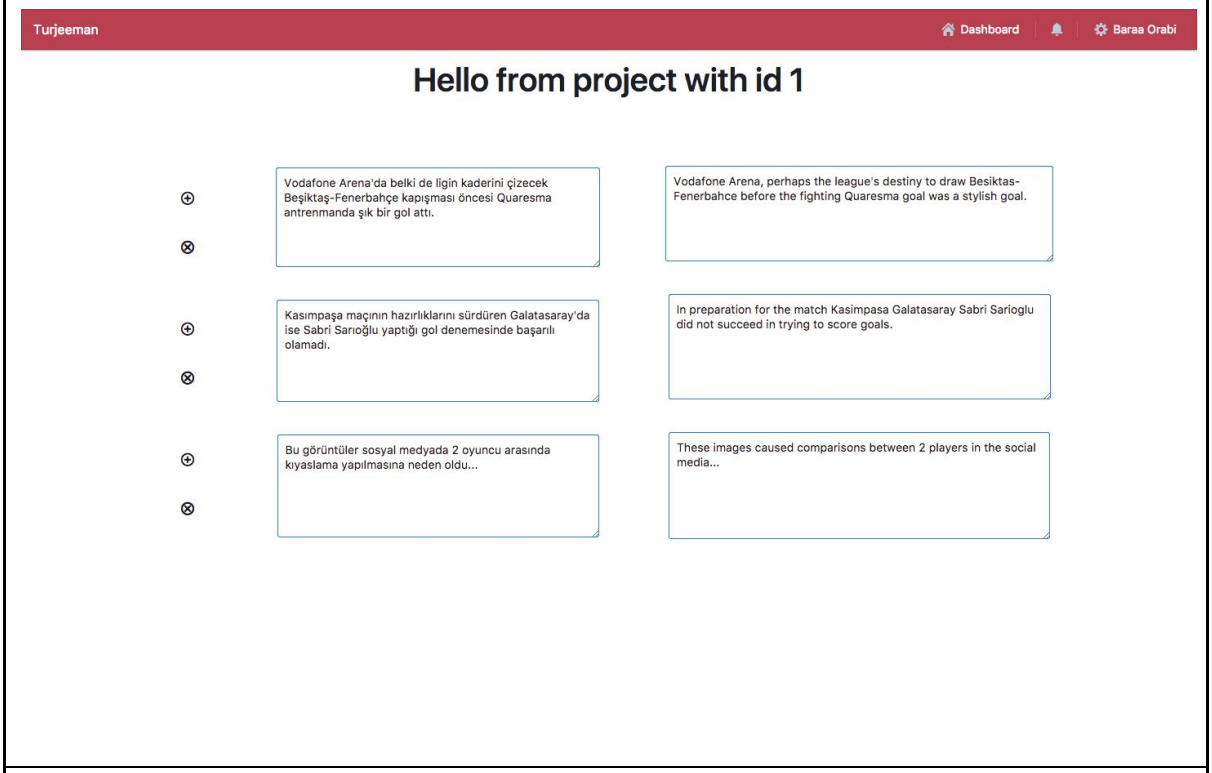

Figure 7.2.3.a A tentative screenshot from sentence-to-sentence mapping page.

In this view, the user is presented with sentences from the source and the target languages that correspond to each other. The sentencing of the texts in the source and the target texts are done by our software and the user is given options to merge contiguous sentences, or break them down before they begin token-to-token mapping.

7.2.5. Token-to-Token Mapping

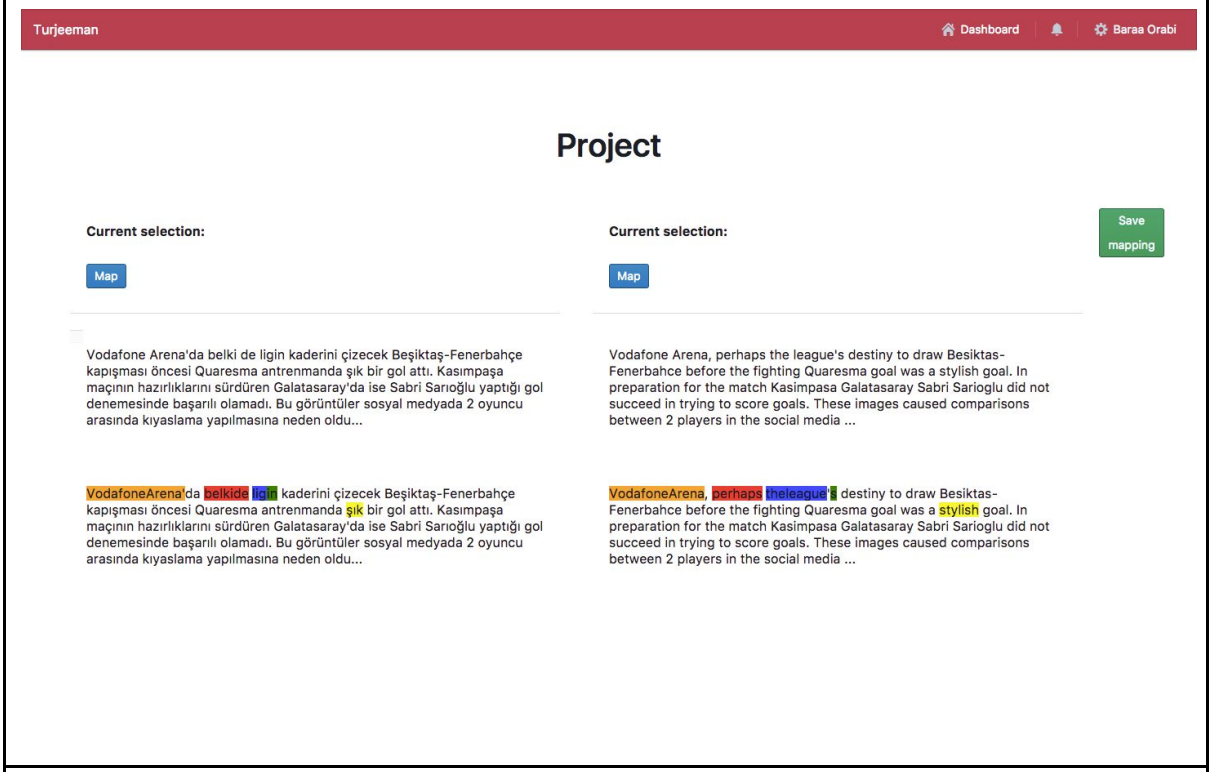

Figure 7.2.5.a A screenshot from the token-to-token mapping page.

The user selects matching tokens in the corresponding sentences in this view. The selected tokens are highlighted with predefined colors. Corresponding tokens between the languages are highlighted with the same color. The user can map multiple tokens to multiple tokens.

8. Resources & References

8.1. Literature

- Clements, Paul, et al. *Documenting software architectures: views and beyond*. Pearson Education, 2002.
- Gamma, Erich. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.
- Rajaraman, A., & Ullman, J. D. (2012). *Mining of massive datasets*. New York (N.Y.): Cambridge University Press.
- Hormozdiari, Fereydoun, et al. "Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes." Genome research 19.7 (2009): 1270-1278.

8.2. Online

- "Cross-Site Request Forgery (CSRF)", *Owasp.org*, 2017. [Online]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).
- "JSON", *Json.org*, 2017. [Online]. Available: http://www.json.org/.
- "PostgreSQL", *Postgresql.org*, 2017. [Online]. Available: https://www.postgresql.org/.
- "Django Web framework", *Djangoproject.com*, 2017. [Online]. Available: https://www.djangoproject.com/.
- Victor, D. (2017, March 16). Lack of Oxford Comma Could Cost Maine Company Millions in Overtime Dispute. Retrieved May 04, 2017, from https://www.nytimes.com/2017/03/16/us/oxford-comma-lawsuit.html?_r=0
- Rossignol, D. (2017, February 16). NASA is crowdsourcing the search for exoplanets. Retrieved May 04, 2017, from https://www.engadget.com/2017/02/15/nasa-is-crowdsourcing-the-search-for-exoplanets/
- Hypergraphs -- from Wolfram MathWorld. http://mathworld.wolfram.com/Hypergraph.html