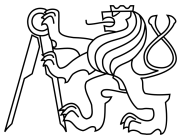
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



# ASSIGNMENT OF BACHELOR'S THESIS

| | |
|---|---|
| **Title:** | Web service for advanced text analysis |
| **Student:** | Jan Švejda |
| **Supervisor:** | Ing. Pavel Kordík, Ph.D. |
| **Study Programme:** | Informatics |
| **Study Branch:** | Software Engineering |
| **Department:** | Department of Software Engineering |
| **Validity:** | Until the end of summer semester 2017/18 |

## Instructions

Survey modern method for text analysis and representation. Focus on machine learning approaches including recurrent neural networks and character-level text processing. Design and implement a web service providing language independent advanced text processing functionalities, such as similarity of headlines (sentences), sentence embedding, predictions of popularity or ephemerality. The main aim of the thesis is to design the service itself, it is supposed to utilize existing implementations of particular methods. Test the service on real datasets and discuss your results.

## References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 22, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

# Web service for advanced text analysis

***Jan Švejda***

Supervisor: Ing. Pavel Kordík, Ph. D.

11th May 2017

# Acknowledgements

I would like to express my gratitude to the supervisor of this thesis, Ing. Pavel Kordík, Ph. D, who provided a lot of support and expertise in the course of this work. My thanks belongs also to the faculty, Ing. Milan Václavík and doc. Ing. Ivan Šimeček, Ph.D. for letting me do some of the heavy computing on university's GPU cluster. Last but not least, I would like to thank Ing. Tomáš Řehořek for helping me with obtaining some of the data used in this thesis.

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 11th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology


### Citation of this thesis

Švejda, Jan. *Web service for advanced text analysis.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

# Abstrakt

Mnoho statistických modelů a modelů strojového učení, které se snaží pokořit problémy zpracování přirozeného jazyka, mají potenciál asistovat lidem v mnoha oborech. V rámci této práce je vytvořena aplikace, která zpřístupňuje takové modely aplikované na doménu článků z internetu prostřednictvím webové služby a webové stránky. Tím je novinářům a editorům článků poskytnuta možnost získat více informací o jejich článku, umožňuje to integraci těchto modelů s externími systémy a nabízí to interaktivní experimentování s nimi také lidem, kteří se o zpracování přirozeného jazyka zajímají. Jak webová služba tak stránka byly úspěšně navrhnuty a implementovány, a to s důrazem na bezpečnost a škálovatelnost. Aplikace je navržena takovým způsobem, aby bylo možné ji v budoucnu snadno rozšířit o novou funkcionalitu.

**Klíčová slova** články z internetu, webová služba, zpracování přirozeného jazyka, strojové učení

# Abstract

Many statistical and machine learning models, which tackle the problems of natural language processing, have the potential to assist humans in various domains. In this thesis, an application is created, that gives access to these

models applied to the field of Internet news through a web service and a website. This provides writers and editors of articles with more information, makes integration of such models with other systems possible and allows people interested in natural language processing to interactively experiment with them and learn about them, too. Both the web service and the website were successfully designed and implemented with security and scalability in mind. The application is designed in such a way, that extending it with new functionalities in the future will be easy.

**Keywords** Internet news, web service, natural language processing, machine learning

# Contents

# List of Figures

# List of Tables

# Introduction

Modern on-line newspapers often rely on the number of visitors for their revenue. Recommendation systems are used extensively so as to show interesting content to the user. By doing that, more views are generated in many ways. In the short term – during one session – the user gets offered more of the content he is interested in at the moment. In the long term, by making the user pleased with the content, he comes to like the website more, visits it more frequently and stays longer. That said, recommending makes sense, generates profits for the company and the user benefits from more (ideally) interesting and relevant content. Furthermore, news published on the Internet are more and more often becoming a substitute for newspaper, which creates an opportunity to be taken advantage of.

This instigated an idea of delivering relevant and useful information not only to the reader, but also to people creating the content (in this case specifically to editors of Internet news) through the means of a web service, that would provide textual analysis capabilities on news articles' titles. By using established statistical and machine learning methods, the intended web service will, for example, enable the editors to adjust the title, allow them to make better decisions on how to create content or ease production by giving access to natural language processing algorithms and therefore increase journal's revenue and number of readers.

In general, web services are characterised by their interoperability, ease of integration, scalability and platform independence. As a result, they make a suitable choice for opening an interface to a company's service or individual modules of a larger software. A web service is, in fact, a client-server application communicating over a standard protocol, published and accessed most frequently via the Internet. Two major models of web services exist – Representational State Transfer (REST) and SOAP, both will be discussed and considered in the first chapter.

Firstly, the thesis aims to design and implement the web service itself, keeping in mind important aspects like extendibility, scalability and security

as well as considering usability. Extendibility at that should be emphasized, because its functionalities will likely be broadened in the future. To that end, the Application Programming Interface (API) of the web service ought to be documented alongside the most important parts of the service. In addition, the web service's main API points must be tested to ascertain its capability of operation.

Second goal of the thesis is to make a website Graphical User Interface (GUI) for the users, so that they get the possibility to try out some of the functionalities. This would serve as sort of an informational portal about the used algorithms and models as well.

The web service itself was required to provide advanced text analysis functionalities by allowing remote access to core modules, however, no specific implementation of all the desired analytical functionalities of the system had existed. On the grounds of that, in the first chapter a short survey of various statistical and machine learning models is carried out, including, but not limited to, character-level processing models. Third goal is to afterwards integrate existing solutions, in the form of frameworks, implemented models or widely spread methods, into the web service. These then will constitute its functionalities.

The thesis stresses out the integration of various models and functionalities into the web service, to allow the developer to work with models' multiple versions and to facilitate the creation of new ones. It also had to be taken into consideration, that when possible, the models should be designed as language independent. That means, that merely by swapping data used for training it, the model will be able to adapt to it, whether the data is in English, German, Spanish, Japanese or Czech. Primarily though, the application will stick to Czech and in some cases show on an English dataset, that the models are capable of being language independent.

Present-day development in the intriguing field of Natural Language Processing (NLP) inspired the theme of this thesis. The research in this field is buoyant and keeps pushing the boundaries of human (machine) knowledge further. New fascinating possibilities of making computers grasp the notion of human language arise every day. For someone interested in this field, trying out some of these algorithms is surely worthwhile. And even if the web service might not exactly be what media companies would pay for immediately, by incrementally improving it and widening its capabilities, it will become all the more useful in the future and maybe commercially appealing.

Chapter 1

# State-of-the-art

## 1.1 Web services

A web service (WS) is, basically, an application that can be accessed via a network, not necessarily the Internet. It can be considered as sort of a standardization layer between application code and application clients. This creates a big advantage – web services allow platform independence when integrating different programs or parts of more complicated applications, because a program can use a WS without any need for compliance with a specific platform where the service is running [1].

From the bottom to the top, web services can be divided into 4 essential layers. Communication, packaging, description and discovery. Communication layer is responsible for transporting messages with technologies like Transmission Control Protocol (TCP) or Hypertext Transfer Protocol (HTTP), packaging layer defines the way data are structured, which is most frequently some type of Extensible Markup Language (XML) or JavaScript Object Notation (JSON), and lastly description and discovery layers that make using and finding the service possible [2].

There are two types of designs of web services that are most used these days, both of which are considered. Following is a summary of them.

### 1.1.1 SOAP

SOAP is very often used as a packaging layer in web services. As stated in [3] SOAP is "*a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment*" [3]. The underlying language of SOAP messages is XML, which gives it wide structural possibilities. Most frequently, SOAP takes advantage of HTTP or Simple Mail Transfer Protocol (SMTP) to transmit messages, however, it can be bound with almost any communication protocol. SOAP interfaces are described using Web Service Description Language (WDSL), which establishes a contract between the

client and server. The biggest advantages of SOAP are, that it can provide enterprise-level security, atomic transactions or built-in error handling. What is more, thanks to various extensions, it has the possibility to add new functionality quite easily [1].

A SOAP message consists of two basic parts both wrapped in a SOAP "envelope". An optional SOAP header and required SOAP body. The header blocks define meta-data which specify how the message should be handled, for example the destination, expiration of the message, authentication information and so on. Then, body is the actual content of the message. That means data, return values, parameters and/or other information expressible in XML format [1].

When talking about SOAP web services, it is understood that the packaging layer of the WS uses SOAP as the protocol. Nonetheless, SOAP by itself is not a WS. It can be used anywhere else as a communication protocol.

SOAP is the platform of choice for large, corporate information systems, where integration must be rigid and compliant. It is highly standardized and therefore becomes inherently more complex and expertise demanding. On the other hand, it offers extensive customization. By not depending on a specific transport layer, it achieves wide re-usability of code and the possibility to simply change the web service, say from a HTTP based to a SMTP based service, or allow various endpoints to interact via different protocols.

### 1.1.2 Representational State Transfer (REST)

REST was first proposed by Roy Thomas Fielding in the year 2000 as part of his dissertation thesis and became widely popular on the Internet for its ease of use. The entire idea is based on HTTP and its four methods GET, POST, PUT and DELETE. As a consequence to this, REST is limited by its use solely with HTTP, which is a trade-off for simplicity [4].

The goal of REST is to utilize the already well established, tested and general-purpose HTTP and simplify web services. It was a reaction to the ever more complex SOAP, which basically consists of a large number of standards, that not rarely get implemented in various ways by the software providers mainly due to dubious and ambiguous definitions. That is a major weakness of SOAP. Different implementations did not necessarily conform with each other sometimes. Such behaviour heavily undermines the principles of web services – implementation independence – which lead to the creation of REST [5].

Even though, some aspects are difficult to directly compare with SOAP, for REST is not a protocol, but an architectural style, similar functionality can be achieved with both REST and SOAP web services alike. That is why both approaches are considered.

As opposed to SOAP, REST is not restricted to the use of XML. In theory REST can take advantage of any kind of format for packaging, although standardized formats like JSON are preferred. These allow data objects to be

seamlessly transformed from in-memory representation to formatted text at one point, and transformed back from text to data objects at another. Though this is not restricted only to REST, but is an aspect that both of the styles share, REST has the upper hand in this regard, for it can produce responses in multiple formats and the client may choose whichever one he/she prefers from the supported formats.

REST has its own specifics as well as SOAP. Everything accessed via a REST service is called a resource. This can be anything ranging from images, structured data, files to HyperText Markup Language (HTML) pages. Each resource has its own Uniform Resource Identifier (URI), by which they are identified and located. The messages between peers are created as HTTP requests with data and parameters (if any) in their body and in the URI. Requests can be of 4[1] types as mentioned before and should adhere to particular rules [6]:

1. GET – serves to retrieve a resource. Is safe. Safety means, that no matter how many times (even never) a GET request is sent, it should not affect any resources at all. Nothing changes when calling it, so nothing can go wrong.

2. PUT – is used mainly for updating existing resources. It is idempotent, which signifies, that repeating the request will not cause any harm. Once the resource has been updated, updating it again is okay[2].

3. DELETE – a request for deleting resources. Is also idempotent, because deleting an already non-existent resource does not do anything.

4. POST – used for creating resources. It is the only request, that if repeated can cause undesired results, so it is neither safe nor idempotent.

An important characteristic of REST is indeed the fact, that communicating sides do not have to remember any states, because everything needed to process a request ought to be present in the request itself (or its URI). Such behaviour particularly reduces complexity, which again encourages software quality, maintainability and scalability [5].

REST is usually faster and takes less bandwidth than SOAP, because it employs the less verbose JSON (as opposed to XML in SOAP). On top of that, SOAP messages need to relay more meta-data. Also, REST is regarded as a more loosely coupled architecture, because it uses a URI to access and expose application logic. From this benefits mostly the client, because he does not need to comply with the entire interface specification unlike SOAP web services [6].

[1] Or more if HEAD and some other methods are also taken into account, but these requests are not as useful in the context of REST and therefore will be omitted.

[2] GET is also idempotent as idempotence is a weaker assumption than safety.

## 1.2 Text analysis survey

There are various state-of-the-art approaches to textual analysis, each with a bit different aim and purpose. These include statistical language models like $n$-gram models, frequency models like Term frequency-inverse document frequency (Tf-idf) or continuous space language models, where neural networks are often used. Therefore, methods, that could have been applied in this work, had to be evaluated to ascertain, which might work best.

An important factor when choosing appropriate models was, whether they are (or not) language independent. This property requires the model to abstract from a particular language in question. This means, that by only changing the language of data provided for the creation of a model, one should be able to reach comparable results.

Emphasis was put on methods that at least partly had been implemented, were very well described or could be implemented with the help of existing frameworks. This is due to the fact, that the thesis's aim is to utilize existing text and language analysis possibilities, adjust them if need be, and apply them to the case of article titles.

For this thesis, a dataset in Czech of about 335 thousand article titles and a dataset in English of about 423 thousand article titles was made available to work with. In order to choose the most suitable method, some particularities of this work's domain and available datasets needed to be taken into account:

- The primary dataset is in Czech. Contrary to English, Czech has many word forms – conjugations, declinations, irregular plurals and so on. On top of that, sentence structure is looser and more varied than in English. The added complexity of the language makes Czech rather difficult to process by a machine.
- Because the analyzed texts are actually titles of on-line newspapers, they usually do not exceed more than about 120 characters, averaging at about 60–70. In principle, it is just a short phrase or sentence that summarizes the newspaper article. As a consequence to that, the text incorporates a lot of information in just a few words, but can also omit language structure, which would otherwise be present in a common sentence.
- A lot of research in text analysis is done only in English, so methods that work well on English, might not be as successful with the Czech language.
- In some topics, the variability of vocabulary and sentence structure is small and in other large. To explain this, tennis or hockey, for instance, are covered very often and with repeating words, names of people etc. On the other hand, topics like mathematics or physics seldom come up in ordinary Internet news and the keywords used are not very usual.

## 1.3 Statistical models

Statistical models rely on well founded mathematical grounds, which makes interpretations and measuring easier. Their methods are concise and accurate in what they do. A number of them were considered and their main concepts are described here.

### 1.3.1 $N$-gram model

In the task of predicting the next word in a sentence, it can be viewed as a probability of a word $w_n$, where $n > 1$, in a sequence of words under the condition of previous words. (For $n = 1$ the function is simplified to the probability of a sentence beginning with the word $w_1$.) That can be expressed with a probability function $P$ as follows:

$$P\left(w_n | w_1, \ldots, w_{n-1}\right)$$

If such a function had been computed on text samples, e.g. sentences, it would be able to effectively predict the next word. However, even with large corpora of texts, it is virtually impossible to have the probability computed on every thinkable sentence in the world. Language is superbly varied and complex, theoretically infinite. So in reality, there would be too many cases of sentences and words not observed in a dataset (as large as it may be, it will never be infinite) and so probability of some words being the next word, would be simply zero [7].

This needs to be improved. The way to do that is by grouping the observations, taking similarity into account so to say. This can be done by making an assumption, that the word being predicted, depends solely on a few preceding words, not the sequence before it. This essentially means to put words into equivalence classes, where each class contains words, which are preceded by the same window of words – the window's length is defined as $n - 1$. This is called an $n$-gram[3] model or also an $(n-1)$th order Markov model [7].

### 1.3.2 Tf-idf weighting

Tf-idf weighting is a way of looking at similarity between two documents or sentences. To calculate similarity, it employs a combination of so called *term frequency* and *inverse document frequency*:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Here $\text{tf-idf}_{t,d}$ is the weight of a term[4] $t$ in a document $d$. Then $\text{tf}_{t,d}$ is the term frequency of term $t$ in document $d$, in other words, number of occurrences of

[3] For $n$ equal to 2 a bigram, equal to 3 a trigram, afterwards simply four-gram etc.

[4] In the case of article titles a word.

some word in an article title. Finally, $\text{idf}_t$ is the inverse document frequency of term $t$ [8].

Idf is a way of discriminating words that appear too often and as a result do not say much about the relevance of documents, e.g. pronouns, connectors and so on. It is defined by:

$$\text{idf}_t = \log\left(\frac{N}{\text{df}_t}\right),$$

where $N$ is the total number of documents and $\text{df}_t$ is the number of documents where term $t$ occurs as well, also called *document frequency*. Idf is low for common terms found in many documents and high for more extraordinary terms [8].

The score defining how similar a document $q$ is to a document $d$ is then a sum of Tf-idf weights of all the terms in $q$. Or mathematically put [8]:

$$S(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}$$

### 1.3.3 Latent Semantic Indexing

Latent Semantic Indexing (LSI), or also called Latent Semantic Analysis, is a method for understanding language contexts and similarity of words or sections in large textual corpora. This technique creates a matrix of high dimensions, which represents words and word sequences. Afterwards, a decomposition technique, called Singular Value Decomposition (SVD), is applied to this matrix. It has been found, that many characteristics of Latent Semantic Indexing (LSI) resemble human perception of language, probably due to the fact, that it tries to find relations between when a word occurs and where [9].

In more detail, LSI transformation starts, first of all, with a matrix of unique words as rows and word sequences as columns constructed from the input text. The matrix's values stand for occurrences of a word in the sequence of words. Then, the matrix is decomposed with SVD into three matrices, whose product equals the initial matrix. Lastly, similarity of words is obtained with cosine similarity of vectors, formed by one of the three matrices' rows of given words [9]. Cosine similarity is defined as the cosine angle of two vectors $\vec{u}$ and $\vec{v}$ of length $n$:

$$\cos(\vec{u},\vec{v}) = \frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n}(u_i)^2}\sqrt{\sum_{i=1}^{n}(v_i)^2}}$$

In this work, LSI is considered for topic extraction from an article headlines dataset. LSI allows to find words, that contribute to a certain topic, but in general can do much more. To name but a few – text summarization, spam filtering, essay scoring, information discovery and retrieval.

## 1.4 Character-level models

Up to now, the described models were all focused on word-level processing. However, for some languages (Czech, for example), the need for more training data is increased, because the model has to be aware of all the word forms, that have almost the same meaning. Another thing is, that word-level models generally keep a known vocabulary of words, which is limited. Therefore, unique or rare words sometimes are not taken into account at all. With flowery languages, the vocabulary size rises quickly.

Hence, a different approach to the language problem was also considered, that is – processing the text character by character. This way, the purpose of vocabulary seizes to exist, instead an alphabet is used. As a result, the benefit of character-level models is the fact, that they are not limited in vocabulary size and thus better generalize the language.

The problem of some languages having a lot of word forms should potentially be eliminated as well. Mainly due to the partial similarity of the word forms, which can be understood, if the word is not taken like a single unit.

Most promising models for character-level models are from the field of machine learning, and especially that of deep learning.

## 1.5 Machine Learning

The research of Machine Learning (ML) has been steadily growing in the past years with diverse applications in all kinds of fields. ML methods are now commonplace so it often happens, that one does not even realize usage of functionalities powered by such algorithms. In the past few years, ML has been gaining momentum in automotive industry, search engines, speech recognition, image captioning, language translation, robotics, optimization, games and many more.

The capabilities of ML are taken advantage of mostly when dealing with problems, that are too complicated to program, or with tasks involving large amounts of data, that a single person could not hope to make sense of. Also a key feature of such programs is their adaptiveness, which can be achieved by simply offering the ML new data, which then results in a change of their behaviour [10].

Generally, two main types of ML exist – supervised and unsupervised learning[5]. In the case of the former, the system is presented with inputs and correct outputs. It then tries to make the most out of it, as if learning by making mistakes. To illustrate, imagine an example of image classification of cats and dogs. The training set consists of images, which all have a label

[5]Sometimes combination of both is used called semi-supervised learning. There is also a type called reinforcement learning, where the program is supposed to predict more from given training data.

determining the depicted animal. During training, the system modifies itself (learns) to distinguish between cats and dogs by correcting its mistakes. When the system is finally served with an unknown picture, it can access knowledge attained in the training phase and decide, whether it is a dog or a cat in the given picture. It is similar to a teacher telling a student what is correct or not, therefore the name supervised [10].

Unsupervised learning, on the other hand, aims to extract interesting content from unlabelled data and, for instance, look for unusualness or anomalies, do clustering of data or also generate new samples. These types of algorithms are often difficult to evaluate, because it is hard to find an objective metric, that would distinguish good results from the bad ones [10].

In this work, upon recent successes of neural networks in countless fields of study, it has been proposed to try to employ deep learning methods in order to achieve the desired functionality. Deep learning refers to neural networks, which have multiple hidden layers, it is widely used in many fields ranging from speech recognition, image processing, search engines and language translation. Deep learning methods prove to approximate problems, that are in nature extremely difficult, with acceptable error. A survey of such neural networks follows.

### 1.5.1 Artificial Neural networks

A neural network is a computational model, that was inspired by brains and neural tissue and has interesting capabilities. It consists of computational nodes interconnected with each other by directed edges. Every node, so called neuron, has a number of input and output edges. It computes its output with an activation function by weighing signals coming from input edges. This is an analogy to synapses in organic neurons, which connect many neurons and transmit signals throughout the body by emitting chemical substances, that stimulate neurons to forward the signal further. Generally, three types of neurons are distinguished. Those, whose input edges are not connected to other neurons, but to the networks input are aptly called *input neurons*. On the other end of the network are *output neurons*. And between these two layers reside *hidden neurons*, whose input and output is connected to other neurons. By changing the network's architecture, one can acquire various types of neural network models, which can have a range of intriguing properties [11].

With data it is then possible to teach the network – which basically means to adjust the weights of each single neuron, so that the network's output is as close to the correct output as possible. This is done using optimization algorithms, usually a variation of gradient descent or back-propagation. In the phase of training, the network is fed with training data in batches. After each batch-processing iteration, a loss function calculates, how correct or wrong the network has been. To improve, the optimization algorithm calculates by how much the weights should be adjusted. Besides loss, also accuracy – a metric

of how precise the network was – is used to estimate the quality of the neural network [11].

### 1.5.2 Multilayer Perceptron

Multilayer perceptron (MLP) is a common type among neural networks, which consists of layers of neurons, that are fully-connected with neurons in their previous layer. The data in MLP flows strictly forward, without loops, which is why they are part of a group of networks called feed-forward neural networks. These networks are frequently used in classification problems[6], speech and image recognition. Furthermore, they often complement other networks in more complex settings, often as the last layer in a stacked architecture [12].

Mathematically speaking, MLP can be described as a directed acyclic graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$. Activation function of the neurons can be expressed as a simple real function $\sigma : \mathbb{R} \to \mathbb{R}$. Usually, only monotone activation functions are used – for instance, sign function, hyperbolic tangent function, sigmoid and many more [10].

Edges in the graph correspond to neuron connections and nodes to neurons themselves. Layers than are disjoint subsets of $V$, in other words, $V = \bigcup_{t=0}^{T} V_t$, where $V_0$ is the input layer, $V_T$ is the output layer and the rest are hidden layers. It also holds, that every neuron in layer $V_{i+1}$ is connected to all neurons in layer $V_i$ for $i \in \{0, \ldots, T-1\}$. For example, for $T = 1$ the neural network's architecture is in the form of a bipartite graph. Also, neuron's output is computed with activation function $\sigma$: $o = \sigma\left(\sum_{i=0}^{n} w_i \cdot x_i\right)$. Here $x_0 = 1$ and $w_0 \cdot x_0$ forms so called bias, and the rest – $x_i$ and $w_i$ for $i \in \{1, \ldots, n\}$ – are inputs and corresponding weights from other neurons. The above mentioned bias enables neurons to learn a specific offset from the inputs [10].

### 1.5.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a feed-forward neural network that changes the neuron structure and network architecture by "*the inclusion of edges that span adjacent time steps, introducing a notion of time to the model*" [13]. This allows for more semantic modelling of time-series, predictions, music, video, language or speech. Recurrent Neural Network (RNN) introduce recurrent edges, which are connections between the same neuron in different time steps – stages in time of processing. This can be better understood from the depiction in Figure 1.1 [13].

The reason these neural networks are called recurrent is, that in each time step, the neuron computes its output based on two sources. The input vector and its previous state in time (its previous time step). This makes it possible for RNN to create relations in time between input data and facilitates, among others, time-series modelling [14].

[6] Deciding whether an input belongs to a specific class or not.

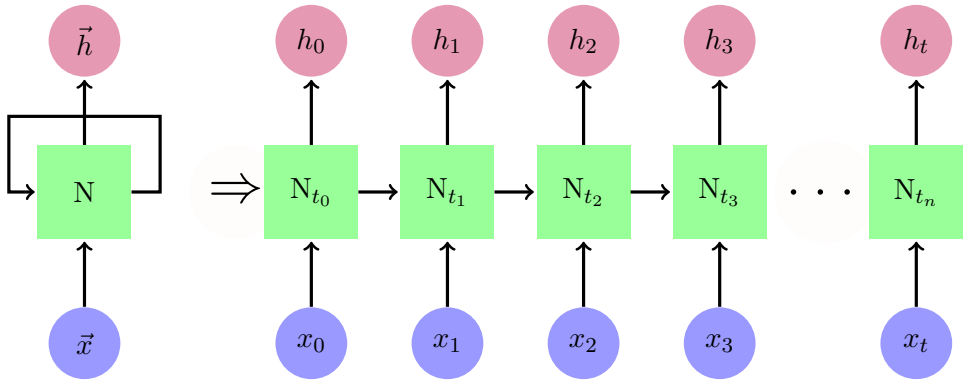



Figure 1.1: An illustration of an unrolled neuron in a Recurrent Neural Network. The vector $\vec{x}$ corresponds to network's inputs, the vector $\vec{h}$ is the output and finally $N_{t_i}$ marks the neuron at time step $t_i$. In each time step, as represented by arrows in the diagram, the neuron gets fed two inputs, $x_i$ and also the previous state $h_{i-1}$.

### 1.5.4 Long short-term Memory model

Traditional RNN models suffer from troublesome training, which is said to be caused by vanishing and exploding gradients due to the fact, that the back-propagated error increases or decreases exponentially with the number of layers. This makes learning hard, although recent advances in computer performance, especially that of GPUs, allow to mitigate these problems. They also are not good at *remembering* things through time [13].

Long Short-Term Memory (LSTM), a modification of RNN, which was proposed by [15] in 1997, introduces a new RNN model with an innovative cell structure counteracting above mentioned issues. LSTM names neurons *memory cells* and enhances each one of them with added properties, so called gates. There are usually three gates – input, output and forget gate. They output a number between 0 and 1, which determines the amount of flow let through such gate [13, 15].

## 1.6 Embeddings

Some domain spaces, like languages, have a very high dimensionality and are very complex. Their typical representation in a computer does not always capture semantic meaning. Numbers, on the contrary, have a near perfect machine representation (except for precision and limited size) coming from the underlying design of computers. Numbers in the computer can be compared with one another and can be used in mathematical operations in their semantic way – on a computer, two plus two equals four and ten is less than one hundred. How does one, however, represent language (or anything other than numbers

for that matter) without losing (at least to some extent) semantic information? The answer is embeddings, which come into play to do just that.

An embedding is a function from a particular domain into a high dimension numerical space, which aims to preserve innate meaning of the domain data. Various methods exist for obtaining such a function. Generally speaking, they simply place inputs closer and further away from each other depending on criteria specific for each of these methods. Afterwards, the embeddings can be used to calculate similarity – with euclidean distance, cosine distance etc. – or to make certain operations with them [16].

A well-known method for word embeddings is called *Word2Vec*. Basically, *Word2Vec* works on the basis of a shallow neural network, which infers the semantic meaning of words based on the context they appear in. The results are quite astonishing. For instance, in embeddings arithmetic, $king - man + woman \approx queen$, $Madrid - Spain + France \approx Paris$ or $Germany + capital \approx Berlin$ [17].

In the context of Natural Language Processing (NLP) and machine learning, embeddings elevate the capabilities of designed models. A single word entails a lot of information, which is lost, if the word is represented as a mere identifier "*word number XY*". Embeddings give a better representation of the data, because they keep this information. As a result, the model can learn natural language tasks much better and in a more meaningful way.

Chapter 2

# Analysis and design

## 2.1 Requirements

Because the service itself is a new, proprietary software solution, requirements – functional as well as non-functional – could be defined arbitrarily. This meant a lot of freedom and many possibilities in choosing what to create and how to do it. However, practicality – that it could in fact be used by a client sometime in the future – had to be taken into account. For just like companies seized the power of recommendations systems to create more revenue, they will want to utilize modern NLP methods.

The broad domain field meant that the WS could be quite an extensive piece of software, which would exceed the scope of a bachelor thesis. That is why it has been decided to stick to basic functionality and known algorithms for now and focus more on the design and architecture of the software. Therefore, more emphasis and priorities were put in some of the non-functional requirements, because the system will likely be extended with new functionalities in the future or will have the current ones enhanced.

### 2.1.1 Functional requirements

Regarding functional requirements (shown in figure 2.1), the web service should be capable of being accessed by other systems via the Internet. This will allow to integrate the WS seamlessly with external systems – be it a specialized editing system for publishers, a website or anything else. This requirement has top priority, for it is an essential part of the system. Without it, the functionality cannot be utilized as intended.

As a way of showcasing the possibilities of the system, a web browser Graphical User Interface (GUI) has been required. Like that, user interactions with the system are made substantially easier. People, who might be interested in the topic of NLP, can use it to understand or get information and hands-on experience with NLP models. It was not necessary for it to be very complex,

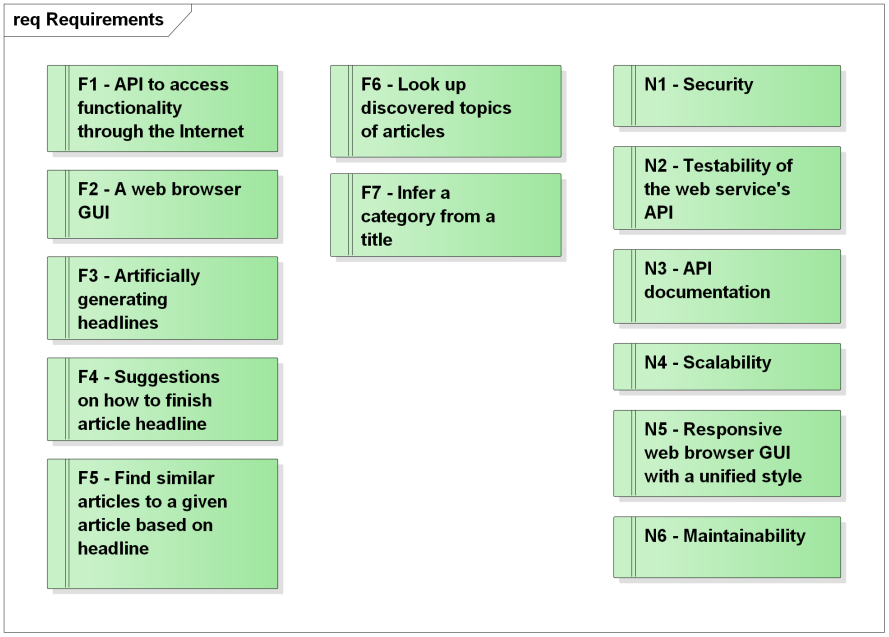



Figure 2.1: Specified requirements of the web service. Functional requirements on the left, non-functional on the right.

but instead rather show, what the web service is capable of. The website should also provide useful explanations about the models.

Following five functional requirements refer to what exactly the system is supposed to provide for the user. Firstly, an important ability to find similar articles in a set of already existing articles – this will enable the creator to compare his work with other articles, find some more relevant information to his topic, use it as a reference and so on. Because it makes for a very useful feature, it has been given a high priority.

Secondly and thirdly, artificially generating headlines either randomly or primed with a starting sequence. The starting sequence would then be finished and in this way create suggestions of an article title. The random generation is more of a secondary feature, but can be used to programmatically assign headlines to articles or serve as an inspiration for writing an article.

Some NLP techniques have the ability to extract what kind of topics appear in from text datasets and what words constitute such topics – e.g. LSI. Such functionality is presented by the requirement *F6*. Just like *F3* it has low priority.

Lastly, the requirement *F7*, which calls for the functionality of inferring category (or categories) from a title. This feature consists in telling the user, which category an article title he had provided falls into. Because this might become quite helpful in many regards (automatic article classification and so on), the requirement has high priority.

### 2.1.2 Non-functional requirements

Nowadays, security is an infamous issue of a large number of applications and information systems, more so on the web. That is the reason why the security of the web service (and the GUI as well of course) was required with high priority to protect against malicious attacks and to be sure, that no private information would get into unauthorized hands or leaked in any way. In other words, communication between the service and the client must remain secret and there should be no evident threat to the service of being compromised. To define it more clearly, the top ten security risks as defined in [18] by *OWASP* (if applicable to this case) should be covered.

An important quality of the web service's Application Programming Interface (API) is its testability. It reflects the need to ascertain, that the system is capable of proper functionality and does not evince signs of unexpected behaviour. Also, if the system is to be extended in the future, there must exist means of testing it, so that new parts do not break any earlier functionality without notice. In practice, it means that the entire API (all of its methods) are required to be tested.

An essential requirement on the system is the API documentation – that of all available methods, with chief descriptions and documentation of parameters, input and output formats. It will serve as a point of reference for anyone who wishes to use it, either themselves or in a program. It has, of course, a high priority, because without it, no one would be obliging to implement it into an external system. Also, public availability of the documentation was of essence. So that the documentation stays updated, automatic generation is preferred, but not required.

Because natural language processing algorithms can have heavy demands on hardware, scalability of the WS was crucial. To ensure reasonable response times and to be able to react to increasing demands (even hypothetically speaking), preparing the software to scale well from the start of the project makes more sense than to arduously compensate for it later. That means, that the software has to be capable of taking advantage of multiple cores, processors or even to be able to run on multiple machines each with its own instance.

Next, the web browser GUI, even though it does not have to be very complex, is required to have a unified style and a responsive layout, so that it can be displayed on devices with smaller screens without trouble and still remain user friendly. The GUI is one of the goals of this thesis, so it has top priority.

Lastly, the requirement for maintainability is also very important, even with not so extensive code bases like this one. To be precise, required is: versioning of files, which are not generated, documentation of the most important parts, logging and readable code.

## 2.2 Primary use cases

It has been analysed that just two actors are going to interact with the system. A human – who would like to obtain more information on the article he/she is creating or someone, who wants to know more about NLP – or, as was mentioned earlier, an external editorial system. Everyone can access all of the functionality of the system without constraints, even though they do it in a different fashion.

The diagram for the use case model can be found in figure A.1. The use cases themselves are rather straightforward – the user simply accesses the capabilities of an algorithm. Therefore, it was not necessary to perform detailed modelling of each case, because the added value would have been basically none.

The table 2.1 then summarizes which requirements realize each of the use cases. Further reference on requirements and use cases can be found in figure 2.1 or A.1.

| | **F3** | **F4** | **F5** | **F6** | **F7** |
|---|---|---|---|---|---|
| **UC1 ($n$-gram)** | | + | | | |
| **UC2 (Char-RNN)** | | + | | | |
| **UC3 ($n$-gram)** | + | | | | |
| **UC4 (Char-RNN)** | + | | | | |
| **UC5 (Word2Vec)** | | | + | | |
| **UC6 (Tf-idf)** | | | + | | |
| **UC7 (LSI)** | | | | + | |
| **UC8 (NN – Seq2Cat)** | | | | | + |

Table 2.1: This table depicts, which requirements are fulfilled by which use case. Requirements F1 and F2 are realized in all cases, because each algorithm is available in the API as well as via the web browser GUI.

## 2.3 Design decisions

The programming language chosen for implementing the web service became Python 3. Python is a high-level, dynamically typed interpreted language with a big community, support and is used by individuals in small projects as well as in big companies for large solutions. It is widely used in hundreds of fields, among many as a rapid development tool in agile software development, as a testing and penetration tool, on the web, in science and engineering, business and also education. Python especially took root in computer science and machine learning thanks to many outstanding frameworks and libraries facilitating experiments, but also real-life applications that are truly changing the world [19].

Despite not being the fastest language – compiled languages like C or C++ are a magnitude faster, even Java – when performance is needed, one can harness the power of a different language through Python, because it has bindings to many of them, even Fortran [20]. Many frameworks use that to bring performance to Python, for instance the ecosystem SciPy. Python compensates for the lower performance with its concise, readable and explicit format, which significantly increases maintainability of the code base.

From the point of developing the web service and web browser GUI, the framework Django seemed like the most suitable choice [21]. Django is excellently documented (that helped immensely), very secure and scalable, which is why it is used by many prominent websites of companies like Pinterest, Instagram, Spotify and NASA. It can be extended with a module for creating REST web services called Django REST Framework, which integrates with Django, has similar semantics and enables browsable documentation for the API.

The decision on the web service's style, whether to employ REST or SOAP as its means of communication, has been partially decided by choosing Django as the main framework for implementation. Mainly because of Django REST Framework. Not only that though, REST fits the needs of the requirements perfectly – it is flexible, straightforward, simple and yet powerful enough. The web service is designed to be available only via the Internet, so the SOAP's advantage of the possibility to use multiple protocols cannot be utilized. Basically, its added complexity was weighed out by REST, because it would not be worth it.

Django has built-in support for communications with databases and comes equipped with a default SQLite database (DB). It is a lightweight, reliable DB and does not run in a separate process – SQLite comes alongside Python and will do with a single regular file. On top of that, Django comes with its own DB migration system, that significantly increases maintainability, because any changes to the DB schema are managed automatically.

Constructing DB tables becomes easy, as they are defined by classes in Python and the communication between Django and the DB is left to the framework. For this Django uses the so called Active Record pattern as defined by Martin Fowler in [22]. This pattern, practically, maps classes with attributes to tables and columns in the database, and then offers CRUD[7] methods to operate with them. Such behaviour, in cases of applications with less complicated database schemes, cuts down duplication of the domain in the DB and in code [23].

Furthermore, a deciding point was also the fact, that machine-learning research is carried out mainly in Python and there exist quite a few good libraries for it to do natural language modelling. For example, TensorFlow by Google, which is a library enabling computations through data flow graphs [24]. It

[7] Create, Read, Update, Delete.

is possible to run it either on CPU or GPU by simply installing a different version of it (the code does not have to be changed at all), in fact with a few tweaks, it can even run on a CPU/GPU cluster. TensorFlow became one of the most popular frameworks for training neural networks. For higher abstraction of neural networks, using Keras makes developing experimental, but also production-ready, solutions even easier [25]. There is also the possibility of swapping the back-end of Keras from TensorFlow to Theano – another popular machine-learning library for Python [26].

## 2.4 Architecture of the system

When designing the architecture of the system, a couple of things had to be considered. First of all, as both a GUI and an API were required, the architecture must make an effortless extension of the presentation layer possible. Besides, these extensions needed to be independent of each other, so that making simple changes in one, would not propagate into other presentation modules.

The architecture was also required to permit easy addition of new models to facilitate extensibility of functionalities. Most of the used algorithms depend heavily on data and various parameters, that when changed, can significantly alter their behaviour. To that end, the architecture had to support loading of different versions of such models and algorithms, including versions trained on different languages.

In the end, it lead to a strict[8] multi-layered architecture with four layers as can be seen in the figure 2.2. Purpose of the technical layer is frankly to manage data and its structure. The engine[9] layer then consists of classes that make up the functionality of the application. That means various kinds of NLP algorithms and models.

After that the business layer provides an abstraction – an interface – that unifies the access to core functionalities. This is essential, because it enables better control over the models and isolates any changes from the presentation layer. This in consequence makes it possible to have the application easily extended from within, without breaking the packages in the presentation layer. After confirming that new functionality works properly, extending also the presentation layer correspondingly is no problem.

In the presentation layer, as many as desired implementations are possible thanks to the layered architecture and therefore the plan of opening more than just one way of accessing the application is fulfilled. As can be seen in the

[8]The packages' dependencies never *skip* a layer and depend on the package(s) directly under them.

[9]In the context of this work, engine does not refer to a motor, as in *car engine*, but rather to something that provides means of accomplishment. In the meaning of, for example, a *search engine* or a *game engine*.

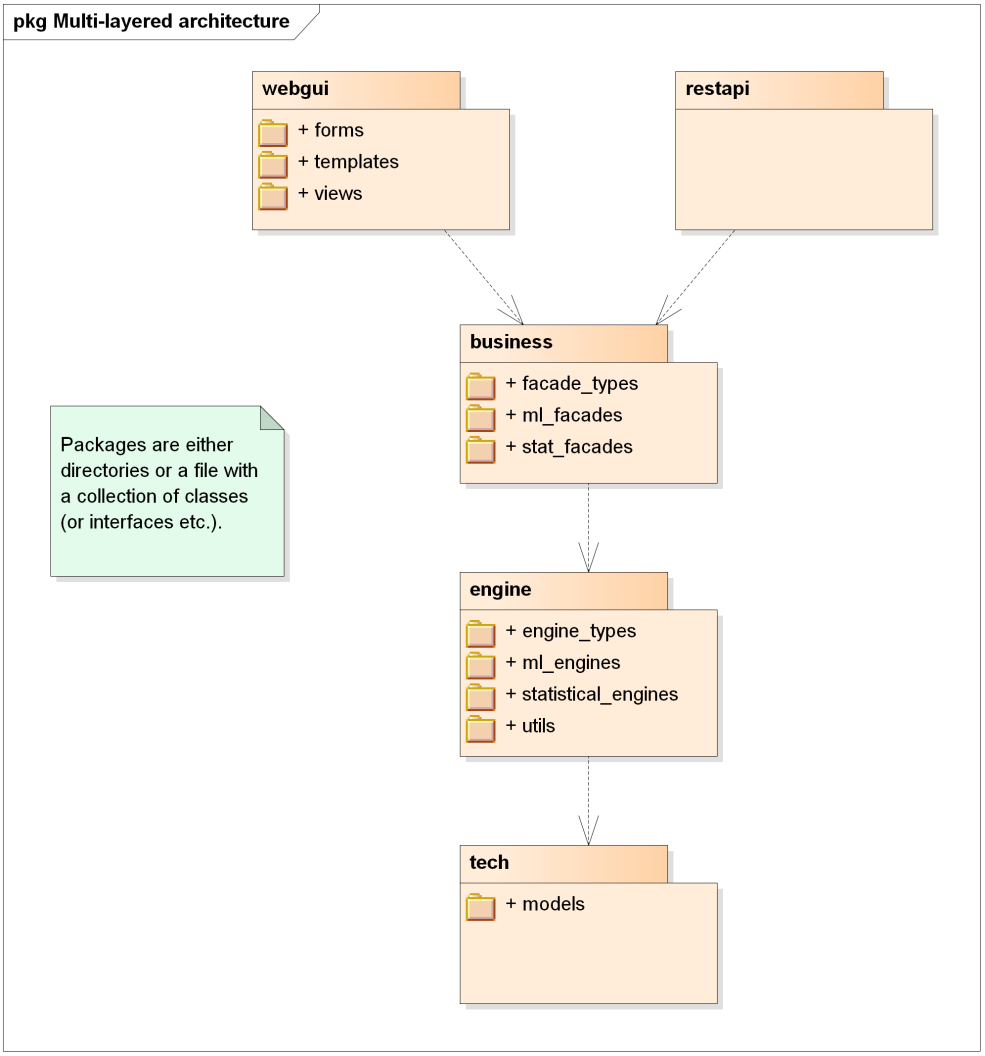



Figure 2.2: Package diagram of the application's architecture. It is a multi-layered architecture comprised of presentation, business, engine and technical layer.

figure 2.2, the web browser GUI is implemented in package *webgui*, and the API in package *restapi* both of which depend solely on the business layer.

One might argue, that the Active Record design pattern used by Django to persist objects, violates the strictness of a multi-layered application, because it gives anyone with a reference to the object the possibility to access data. Though that might be true in some cases, in this application, however, data persistence is used only by the engine layer, and if any other subsequent layer needs it, it would receive the data merely in the form of basic data types.

### 2.4.1 Web browser interface

The web browser GUI then is organized much in the same way as the well-known and widely used Model View Controller (MVC) design pattern. The concept still remains similar as illustrated by figure 2.3, but to stay consistent with Django's naming, the packages are a little renamed [22].

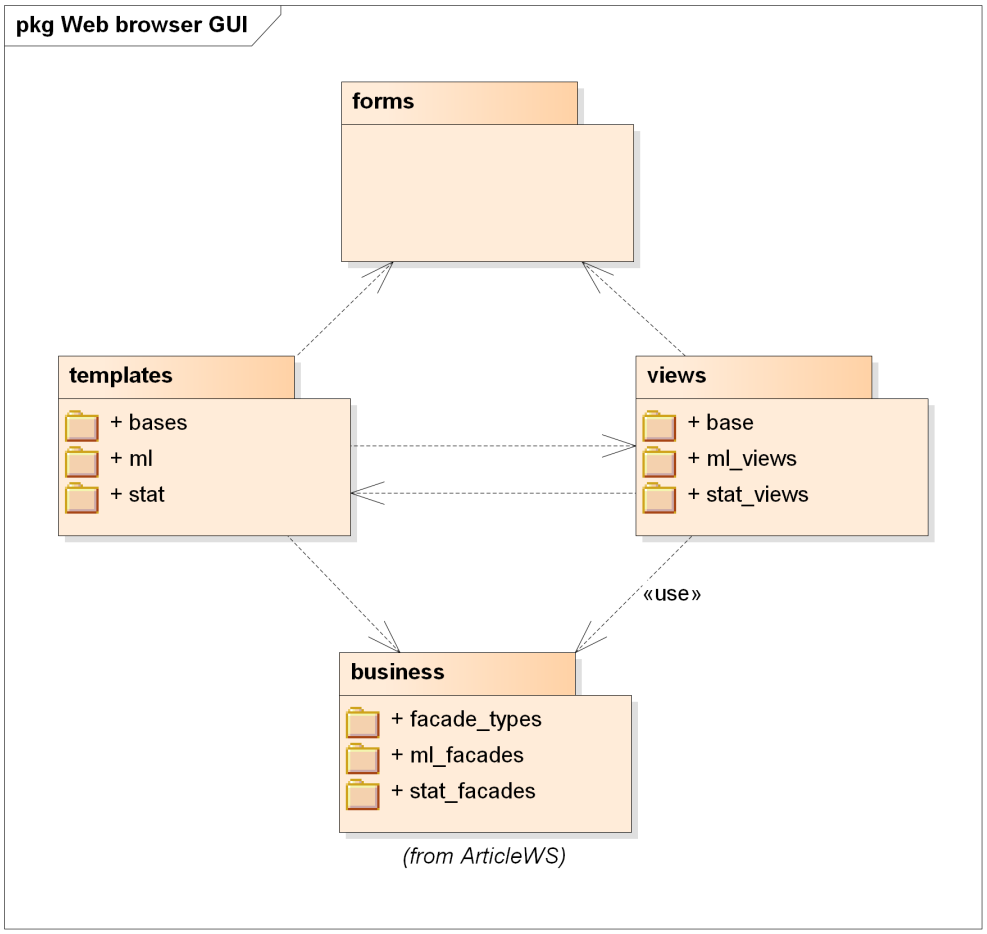



Figure 2.3: UML diagram showing how the package *webgui* is organized similarly to the MVC design pattern, but with addition of the *forms* package. The view from MVC corresponds to *templates* and controller to *views* so as to stay consistent with Django's naming. The forms and business package together make the model from MVC.

Django's templating system for creating websites, goes hand in hand with MVC and so eases the design of HTML pages by enabling component composition and with flow control elements. It resembles in many regards JavaServer Faces (JSF). A template may inherit from another one and override some appearance or may also include different templates. This makes for an un-

complicated and yet flexible system, with which even complex websites are feasible.

The forms package contains a specific structure of parameters and data used for communication between the templates and views. Templates display content directly to the user and relay his input to view classes (controllers). These process the user's input and call business methods, whose results are then updated back to the user via templates. The results' structure, however, is not changed by the views, but gets presented by the templates, which is why templates depend on the business layer as well, even though do not call any methods. Basically, the forms and business package constitute the *model* from MVC.

For some it might seem counter-productive, nonetheless, it brings many advantages. If, say, a business method's name gets changed, only the views have to be adapted. If method's parameters change, possibly only the views need modifications again. In some cases (a totally new parameter added to the method), the change would propagate into forms and templates as well, but that would be the same for a traditional MVC. And if the result format of a business method changes, only templates would need an alteration, in contrast of both views and controllers in MVC.

### 2.4.2 Web service's API

The API is formed by classes defining REST methods, see also figure 2.4. The data is transformed from data objects into various data formats and vice-versa automatically with the aid of serializer classes. The user can choose the format himself by specifying it in the URI, as a *format=<format>* query parameter of a request or as a content and accept header in the HTTP method depending on what is more suitable for him. The representation of objects in a given format is then handled by the serializers. Among available formats for output are JSON, XML and HTML, the same for the input with an exception of HTML.

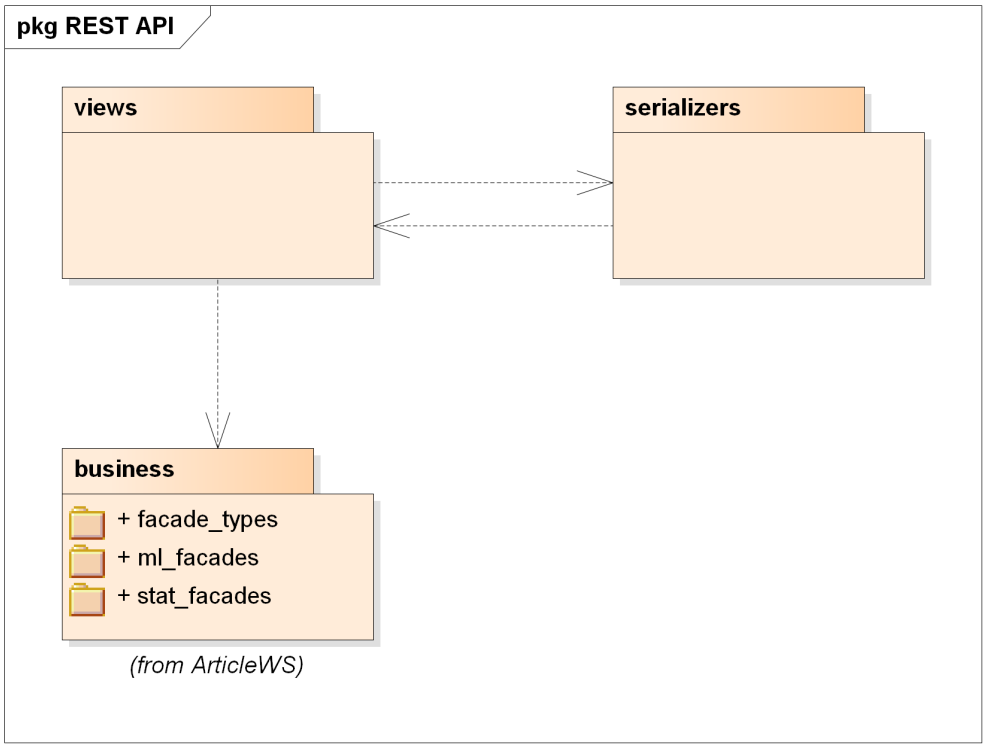



Figure 2.4: The package *restapi* defining the accessible API. The *views* consist of classes that implement and determine API methods with the help of *serializers*, which transform the data into and from desired formats.

Chapter 3

# Realization

## 3.1 Implementation

### 3.1.1 The engine layer

The application's backbone, classes that provide functionality, are located in the package *engine*. They constitute what the service actually does. Because the application needs to be well maintainable and expandable, the engine classes would not do without a proper structure. Because there are quite a few classes, only a subset of them is illustrated in the figure 3.1. The entirety of the documentation on classes from this package is available on the attached CD.

The concept of the *engine* package is, that the classes are programmed "against an interface". This means, that a class, that wants to provide certain functionality has to form a contract with an interface by implementing it. By forming a contract, the class is bound to certain behaviour, which means, that all the classes, that share the same interface, can be uniformly accessed and dealt with. On account of that, it is possible to reuse a lot of code, that uses different engines, which, however, implement a common interface.

For instance, the base interface of the engines, called *IEngineType*, defines what method every model should implement. After that, sub-interfaces, like *ISimilarityType*, establish what kind of functionality is to be expected from classes that implement it. A class implementing *ISimilarityType* then would have to implement all of its methods including those from *IEngineType*.

Because of this, not only can the interface be implemented by an arbitrary number of classes[10], but also a class can implement many interfaces, without the issue of inheritance ambiguity[11]. The interfaces do not implement any-

[10] As is the same with classes and inheriting sub-classes.

[11] The diamond problem, for example, where a class inheriting from two classes that share a common abstract parent class (with some abstract method), does not know, which method implementation to inherit.

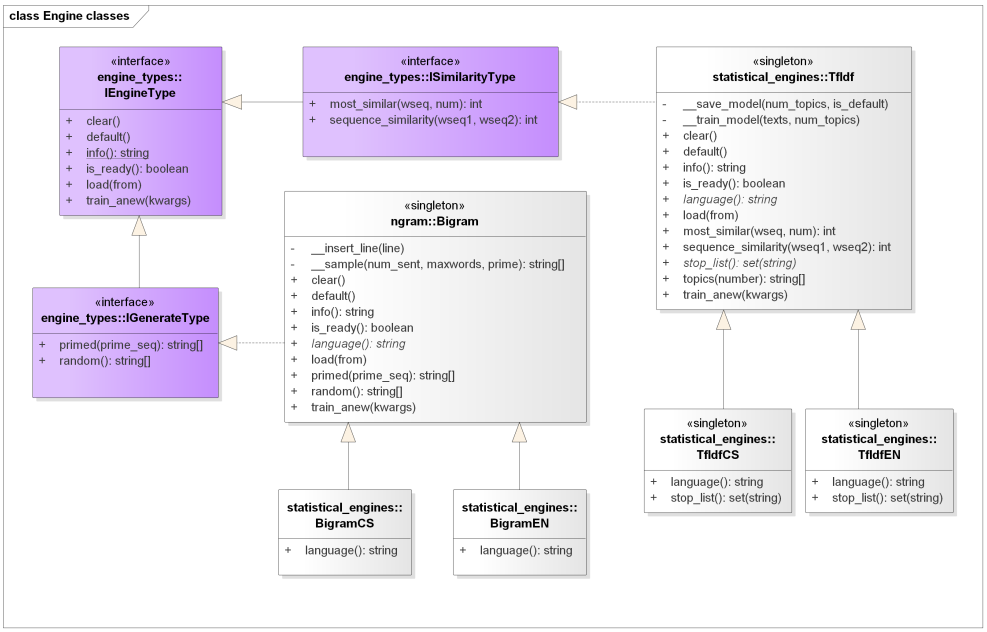



Figure 3.1: The package *engine* contains classes responsible for actual functionality of the web service. This UML class diagram shows only a subset of them for better understanding. The whole structure can be found on the attached CD. Also note, that Python has no special construct for interfaces, so instead in the code, abstract base classes are used, whose methods have no implementation.

thing, so even if two interfaces defined methods with the same signature, the class would implement the method just once and satisfy both contracts.

This way, new engine models can be added merely by implementing either an existing interface, or a brand new interface can create a new contract for a model providing a totally different functionality to that thus far.

### 3.1.2 The business layer

The business layer serves as an isolation between the actual implementation and the presentation layer, hence it follows the *facade* pattern. It isolates any changes done to the inner workings of the engines from upper layers, but also leaves space for more control over the engines.

More on the actual structure of the classes can be found in figure 3.2. All classes of a type of an engine comply with the exactly same contract, which also means, that they can be handled in the same manner. Therefore, the common functionality of the facades is pulled up into classes with static methods, which take over the control and operate on instances of an engine. Moreover, facade classes are responsible for calling appropriate models according to the language

requested. Some of the models – e.g. Char-RNN, Seq2Cat – do not yet support multiple languages, only Czech. In such a case, an exception indicating, that a language version is not supported, is thrown.

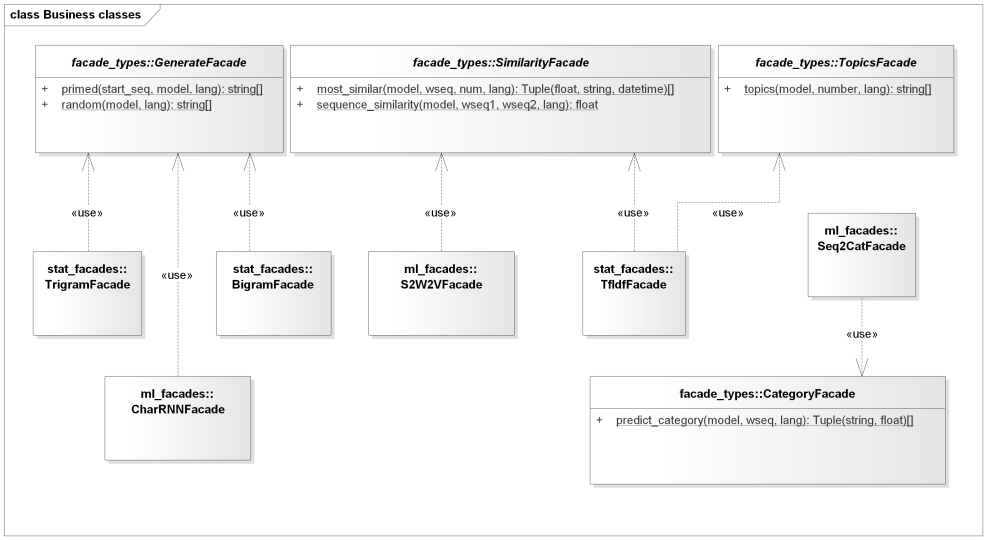



Figure 3.2: A UML class diagram showing the organization of the business layer. Models of the same type (that implement the same interface), can be dealt with the same way, therefore, functionality is pulled up into common classes with static methods.

## 3.2 API documentation

It was crucial that the API had good documentation available. Therefore, there are actually two ways of exploring it. First of all, it is possible to open the API in a web browser and visit any available method by specifying its URI – for the root method it would be *https://⟨*domain-name*⟩/api/*. From there one can send any requests and see the results immediately. In addition, *Swagger* docs are available too – simply by going to *https://⟨ domain-name⟩/api/docs/*. At that site all possible methods are listed with signatures, formats, parameters, responses and brief description. Swagger docs can be used to access and examine the API as well, probably in an even easier way than via the browsable API.

## 3.3 Testing

Frankly, the entire API was tested. The Django REST framework provides a useful suite integrated with Python's unit testing framework *unittest*. The tests then can build regular HTTP requests and send them as, for example, a

client would. Hypertext Transfer Protocol Secure (HTTPS) is possible, too, which was vital, because the whole application runs behind it.

The testing itself is run on an in-memory testing database, which is re-populated with data from the production database, because data for running the algorithms needed for their execution is stored in there. As a result of creating a separate DB, no data in the production DB can be damaged by running tests, because it is ensured, that they run in an isolated environment. Above that, each test runs in a DB transaction and any changes done by it are reverted back immediately to how they were before its execution. So, each test begins with a fresh DB.

Generally, the application was tested in the manner of black-box testing as a whole. Tests are not designed to go through certain paths in the system, but are examining, whether the API meets promised functionality. To expand on that, how well certain algorithms or engines work – how accurate are their results or predictions etc. – was not tested, as that can be very ambiguous and sometimes nearly unfeasible.

It was assumed, that engines of the same type are supposed to behave in the same way, which is why the tests are homogeneous for all of one type. The content of the messages is not tested in great detail, on the other hand, an emphasis was put on proper structure, parameters, types, status codes and expected errors. Tested was mainly the API – by doing that, most of the functional requirements were covered and verified.

## 3.4 Deployment

The Django application runs as a Web Server Gateway Interface (WSGI) compliant client in a *uWSGI* server. WSGI is a Python standard which defines how web applications communicate with each other and how they process requests. All in all, the *uWSGI* server does nothing else, except for passing on requests to the application and then its response back to the client [27].

For persistence of data, the application accesses an instance of *SQLite 3* database. It is, however, only a matter of configuration (setting up how Django should access a database server, authorization etc.) to switch the database for a different one like *MySQL*, *PostgreSQL* or *Oracle*. This has the advantage, that the database can run as a separate process or even on a totally different machine for that matter. On account of that, scaling the data layer is undemanding and uncomplicated. For now, though, a simple set-up of the database like this, fulfils the needs for enough performance, because the application is not data demanding. On top of that, the *SQLite* database is embedded into Python, so migrating the application to a different node means just relocating one file – no need for installation of a new database, configuration, restoration and so on. This further elevates the flexibility of the application.

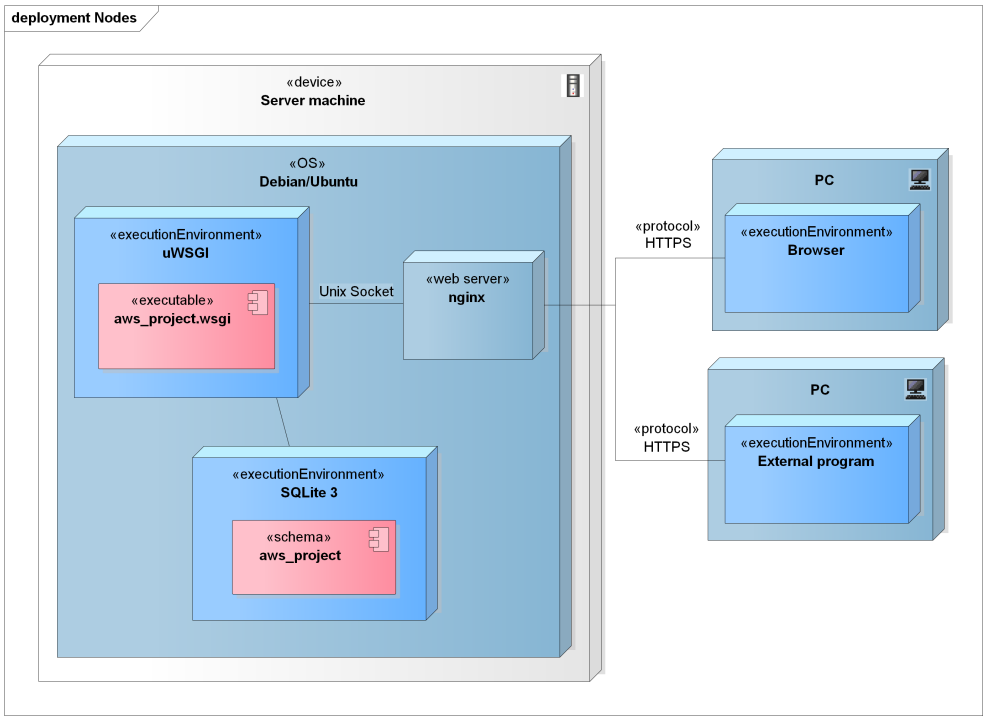



Figure 3.3: UML deployment diagram describing the deployment settings of the web service. The web server *nginx* communicates with a *uWSGI* server, that contains the Django application.

As can be seen in figure 3.3, *uWSGI* communicates with a reverse proxy server, *nginx*. *Nginx* is a high-performance and lightweight production web server used on high traffic websites, but is perfect for less demanding jobs as well. The two servers communicate with each other through a Unix Domain Socket, which is faster than through a TCP/IP port. Nevertheless, it is possible to run *uWSGI* and *nginx* on different server machines and let them communicate via TCP/IP. And not only that, but multiple instances of the application may run on different nodes. The *nginx* then would take care of load balancing between the nodes, so that they share the traffic.

As a consequence, this configuration scales without any particular hindrances, which makes it just a question of changing the communication settings, installing dependencies and the application itself on new hardware to have it utilize more resources. For instance, the application would benefit from dedicating algorithms, that could be sped up by running on GPUs, to a node specifically designed for them, like a GPU cluster. With this, the advantage of TensorFlow being able to run either on CPU or GPU just by swapping an installed Python package, would really come to fruition. As an illustration of how such deployment would work, an example deployment diagram can be seen in figure 3.4. For more optimization, the nodes can communicate through

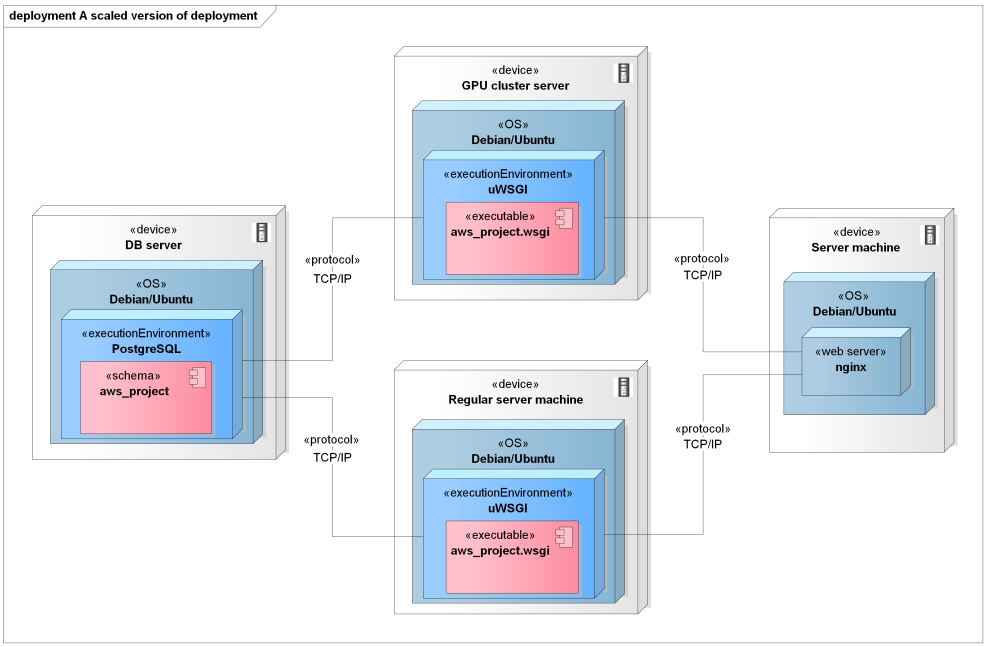



Figure 3.4: This deployment setting shows how the web service could be scaled to utilize more resources. The *nginx* web server can be configured to balance load between multiple nodes or to route specific requests to a certain node. Here, the *nginx* server sends traffic of algorithms, that are able to use GPU, to a dedicated GPU cluster. Also, the database runs on a separate server.

faster means, the database could be replicated and run on more machines or more machines might be dedicated to the algorithms.

## 3.5 Security

In this day and age, securing software continues to be of profound importance. For this reason, it had been decided, to make the application and communication between the server and client secure. The sent data most probably will not entail any truly confidential data, but communication privacy, even for simple applications, is becoming a standard. That is why the application employs HTTPS entirely – for all of its communication through the Internet.

Moreover, as low a possibility of someone targeting one's web application (or server, website...) is, it never equals zero. That is why, the application was designed with security in mind. The Django framework helped a lot in achieving this goal.

Security of the program on its own was required (as defined by the non-functional requirement *N1*, see figure 2.1 and section 2.1.2) to focus on security risks summarized by [18] and try to mitigate them as much as possible. Some

of them do not, in fact, apply to the application's case, because users do not authorize themselves at all and no data about them is stored. That means, that the risks of *Broken Authentication and Session Management*, *Sensitive Data Exposure* and *Missing Function Level Access Control* are not relevant. Just like the risk of *Unvalidated Redirects and Forwards*, because there are none.

For the case of *Injection*, the Django framework takes care of properly escaping all inputs, so that no DB queries result in vulnerable points of an application. Subsequently, *Cross-Site Scripting (XSS)* is prevented by Django's innate auto-escaping feature and by not storing or sending any user data [28]. Then, *Insecure Direct Object References* were carefully avoided using only serializers (which allow only concrete parameters of primitive data types) to show results to the user and by catching and controlling any exceptions that might occur. This way, no unwanted information gets to the user.

In order to avoid *Security Misconfiguration*, a lot of effort was put into configuring correctly the *nginx* web server. It accepts merely HTTPS and any HTTP requests are redirected with a status code 301 *Moved Permanently* response. The employed Secure Sockets Layer (SSL) protocol is Transport Layer Security (TLS) 1.2, just as recommended by *Mozilla* in [29] and only modern cipher suites are used. HTTP Strict Transport Security (HSTS)[12] is enabled to have the web server tell the client's browser to use only HTTPS, if he tries to connect with HTTP. Thanks to that, Man-In-The-Middle attacks' difficulty is greatly elevated. Afterwards, during handshake, it is set up to prefer the server's cipher, instead of the client's. Also, to prevent reduced security because of unreliable Online Certificate Status Protocol (OCSP) responders, OCSP stapling is allowed.

The next common weak point, *Cross-Site Request Forgery (CSRF)*[13], has in Django a standard countermeasure – every page that has a form with a POST request, creates an authentication token to prevent CSRF attacks.

Finally, the issue of *Using Components with Known Vulnerabilities* is, admittedly, a little problematic. All software has bugs – that is why software maintenance exists. Most importantly, however, these bugs, when discovered, need to be fixed by someone. As for that reason, frameworks with a lot of support, big community or commercial backing have the upper hand when it comes to mending mistakes and releasing updated packages as soon as possible. In that regard, the key components of the application – Django and Django REST Framework – have it all. In addition, no known vulnerabilities, which would threaten the safety of the web service, were identified in any of

[12] If a user sends a HTTP request on the server, it tells the browser to use only HTTPS for communication.

[13] The application is requested to do a certain unintended action when the attacker takes advantage of an already signed in and authorized user to do it. The changes done might not be even detected, because the vulnerable application takes it as a regular, legitimate request.

the used libraries. Nevertheless, as indicated above, one cannot be really one hundred percent sure about that.

## 3.6 Experiments

In order to create the intended functionality, it was necessary to conduct some data analysis and a number of experiments to evaluate, what could or could not work. This section describes some of the more interesting models, that have been experimented with. As stated in the thesis' task, language-independence should be taken into account when designing or choosing the models. In some cases therefore, the web service allows to choose the language, either for Czech or English article headlines. The dataset of Czech headlines originate from a Czech company *e.conomia* and the English dataset comes from a data repository for machine learning, fittingly called – *Machine Learning Repository* [30].

### 3.6.1 Popularity prediction

Predicting how popular a certain article might be, would definitely proof to be a very useful feature, which is also a reason why it had been desired in the first place. However, expertly predicting popularity is quite difficult even for humans, let alone machines. How would it be even defined? Is, for instance, bad popularity a popularity as well? How is it measured? In page views, comments, likes and/or hates?

These and more questions arose during investigating what kind of method to use. Deep learning classification models seemed like the best choice, as they exhibited promising results not only in natural language processing. To make the problem a little easier for the models, it was simplified to a simple decision, whether or not the article will be popular. Nevertheless, in the end, none of the tried classification neural networks could learn how to predict popularity.

Only Czech data were used for this model in particular, because the English dataset does not contain any information, that could be used as a popularity factor. In the Czech dataset, the only information available were page views and on top of that, the data was very noisy (see figure B.1) and had only about 7000 samples even before excluding outliers. Articles, whose sum of page views was below the median, were deemed unpopular and those above popular.

Experimented was with a combination of a recurrent and convolutional network, with a perceptron classifier on top. Then also with just an embedding layer, recurrent neural network and a perceptron classifier. Both could not learn anything in particular, but always predicted the headline to be popular, which clearly is quite miserable. Changing the parameters of the models was of no avail.

The reason for such an ineffectiveness lies not only with the data, but also with the task itself (as implied above). There is no major indication of popularity in the text alone. Moreover, multiple external factors influence the popularity of an article, so the model does not receive relevant information, from which it could possibly assess how popular an article would be. That is a major problem and without data on these factors, creating a model capable of predicting popularity is probably impossible. To mention but a few of the external factors:

- Time, when the article is released on the website, determines the audience and number of people, who will see it on top of the page.
- The author can also decide, where he/she will place the article, which again influences page views a lot.
- Some articles are posted on social media, some not. Because of that, comparability of articles is lessened.
- Some articles are promoted, which causes the same issues as the one before.

All in all, noisy and scarce data, external factors and a problem hard to grasp even for humans meant, that the models were not successful. So for now, prediction of popularity was determined not to be incorporated into the web service.

### 3.6.2 $N$-gram models

The advantage of $n$-gram models is, that they are well interpretable. They get significantly more efficient with more and more data, as they gradually approximate a language better and better. The choice of $n$ largely affects how it behaves. A bigram, for instance, has a lot of freedom in the probability function – the probability of the next word depends solely on the one before it. With a growing $n$ though, the probability is stricter. The model loses room for randomness little by little on the same data. If a dataset had sentences of a maximum length of 20 words, a 21-gram[14] would be just a uniform probability of the sentences.

With a higher $n$, the demand for a large data corpus increases. That is why in practice, bigger models than 5-grams are usually not used. Considering the data available, bigrams and trigrams were most suitable and interesting to work with. They are accessible through the web service for Czech as well as English article headlines. The only downside to $n$-grams is, that they are not at all creative, they will never create a new word or will not work at all when

[14] $N$-gram is conditioned with $n-1$ preceding words, so a 21-gram would even for a 20 word long sentence always predict the next word as an end of a sentence.

told to predict what comes after a word (or a word sequence of course), that did not occur in the training dataset. That is understandably, a limitation of their design itself.

### 3.6.3 Tf-idf and Latent Semantic Indexing

For implementation of Tf-idf and LSI, a NLP framework *gensim* was used [31]. It provides functionalities for both out of the box, which greatly facilitated its integration into the web service. Tf-idf is able to find similar headlines quite well, especially when the headline contains keywords, names and terminology. It is available for the Czech articles and English ones, too.

Results of LSI are not as good on the Czech dataset. As described in subsection 1.3, LSI was used for topic extraction, but the generated sets of words, which are supposed to contribute the most to a certain topic, show only marginal and sometimes hardly noticeable similarity and correlation. The reason for it not working well was found to be attributed to the fact, that the Czech language has too many word forms, which for Czech will always be troublesome.

The English model, on the other hand, seems to have worked much better in some cases. The LSI could extract rather interesting topics about Harry Potter, vehicles, air-plane catastrophes, many technology and mobile phones topics. However, some words still appear quite randomly and without any relation. The model is, quite understandably, not perfect.

### 3.6.4 Char-RNN

Char-RNN is a machine learning model first devised by Andrej Karpathy, a prominent figure in deep learning, and posted on GitHub [32]. It was however implemented in Lua, so for the actual integration with the web service, a spin-off by Sherjil Ozair developed for TensorFlow and Python was used [33]. The model employs a deep recurrent neural network and predicts the next character based on the given dataset. As a result, it is possible to train the network on arbitrary textual corpora. The author of Char-RNN shows, that it has the ability to learn complex syntax and generate nearly valid mathematical articles in LaTeX, valid Wikipedia pages, XML or Linux source code in C, which is rather impressive [34].

For purposes of the web service, the Char-RNN was trained on the Czech dataset only. It was due to quite a lot of computation time needed, even when run on GPUs[15], and because later on, the GPUs were unavailable. Also, generating English headlines was already tried out in [35].

The various configurations of the model, that have been experimented with, can be seen in table 3.1. As the model learns in an unsupervised manner,

[15] On a GPU cluster with three GPUs, the training of the second and third network lasted for about a day.

| | #1 | #2 | #3 |
|---|---|---|---|
| **Sequence length** | 70 | 70 | 100 |
| **RNN cell** | RNN | LSTM | LSTM |
| **RNN size** | 500 | 600 | 600 |
| **Number of layers** | 2 | 3 | 2 |
| **Epochs** | 20 | 70 | 100 |
| **Training Loss** | 1.235 | 1.065 | 0.934 |

Table 3.1: Configurations for the Char-RNN model experiments.

evaluating which version can generate best headlines was left to the author. The training loss[16] can be a good indication. If it is too high, then the model did not learn anything. If it is too low then it over-fitted on the training data. In the end, the third version was chosen, because the headlines made sense the most and still were creative. Epochs, the amount of iterations on the training data, varied between the versions, because the training was stopped when no change was observed in loss in a span of multiple epochs.

The Char-RNN model achieved interesting results for generating random headlines. They are often quite readable, even grammatically correct more or less. From time to time, when the neural network has a "good mood", it generates entertaining headlines – funny and witty at that. Altogether, it is intriguing how the neural network hallucinates words and names, creates headlines that one would even be keen on reading or outputs random gibberish.

It was hoped, that the model would be capable of producing embeddings among its other functionality. The idea was to take the final inner representation of a sentence, when fed into the Char-RNN. These embeddings were then plotted with *t-SNE*, as can be seen in figure B.2. *T-SNE* is a technique for dimensionality reduction suitable for visualization [36]. On behalf of that, it can be seen, that the data is quite well clustered. However, because the embeddings come from sentences after they are fed into the network, the final embedding is retrieved when the last character is processed. This resulted in the problem, that embeddings corresponded to the ending character sequence rather than to their semantic properties. The above mentioned figure illustrates that, because the colour represents the last character of the sentence. For example, a big cluster of questions can be found, coloured in purple below on the right, they all end with a question mark.

Consequently, the Char-RNN model was deemed inapplicable to embeddings. Using it to get similarity between titles would not work well at all.

[16]Loss is a sum of errors (usually something like negative log-likelihood, residual sum of squares or alike), that tell how well a model is doing its job of learning. The lower the better. There are two types – training and validation loss, depending on for what data it is calculated.

### 3.6.5 Auto-encoders

A way of obtaining similarity between article titles would be to obtain their embeddings in a multidimensional space and use their distance to measure it as described in 1.6. Auto-encoders are neural networks that can produce embeddings of arbitrary data. Their concept is quite simple. Auto-encoders are layered networks, whose input and output layers share the same dimension. The middle layer, which usually has less dimensions, then constitutes the embeddings and functions as sort of a compression of the original data. The network is trained by instructing it to reconstruct its inputs as closely as possible.

For our datasets, the auto-encoders did not work as well as expected, even with multiple different configurations of the network – recurrent, convolutional nor multi-layer perceptrons performed enough to be used in the application. They were set up to reconstruct the titles character by character, with all kinds of hyper-parameters[17], however, the embeddings did not show any signs of somehow generalizing the titles, not even on the structural level of the sentences.

### 3.6.6 Seq2Cat

Sequence to Category, or *Seq2Cat* in short, is a neural network designed to classify a category of an article. It consists of an embedding layer, then a bidirectional Gated Recurrent Unit (GRU) layer and finally a multi-layer perceptron classifier with softmax[18]. Supervised training was used on annotated data – in the Czech dataset, a big amount of categories had to be merged with their more abstract equivalents, because these categories were not as numerous as others. After re-annotating some of the data, 18 categories remained – sport, hockey, football, free time, women, economy, science, life, motor sport etc. The network is not character-level though, it has a fixed vocabulary of about 80 thousand words – those that occurred at least twice in the dataset. The rest of the words is substituted with an $\langle unknown \rangle$ token.

The results with Czech data were fairly satisfying, for the network reached a validation loss of about 1.19[19] and an accuracy of 67.9%. Especially good in categorizing is the network when it comes to sports, football and hockey, these are recognized almost always. Exceptionally well fares Seq2Cat also in recognizing politics, domestic and foreign news. With motor sport, it is quite sure of itself as well, probably due to specific terminology used in such headlines. The model is limited only to the Czech data, but can, however, be easily extended to provide category classification for English as well.

[17] A term referring to activation functions, optimization algorithm, size and other parameters influencing the behaviour of a neural network.

[18] A type of an activation function

[19] For loss, categorical cross-entropy was used.

All in all, Seq2Cat was a success. Rather pleasing is also the fact, that it might become quite useful, not only for suggesting which category should the editor put an article in. For example, if a company had lots of data on unclassified articles (e.g. because they were old, some information was lost after a while etc.), rather than annotating them by hand, a Seq2Cat model would be a reasonable option to consider. With its help an author could also customize the title, so that it better fits a category, which would make it more understandable for the user at the same time.

# Conclusion

The thesis' main goal was to design and implement a web service that would be the basis of a service, helping content creators of Internet news websites adjust their articles – the title of an article – to better suit the needs of their clients. This was achieved by means of a multi-layered Django web application written in Python, that took advantage of a web service framework Django REST Framework. As clarified in part 2.2 and 3.1, the implementation succeeded in making the service widely extendible and secure. On top of that, it was created and designed to scale without any hindrances, because of the service's elevated hardware resource requirements. The web service was also tested, with automated tests, that run in total isolation from a running application in production deployment, even alongside each other on the same machine.

Secondly, implementation of a GUI in Django resulted in a responsive website, that is very informative and creates the possibility of examining how the employed functionalities work. The website makes for an access point to the API documentation as well, which further elevates the usability of the web application. The API itself can be accessed in multiple ways – from a web browser through the browsable API, from documentation or through a REST client. On top of that, the request body and the response can be formatted in JSON or XML, too. Because of that, the client has more freedom in implementing and using the web service.

Attaining desired functionality proved to be difficult at times, as some of the NLP models, which were meant to be used in the web application, did not reach expected results. Nevertheless, already as it is now, the web application offers an intriguing insight of what NLP models are capable of when dealing with news article data. While that is true, extensions of the functionalities will probably follow. Especially successful was the *Seq2Cat* model, which can classify an article title according to its category. This can be used to automatically annotate existing data or to ensure that a certain title will be immediately recognized by a human as belonging to a given category by modifying it in such a way, that *Seq2Cat* will classify it correctly.

Further work might consist in extending the functionalities of the web service. To give an example, summarizing the whole article body into a shorter sentence would yield interesting results in providing an attractive way of leaving the task of naming an article to the computer. Such a functionality might be supported (or be entirely separate for that matter) by an image captioning system, because practically all articles are accompanied with an image that further elaborates on what the article is going to be about.

If then some functionality of the web service would need to be exclusive only to some users (a company paying for a service for instance), an user authentication system would likely have to be implemented into the web service. Luckily, Django has inherent support for user authentication, so such a task would not be difficult to do. In addition, if there was a need, a mobile app could be another interesting way of demonstrating the power of NLP, because the web service's API can be integrated with basically any kind of device or system by design.

# Bibliography


[1] Tidwell, D.; Snell, J.; et al. *Programming Web Services with SOAP*. O'Reilly, 2001, ISBN 0-596-00095-2.

[2] W3C. Web Services Architecture. 2004, [Online; accessed: 2017-03-10]. Available from: `https://www.w3.org/TR/ws-arch/`

[3] W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). 2007, [Online; accessed: 2017-03-09]. Available from: `https://www.w3.org/TR/soap12-part1/`

[4] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation thesis, University of California, Irvine, 2000, aAI9980887.

[5] Wilde, E.; Pautasso, C. (editors). *REST: From research to Practice*, chapter REST and Web Services: In Theory and in Practice. Springer-Verlag New York, 2011, ISBN 978-1-4419-8303-9, pp. 35–57.

[6] Richardson, L.; Ruby, S. *RESTful Web Services*. O'Reilly, 2007, ISBN 978-0-596-52926-0.

[7] Manning, C. D.; Manning, H. S. *Foundations of Statistical Natural Language Processing*. The MIT Press, second edition, 2000, ISBN 0-262-13360-1.

[8] Manning, C. D.; Raghavan, P.; et al. *An Introduction to Information Retrieval*. Cambridge University Press, 2009, ISBN 0521865719.

[9] Landauer, T. K.; Woltz, P. W.; et al. An introduction to latent semantic analysis. *Discourse Processes*, volume 25, no. 2-3, 1998: pp. 259–284, doi:10.1080/01638539809545028, [Online]. Available from: `http://dx.doi.org/10.1080/01638539809545028`

[10] Shalev-Schwartz, S.; Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014, ISBN 978-1-107-05713-5.

[11] Fausett, L. *Fundamentals of Neural Networks: Architectures, Algorithms And Applications*. Pearson, 1994, ISBN 978-0133341867.

[12] Popescu, M.-C.; Balas, V. E.; et al. Multilayer Perceptron and Neural Networks. *WSEAS Transactions on Circuits and Systems*, volume 8, no. 7, July 2009, ISSN 1109-2734.

[13] Lipton, Z. C. A Critical Review of Recurrent Neural Networks for Sequence Learning. *CoRR*, volume abs/1506.00019, 2015, [Online]. Available from: `http://arxiv.org/abs/1506.00019`

[14] Kacprzyk, J.; Pedrycz, W. (editors). *Springer Handbook of Computational Intelligence*, chapter Artificial Neural Network Models. Springer-Verlag New York, 2015, ISBN 978-3-662-43504-5, pp. 455–471.

[15] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.*, volume 9, no. 8, November 1997: pp. 1735–1780, ISSN 0899-7667, doi:10.1162/neco.1997.9.8.1735, [Online]. Available from: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`

[16] Kusner, M.; Sun, Y.; et al. From Word Embeddings To Document Distances. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, edited by D. Blei; F. Bach, JMLR Workshop and Conference Proceedings, 2015, pp. 957–966, [Online]. Available from: `http://jmlr.org/proceedings/papers/v37/kusnerb15.pdf`

[17] Mikolov, T.; Sutskever, I.; et al. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, edited by C. J. C. Burges; L. Bottou; M. Welling; Z. Ghahramani; K. Q. Weinberger, Curran Associates, Inc., 2013, pp. 3111–3119, [Online]. Available from: `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`

[18] OWASP. *OWASP Top 10 - 2013*. OWASP Foundation, October 2013.

[19] Python Software Foundation. Python Success Stories. 2017, [Online; accessed: 2017-04-22]. Available from: `https://www.python.org/about/success/`

[20] Python Software Foundation. Integrating Python With Other Languages. April 2016, [Online; accessed: 2017-04-22]. Available from: `https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages`

[21] Django Software Foundation. Django (Version 1.11) [Computer Software]. 2017, [Online]. Available from: https://www.djangoproject.com/

[22] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, first edition, 2002, ISBN 978-0321127426.

[23] Holovaty, A.; Moss, J. K. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2007, ISBN 978-1590597255.

[24] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, [Online]. Available from: http://tensorflow.org/

[25] Chollet, F. Keras. 2015, [Online]. Available from: https://github.com/fchollet/keras

[26] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, volume abs/1605.02688, May 2016, [Online]. Available from: http://arxiv.org/abs/1605.02688

[27] Eby, P. PEP 3333 – Python Web Server Gateway Interface v1.0.1. September 2010, [Online; accessed: 2017-04-24]. Available from: https://www.python.org/dev/peps/pep-3333/

[28] Django Software Foundation. Security in Django. 2017, [Online; accessed: 2017-04-26]. Available from: https://docs.djangoproject.com/en/1.11/topics/security/

[29] Mozilla. Security/Server Side TLS. 2016, [Online; accessed: 2017-04-26]. Available from: https://wiki.mozilla.org/Security/Server_Side_TLS

[30] Lichman, M. UCI Machine Learning Repository. 2013, [Online]. Available from: http://archive.ics.uci.edu/ml

[31] Řehůřek, R.; Sojka, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, ELRA, May 2010, pp. 45–50, [Online]. Available from: http://is.muni.cz/publication/884893/en

[32] Karpathy, A. Multi-layer Recurrent Neural Networks (LSTM, GRU, RNN) for character-level language models in Torch. 2016, [Online; accessed: 2017-04-29]. Available from: https://github.com/karpathy/char-rnn

[33] Ozair, S. Multi-layer Recurrent Neural Networks (LSTM, RNN) for character-level language models in Python using Tensorflow. 2017, [Online; accessed: 2017-04-29]. Available from: https://github.com/sherjilozair/char-rnn-tensorflow

[34] Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks. May 2015, [Online; accessed: 2017-04-29]. Available from: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[35] Eidnes, L. Auto-Generating Clickbait With Recurrent Neural Networks. 2015, [Online; accessed: 2017-04-29]. Available from: https://larseidnes.com/2015/10/13/auto-generating-clickbait-with-recurrent-neural-networks/

[36] Maaten, L. v. d.; Hinton, G. E. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*, volume 9, 2008: pp. 2579–2605.

Appendix A

# UML diagrams

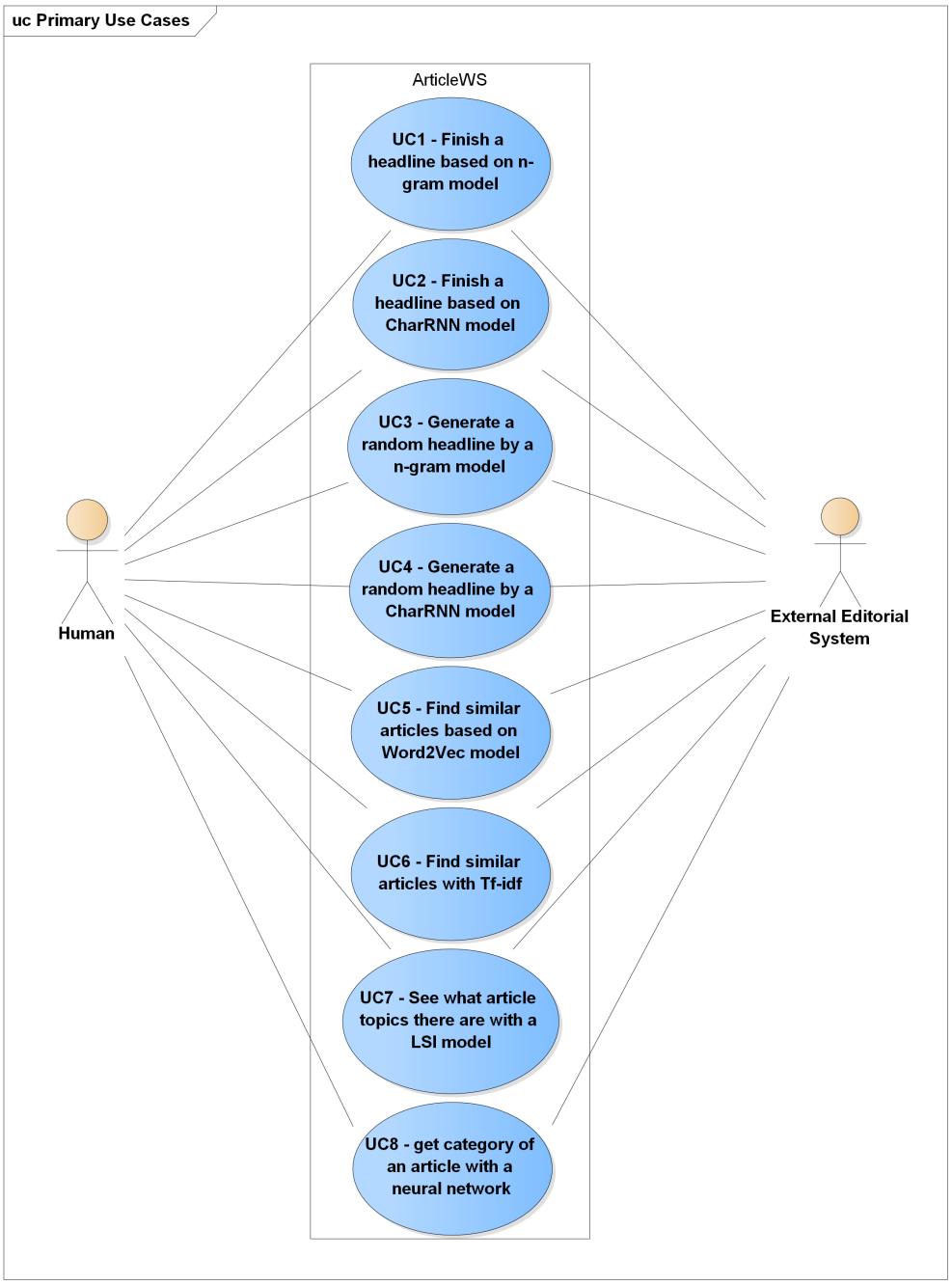



Figure A.1: Use case diagram of actors and primary use cases of the web service. Both actors can access the entire WS, but they are depicted separately to emphasize, that both humans and machines can use the system.

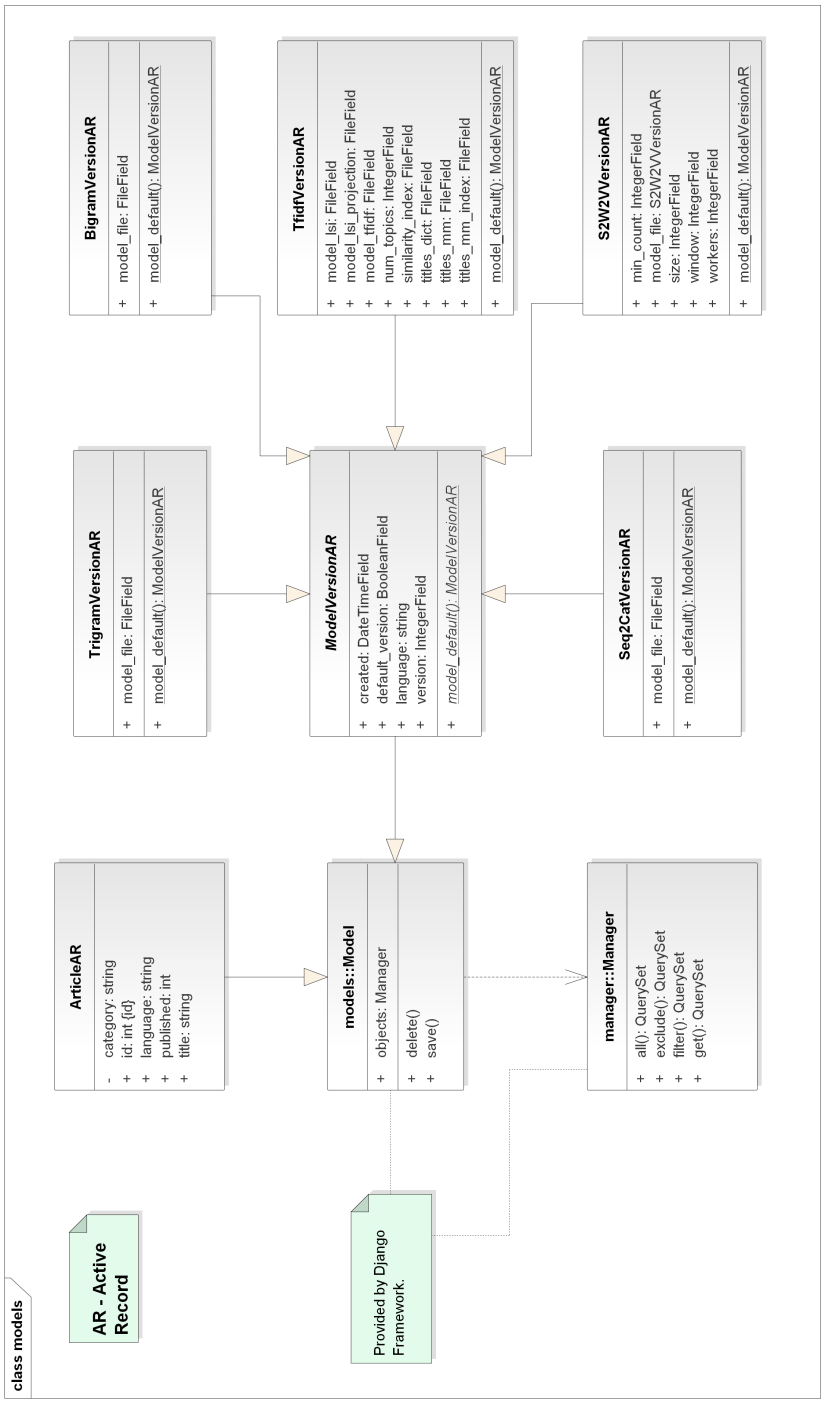



Figure A.2: Class diagram of the package *tech*. These classes are mapped into the database schema of the application.

Appendix B

# Data analysis and experiments

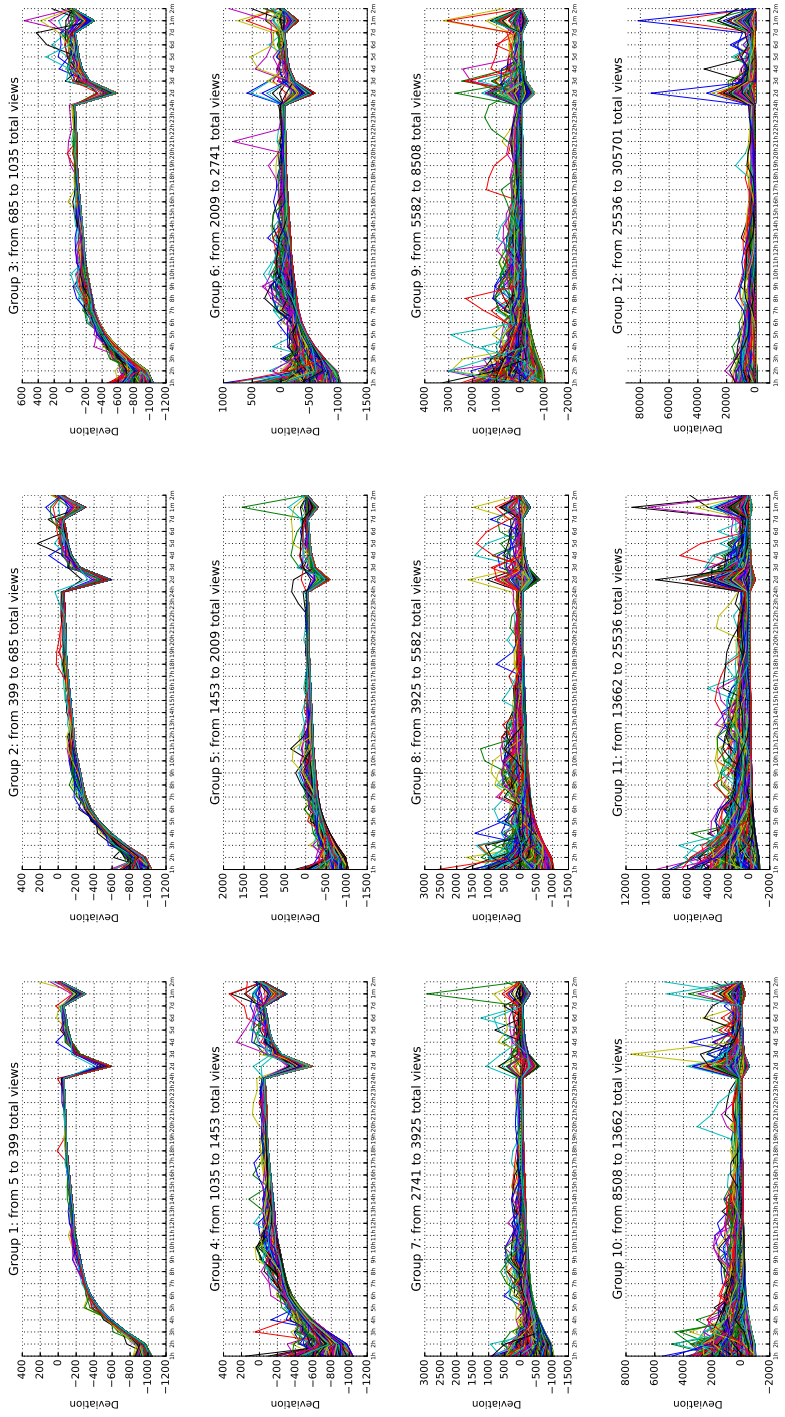



Figure B.1: This plot shows the number of views in time intervals. On axis $y$ is the deviation from the average in a given group. There is equal number of articles in each group.

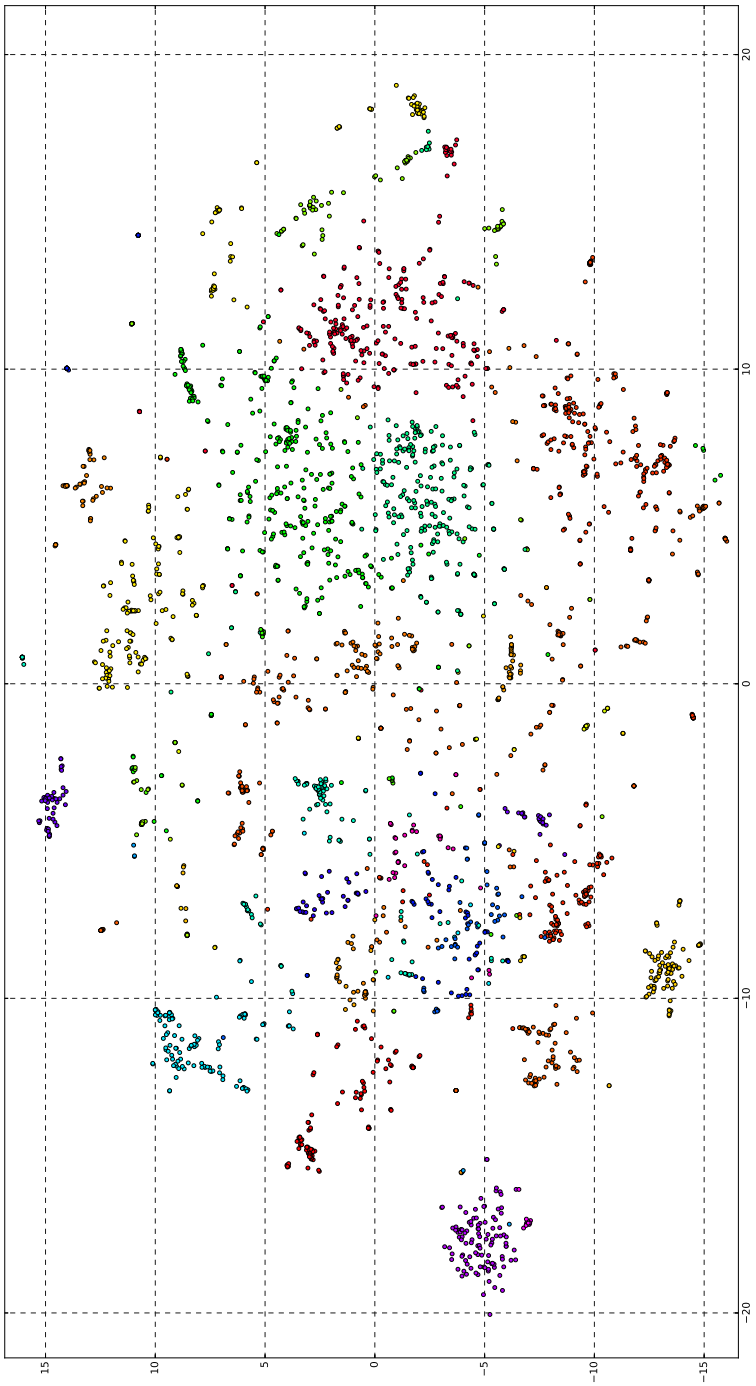



Figure B.2: A *t-SNE* visualization of embeddings acquired from the Char-RNN model.

Appendix C

# How to expand the functionality

To integrate new natural language processing algorithms into the web service, you need to do a few things according to what you want to do. Following are the instructions corresponding to that.

## C.1 Train a new version of an existing model

This is the easiest way of expanding the functionality. However, some models do not support this, as training them on a regular computer would take too long, sometimes even weeks.

1. Enter the Django shell with `python3 manage.py shell` and then import the model you want. Then train a new model by calling its method `model.train_anew(...)` on it. You could write a Python script for it, too.

2. To change its behaviour, use existing, or create new parameters for it. The model will store itself afterwards into the database (so if you want to test it later, do not forget to create a new fixture – see `README.md` in the project on how to do that).

3. You can make this newly trained version the default one. The application will load for the website and API the model with `is_default` attribute set to `true` in the database. You can also specify it directly when calling `train_anew()`.

## C.2 Create a new model engine for an existing functionality

1. To create a new way of delivering the same functionality as is already defined (for example similarity, category extraction and so on), create

a Python class inside `engine` package and inherit the desired interface from `engine_types.py` and the `SingletonMixin` class, so that it does not have to load data every time a new request needs to be processed.

2. To persist data, create a class in `tech/models.py` sub-classing the common `ModelVersionAR` class. This will enable you to use an *Active Record* class directly mapped to the DB. In order for the DB to be created, run `python3 manage.py makemigrations` and subsequently `python3 manage.py migrate` so that changes in the DB schema take effect.

3. Then implement all of the abstract methods. If you want to make it language independent, implement it like the `Bigram` model does or create your own way of achieving it.

4. Afterwards, you need to create a facade class in the `business` package, in order to access the model.

5. If you want to showcase the model, create a website base template for the functionality or use one from `webgui/templates/webgui/bases`, inherit from one of `statistical_base.html` or `machine_learning_base.html` and then create a template for the model in `webgui/templates/webgui`. You can use the prepared templates for results and errors, but feel free to make new ones:

   a) create links to the new model's web page in the sub-menu inside `website_base.html`,

   b) to transmit parameters from the website, use forms in templates and inherit from `View` in a class, which you will put into package `views`. Then provide serializers and take care of any exceptions and handling of the input/output,

   c) add the new `View` to `urls.py` and assign a path to it to let Django know of your new web page and view.

6. If you would like to make it accessible through the API, create an `APIView` subclass, that implements the desired REST methods, and use serializers to process input/output. Then:

   a) document the API in `webgui/static/docs/swagger.yaml`,

   b) test the API in `restapi/tests`.

## C.3 Create a new functionality

1. Inherit the interface `IEngineType` and create new interface methods defining what functionality you want the models to provide, but do not implement any of the methods yet. Then follow instructions in C.2.

# Appendix D

# Screenshots

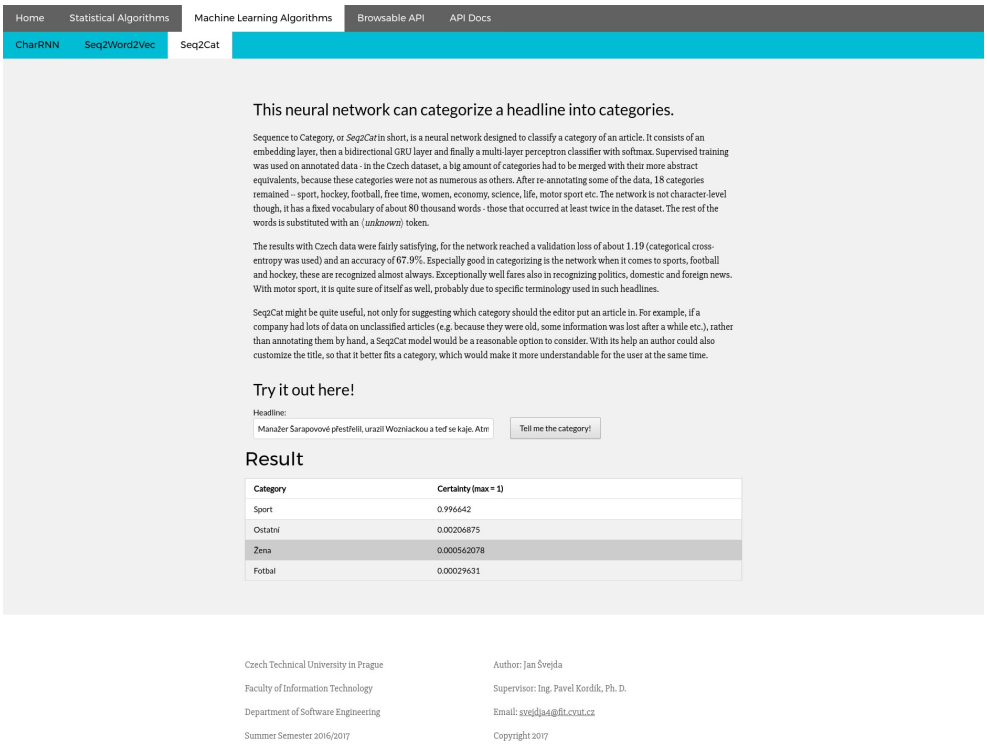



Figure D.1: Screen of the Seq2Cat model web page.

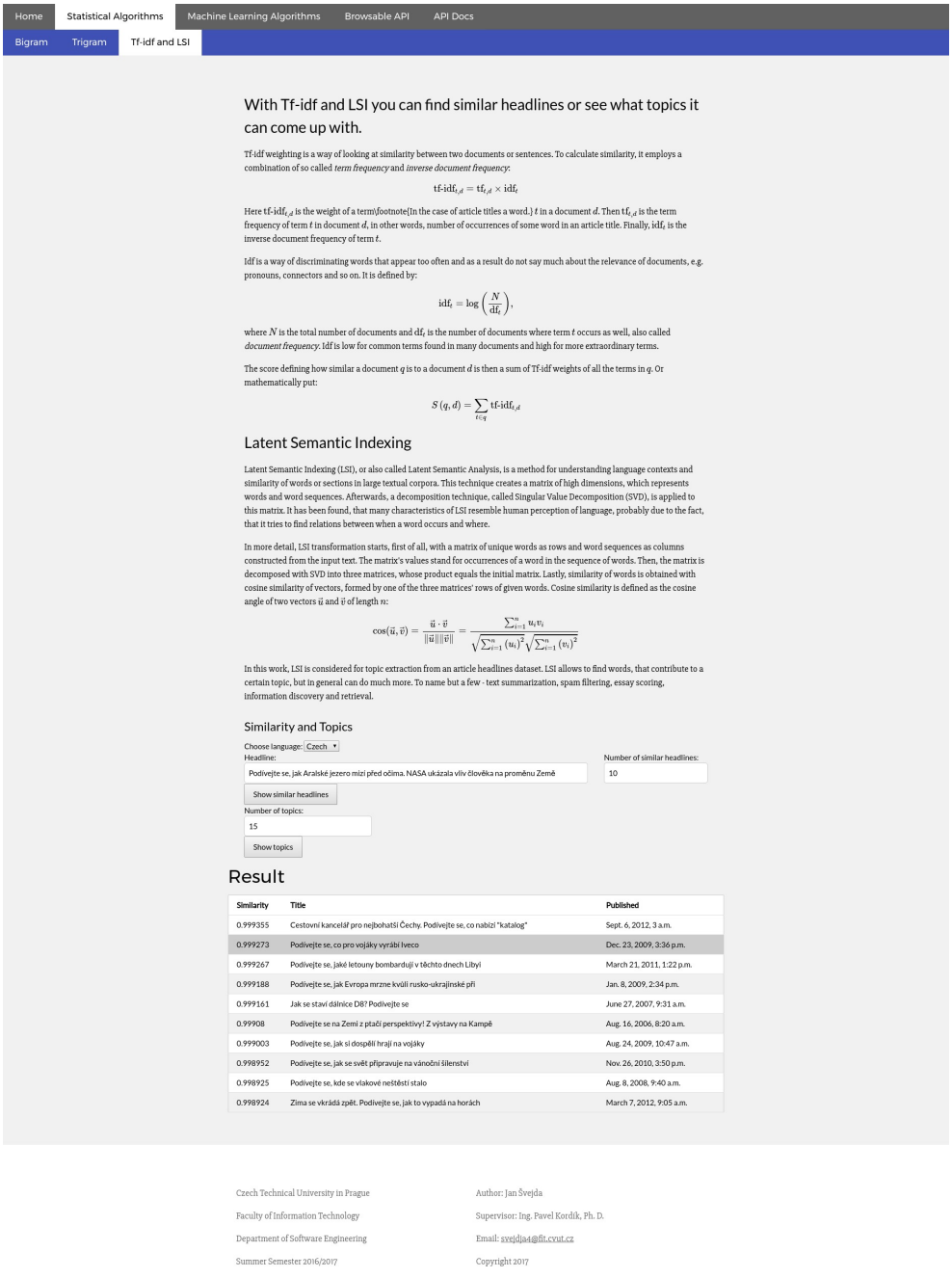



Figure D.2: Screen of the Tf-idf model web page.

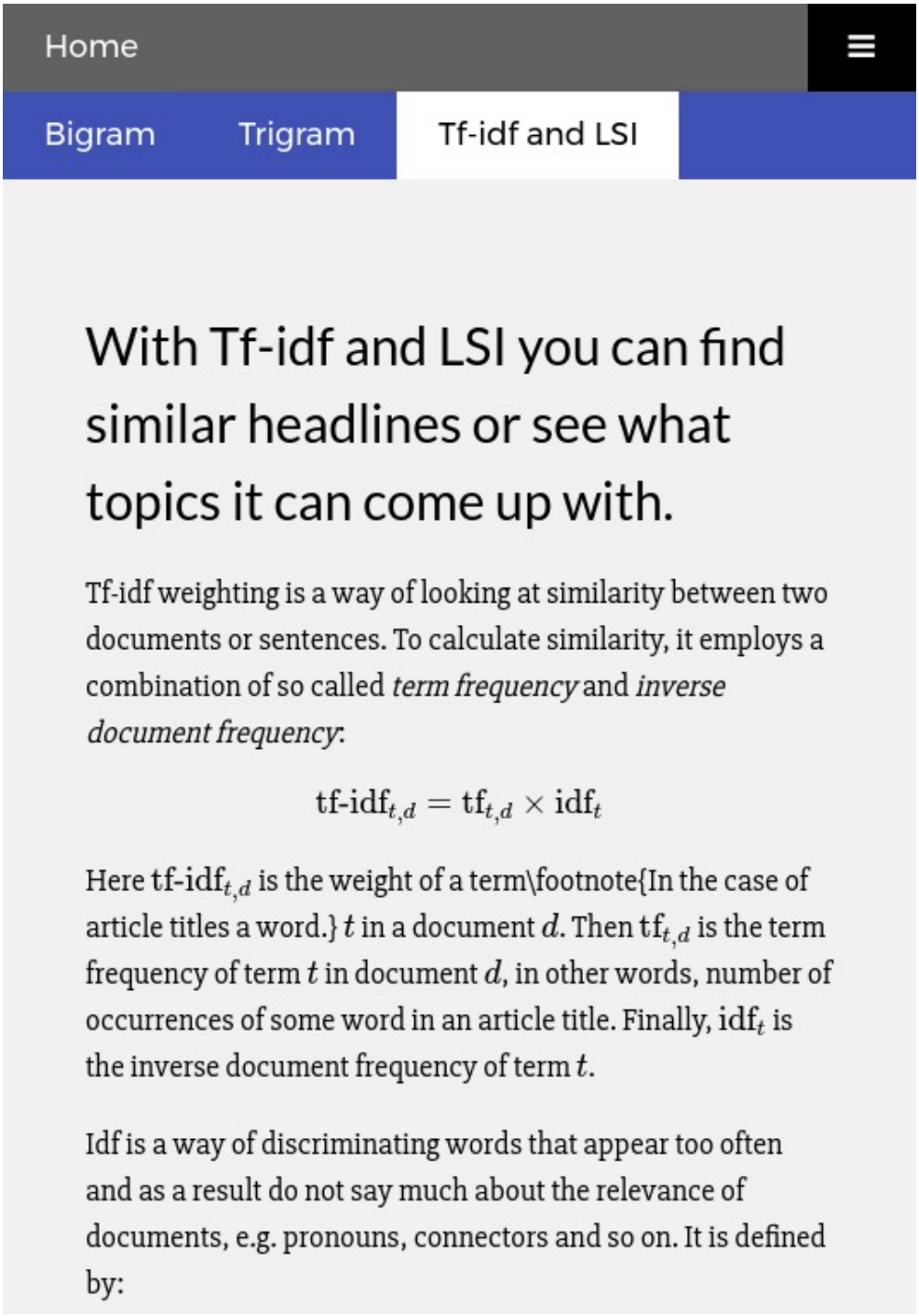



Figure D.3: Screen of Tf-idf page as it shows up on smaller width displays. All of the web pages are responsive similarly to this one.

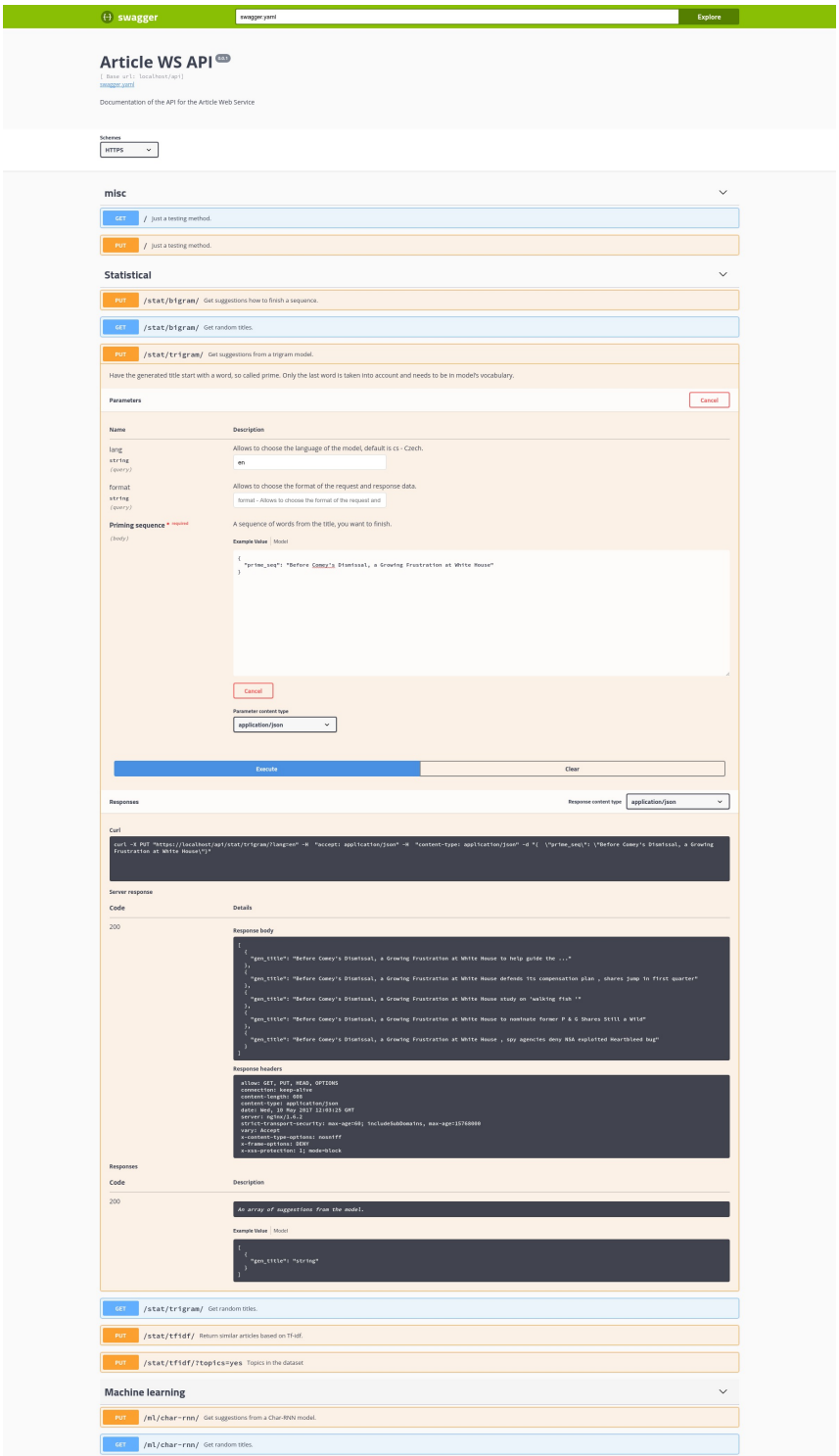



Figure D.4: Screen of Swagger documentation of the API. It is possible, as shown in the picture, to call the API methods from the documentation.

Appendix E

# Acronyms

**API** Application Programming Interface.

**CSRF** Cross-Site Request Forgery.

**DB** database.

**GRU** Gated Recurrent Unit.

**GUI** Graphical User Interface.

**HSTS** HTTP Strict Transport Security.

**HTML** HyperText Markup Language.

**HTTP** Hypertext Transfer Protocol.

**HTTPS** Hypertext Transfer Protocol Secure.

**JSF** JavaServer Faces.

**JSON** JavaScript Object Notation.

**LSI** Latent Semantic Indexing.

**LSTM** Long Short-Term Memory.

**ML** Machine Learning.

**MLP** Multilayer perceptron.

**MVC** Model View Controller.

**NLP** Natural Language Processing.

**OCSP** Online Certificate Status Protocol.

**REST** Representational State Transfer.

**RNN** Recurrent Neural Network.

**SMTP** Simple Mail Transfer Protocol.

**SSL** Secure Sockets Layer.

**SVD** Singular Value Decomposition.

**TCP** Transmission Control Protocol.

**Tf-idf** Term frequency-inverse document frequency.

**TLS** Transport Layer Security.

**URI** Uniform Resource Identifier.

**WDSL** Web Service Description Language.

**WS** web service.

**WSGI** Web Server Gateway Interface.

**XML** Extensible Markup Language.

**XSS** Cross-Site Scripting.

Appendix F

# Contents of enclosed CD

- `README.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `aws_project` ..... implementation sources
    - `aws_project` ..... Django project directory
    - `tech` ..... python sources for technical layer
    - `engine` ..... python sources for engine layer
    - `business` ..... python sources for business layer
    - `webgui` ..... python sources for the Web GUI
    - `restapi` ..... python sources for the REST API
    - `misc` ..... data, scripts and miscellaneous
    - `certs` ..... folder for certificates
    - `README.md` ..... explanation how to install and run
    - `aws_site.ini` ..... init file for uWSGI
    - `aws_nginx.conf` ..... configuration file for nginx
    - `secret_key.txt` ..... Django's secret key
    - `manage.py` ..... Django project management file
    - `requirements.txt` ..... python package requirements for pip3
    - `uwsgi_params` ..... configuration for uWSGI and nginx
    - `ArticleWS.EAP` ..... Enterprise Architect documentation file
  - `thesis` ..... the directory of LaTeX source codes of the thesis
    - `res` ..... resources for the thesis
      - `img` ..... images
      - `pdf` ..... screenshots and other
      - `bib` ..... bibliography files
    - `BP_Švejda_Jan_2017.pdf` .... the thesis text in LaTeX source code
- `text` ..... the thesis text directory
  - `BP_Švejda_Jan_2017.pdf` ..... the thesis text in PDF format