Project Ovcharka

Philipp Malkovsky

February 1, 2019

Abstract

Examination is an integral part of the education system. There are different ways of preparing for the exam. In this project we introduce a smart exam assistant intended for self-training in theoretical aspects of various subject areas. Given a list of related keywords it asks about some of them with clarifying questions additionally.

1 Introduction

The development of chat bots and the simplicity of their integration has led to a growth of the number of different testing systems. Many universities and companies have their own for training or testing students or staff.

However, unlike a real person, most often they provide only questions with a choice of an answer or can offer to enter a number or a short string. Usually they do not involve branching the conversation and ask questions sequentially or randomly. Moreover, writing a long text message is very different from an oral exam or an interview.

1.1 Intentions

The main goal of the project is to create a bot that asks key questions on the subject and is distracted by additional questions after the user answer. These can be either deeper questions related to the original one or words extracted from the user answer. Users are able to choose a subject and see their statistics.

It is also necessary to get a fair final grade, so the questions should have different weights. Also, the bot should indicate where a user is wrong. In addition, we would like to communicate with it verbally.

1.2 Ovcharka

The project features meet the needs specified above. It is a multi-user client-server application with a network connection. All data is stored on the server side. Each client can connect to server in different ways which are described further in the user guide.

2 User Guide

There are several ways to connect to Ovcharka. The first is a REST API that is part of the project. The second is a Telegram bot. Both of them could be used to communicate with Ovcharka in text form.

Both are described in the following sections.

The main feature of the application is the same for all connection methods. After starting the application a user can choose the subject from list and then begin training. All answers will affect further questions of Ovcharka. Incorrect answers may cause additional questions while right answers will shorten the training time. Additional questions can be based both on the relevance of the previous question and on the user answer. After the end of the exam you will be able to see your grade and correct answers.

2.1 REST API

An application is available via REST API that provides all the features: you can sign up and see the statistics of your trainings. It is described further in system overview section.

REST API provides flexibility in choosing a client including existing voice assistants or text bots. It also allows to build any custom frontend for our application.

2.2 Telegram Bot

(a) Telegram Bot

Figure 1: Use Case of the Application

We have implemented a Telegram bot client in order to demonstrate the simplicity of integrating clients into application. To interact with the application this way, you need to request an authentication token via REST API. Then just click the generated link which will

follow you to the bot and you will get all the functionality including training and viewing statistics.

Figure 1a demonstrates the training using Telegram bot.

3 System Overview

First, we will provide a high level overview of the architecture. Then, we will discuss all the parts of the application and their interaction.

3.1 Ovcharka Architecture Overview

In this section, we will give a high level overview of how the whole system works as pictured in Figure 2.

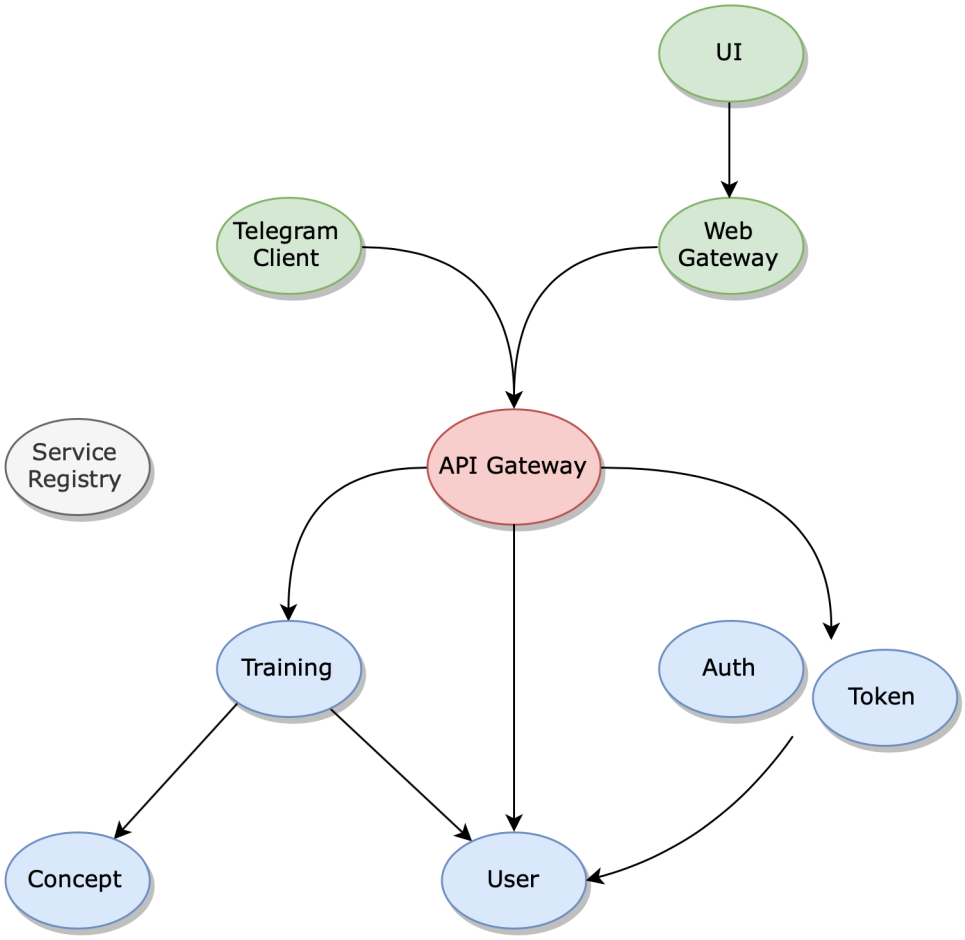

Figure 2: High Level Ovcharka Architecture

The project consists of Spring-based RESTful services interacting with each other via HTTP. The common formats of requests is described in the following section.

There is a Service Registry marked in grey that finds available instances of a service and tells other services where they are located.

Services marked in blue execute all the key tasks: perform algorithms on received data, store it and retrieve the requested information.

Concept service is responsible for the information about all the concepts currently supported.

User service stores all the information about any registered user of the application.

Training service stores the current state of a particular training of a particular user. It also performs user answer checking, choosing the next question and calculating the result.

Auth service is responsible for registering a new person and retrieving an authentication token for the logged in user.

Token service is used to create a token for authentication in external applications.

API Gateway marked in red is an entry point to the application and a proxy to the downstream services.

There are also some client services marked in green specific to the particular kind of connection.

There is Web Gateway that proxies the requests with the user details to the API Gateway. It checks the user rights for the specific request and redirects it down flow.

Telegram Client is a Telegram bot that makes sure that user has authenticated and proxies his requests in a downstream direction.

3.2 Request and Response Format

Services interact via HTTP protocol (basically GET and POST methods) in JSON format.

A request format is service-specific. For instance, it could be:

```
{
    "username": "john",
    "message": "\start"
}
```

All the responses have a common design. It consists of three fields: *status*, *data* and *message*.

If status is *success* then data is the desired information and message is *null*:

```
{
    "status": "success",
    "data": "FkpyY7dIneoIW2XS",
    "message": null
}
```

In such case the response code is 200 OK.

Otherwise, if status is *error*, data is null and a message contains error description:

```
{
    "status": "error",
    "data": null,
    "message": "No such user: mary"
}
```

If an error occurred due to an incorrect user request then response code is 400 Bad Request. Otherwise, it is 500 Internal Server Error.

If a user has no rights for the particular action then code is 401 Not Authorized.

3.3 Concept Service

Concept service stores data as a document with fields *id*, *word*, *definition*, *score* and *related* — list of related words sorted by relevance in descending order. The way the relevance is computed is discussed later in the Word Similarity subsection.

GET / — returns the list of all concepts

Response example:

```
{
    "status": "success",
    "data": [
        {
            "id": "5c51dff01b47ab1b0fbd498b",
            "word": "hardware",
            "definition": "the mechanical, magnetic, ...",
            "score": 3,
            "related": [
                "memory",
                "computer",
                "network",
                "packet",
                "bit"
            ]
        },
        ...
        {
            "id": "5c51dff01b47ab1b0fbd4997",
            "word": "Java",
            "definition": "a platform-independent ...",
            "score": 5,
            "related": [
                "C",
                "buffer",
                "map",
```

```
                "object",
                "bit"
            ]
        }
    ],
    "message": null
}
```


GET ?word= — returns a concept with a given word

Success response example:


```
{
    "status": "success",
    "data": {
        "id": "5c51dff01b47ab1b0fbd4997",
        "word": "Java",
        "definition": "a platform-independent ...",
        "score": 5,
        "related": [
            "C",
            "buffer",
            "map",
            "object",
            "bit"
        ]
    },
    "message": null
}
```


Error response example:


```
{
    "status": "error",
    "data": null,
    "message": "No such word: Kotlin"
}
```


GET /words — returns a list of all words

Response example:


```
{
    "status": "success",
    "data": [
        "bit",
```

```
            "buffer",
            "byte",
            ...
            "operating_system",
            "computing",
            "cipher"
        ],
        "message": null
    }
```

POST /words — deletes all concepts, inserts given and calculates related for all of them

Concepts update request example:

```
    {
        "concepts": [
            {
                "word": "biology",
                "definition": "the natural science...",
                "score": 4
            },
            {
                "word": "geography",
                "definition": "a field of science devoted to...",
                "score": 5
            }
        ]
    }
```

Response example:

```
    {
        "status": "success",
        "data": true,
        "message": null
    }
```

POST /word — if concept with this word is present then its fields are just updated, if not — total recalculation of related words occurs

Concept update request example:

```
    {
        "word": "biology",
        "definition": "the natural science...",
        "score": 4
```

```
}
```

Response example:

```
{
    "status": "success",
    "data": true,
    "message": null
}
```

3.4 User Service

User service stores data as a document with fields *id*, *name*, *username*, *password*, *role* and *stats*. Password should not be persisted as is, it should be hashed. Role can be equal to USER or ADMIN. Stats consists of fields *totalTrainings*, *passed*, *gpa*, where gpa is an average grade of all trainings for this user.

GET ?username= — returns all the information about a requested user

Success response example:

```
{
    "status": "success",
    "data": {
        "id": "5c51eb131b47ab1b867bb1da",
        "name": "John",
        "username": "john",
        "password": "h2a510eiJXkGhHzn8v...",
        "role": "ADMIN",
        "stats": {
            "totalTrainings": 4,
            "passed": 5,
            "gpa": 2.6
        }
    },
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
    "message": "No such user: mary"
}
```

The following methods have the same request format:
POST /create
POST /update
The first method checks if no user with a given username is already existed and after that creates a new user with a given parameters. The second updates user's information if there is a user with a given username.

User update request example:

```
{
    "name": "Mary",
    "username": "mary",
    "password": "ewj2jqkfq239f...",
    "role": "USER"
}
```

Response example:

```
{
    "status": "success",
    "data": true,
    "message": null
}
```

POST /stats — recalculates stats on each request given a new grade

Stats update request example:

```
{
    "username": "mary",
    "grade": "B"
}
```

Success response example:

```
{
    "status": "success",
    "data": true,
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
```

```
    "message": "No such user: mary"
}
```

3.5 Training

Training service stores data as a document with fields *id*, *userId*, *words*, *curWord*, *questionsLeft*, *score*, *debt* and *max*.

Each training entity exists if and only if a user has started the training and have not finished it yet. *Words* is a list of possible and available for this user words that have not been asked yet. *curWord* is the last question given to a user. *questionsLeft* denotes number of questions left to the end of the current training. *max* is the sum of all word scores asked by the moment, score is sum of scores of words for those a user gave a correct answer.

debt increases by the *curWord*'s score if a user gives a wrong answer and decreases by the same value if a user is right. If *debt* is lower than zero a user is asked relevant words rather than random ones.

The process is described in-depth in the Question Choosing and User Answer Checking sections.

The step-by-step example is provided in the Training Example section.

POST / — the only method in this service. If there is no trainings for a given user the only way to create it is to send a message "\start". It sets *curWord* to a random word and turns to zero *score*, *max* and *debt*. While training has not finished the "\start" command will only send a current word to be answered.

To force training to end at the current question it is needed to send a message "\end". It will count a grade and delete the current training. Otherwise, the current grade is count when *questionsLeft* becomes zero.

All the other messages are considered as an answer to the *curWord*.

Request examples:

```
{
    "username": "john",
    "message": "\start"
}
```

```
{
    "username": "john",
    "message": "/*user answer*/"
}
```

Success response example:

```
{
    "status": "success",
    "data": "Java",
```

```
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
    "message": "No current trainings for user with username: john"
}
```

3.6 Auth Service

Auth service is based on JSON Web Token (JWT).

POST /login — performs user authorization. It returns a unique access token for each user authorization and encodes in it his credentials. A user has to submit a token each time he requests a restricted endpoint.

Request example:

```
{
    "username": "john",
    "password": "password1"
}
```

In case of failure a user gets a 401 Not Authorized response. If a user exists and password is correct than the user gets 200 OK response with field *Authorization* in the header. Field value format is *Bearer* ⟨*token*⟩. The token has to be submitted for all requests that require authentication.

POST /signup — performs user registration. It checks if a user with this username is not present and sends a request to User service to create a new user with role USER.

Sign up request example:

```
{
    "name": "John",
    "username": "john",
    "password": "password1"
}
```

Success response example:

```
{
    "status": "success",
    "data": "true",
```

```
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
    "message": "This username is not available: john"
}
```

3.7 Token Service

Token is used for an authentication in external apps.

First, a user has to request a token via web app. Token is created and a key-value pair (token-username) is stored in Redis for one hour. After the token is expired a user has to request for a new one.

When a user sends a token to a client in external apps (for instance, to Telegram bot), the client asks Token service to return a username given a token if such pair is present. After that a client just has to save a mapping from a user id in external app (*chatId* in Telegram) to a username and convert to it each time it passes the queries to the server.

POST /generate — calls User service to check if a given name is present, generates a random token and stores it for one hour.

Request example:

```
{
    "username": "john"
}
```

Success response example:

```
{
    "status": "success",
    "data": "0jlASfN89CEON",
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
    "message": "No such user: john"
}
```

POST /confirm — if a given token is present return a username value for this token

Request example:

```
{
    "token": "0jlASfN89CEON"
}
```

Success response example:

```
{
    "status": "success",
    "data": "john",
    "message": null
}
```

Error response example:

```
{
    "status": "error",
    "data": null,
    "message": "No such token"
}
```

3.8 API Gateway

API Gateway is an entry point to the application server-side. It consumes all the requests and serves as a proxy to the downstream serves.

- /auth/** → Auth service
- /token/** → Token service
- /training → Training service
- /admin/users** → User service
- /admin/concepts** → Concept service

3.9 Web Gateway

Web Gateway serves as a proxy as well but it also performs an authentication. When a user is logged in he is given an access token. A user stores it on the client side and sends it in Authorization header when he wants to have an access to the restricted endpoints.

Web Gateway submits a token and proxies user request to API Gateway if it is correct or sends a 401 Not Authorized response to a user otherwise. The only endpoints available for not authenticated users are /login and /signup.

It also gets a username from a token to send a correct redirect to the API Gateway.

3.10 Telegram Client

Telegram Client proceeds all the user messages to the chat bot in Telegram.

To authenticate in Telegram bot with an existing account a user has to request a *token* in the web client. Then user should click a link *https://telegram.me/ovcharka_bot?start=token* or send a */start token* message to bot.

After that a user is fully authenticated and can pass trainings.

4 Algorithms Overview

In this section, the major parts of the application: the question choosing and the user answer checking will be examined.

4.1 Question Choosing

The natural form of a conversation during the exam is asking some questions on subject with related additional questions when given an inaccurate answer. When a user makes a mistake his *debt* increasing by the score of an answer and while his debt is more than 0 he will be asked additional questions. Giving a correct answer decreases a debt.

In order to simulate such a process we need to define a notion of relatedness between terms and store this metric for our questions. For this reason our question set is represented in form of a graph where nodes are questions themselves and edges are relatedness scores between them. Relatedness notion is discussed in the 4.1.1 subsection.

4.1.1 Word Similarity Notion

One of the key problems of natural language processing is a word sense representation and one of the aspects of this problem is the relationship between two words. The following notions are discussed in (Jurafsky and Martin 2009).

The first kind of relationships discussed here is word **similarity**. For instance, *river* is similar to *lake* while *lake* is not similar to *cat*.

The second notion is word **relatedness**. Consider the following example. While *cook* and *oven* have definitely different meanings they are obviously related (a cook uses an oven in his work).

The widely used method for evaluating these relationships is to use a thesaurus: a structured lexical database with words organized by meaning. The most popular thesaurus for English language is **WordNet**.

There are different algorithms to evaluate the relatedness between two words using WordNet. Their output is a number from 0 to 1 where 0 is for not related and 1 is for equivalent words. Those that we use are discussed further in 4.1.2 subsection.

4.1.2 Word Similarity Algorithms

We are currently using an algorithm proposed by Lin for a user answer matching. It is based on the probability that a randomly selected word in a corpus is an instance of concept and

on the lowest node in the hierarchy that subsumes both words.

The (extended) Lesk algorithm is used for determining the next word to ask. An intuition behind it is that two concepts are similar if their difinitions contain overlapping words.

Both algorithms are described in (Jurafsky and Martin 2009). We use (Shima 2013) implementation of these algorithms.

4.2 User Answer Checking

In order to rate a user we have to check each of his answers. As we store our questions with answers we need a way of comparing a given answer with predefined. The method we chose is based on **sentence similarity** metric which is discussed in 4.2.1 subsection.

4.2.1 Sentence Similarity Algorithm

One of the approaches to evaluate a sentence similarity is to extend a word similarity to the entire sentence. We use a vector cosine similarity based method described in (Liu and Wang 2013).

Consider two example sentences that are close by meaning:

S_1 : A software program that transforms high-level source code written by a developer in a high-level programming language into a low level object code (binary code) in machine language, which can be understood by the processor.

S_2 : A computer program that transforms computer code written in one programming language (the source language) into another programming language (the target language).

Preprocessing

First, we split our sentences into sets of lemmatized tokens. **Lemmatization** is a process of determining the lemma of a word which is the original form of a word. For instance, the words *wrote*, *written* and *writing* are forms of the verb *write*.

Moreover, we omit all the articles, prepositions, and conjunctions.

For our sentences these sets are:

T_1 : [code, software, level, understand, language, program, source, high-level, processor, can, transform, low, machine, binary, developer, write, programming, object];

T_2 : [code, one, another, language, program, source, target, computer, transform, write, programming].

We also form a union of two sets:

$T = T_1 \bigcup T_2$: [code, software, level, one, another, understand, language, program, source, high-level, processor, target, can, transform, computer, low, machine, binary, developer, write, programming, object].

Semantic Vector forming

Second, we construct the semantics vectors for T_1 and T_2 with dimension equal the size of T. For each word w in T we compute a similarity for w and each word in T_1 using an algorithm from the discussed in 4.1.2 subsection. The most similar to w word is used to obtain a corresponding vector entry value. Similarly, the second vector is constructed for T_2.

The most similar pairs of words with similarity values for example sentences are shown in Table 1. The actual values of vectors are:

$\vec{V_1}$ = [1.0, 1.0, 1.0, 0.558, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.837, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0];

$\vec{V_2}$ = [1.0, 0.849, 0.840, 1.0, 1.0, 0.310, 1.0, 1.0, 1.0, 0.0, 0.088, 1.0, 0.390, 1.0, 1.0, 0.0, 0.837, 0.128, 0.619, 1.0, 1.0, 1.0].

Table 1: Entries for Semantic Vectors

[(code, code): 1.0,	[(code, code): 1.0,
(software, software): 1.0,	(software, code): 0.849,
(level, level): 1.0,	(level, target): 0.840,
(one, program): 0.558,	(one, one): 1.0,
(another, software): 0.0,	(another, another): 1.0,
(understand, understand): 1.0,	(understand, program): 0.311,
(language, language): 1.0,	(language, language): 1.0,
(program, program): 1.0,	(program, program): 1.0,
(source, source): 1.0,	(source, source): 1.0,
(high-level, high-level): 1.0,	(high-level, computer): 0.0,
(processor, processor): 1.0,	(processor, program): 0.088,
(target, object): 1.0,	(target, target): 1.0,
(can, can): 1.0,	(can, write): 0.390,
(transform, transform): 1.0,	(transform, transform): 1.0,
(computer, machine): 0.837,	(computer, computer): 1.0,
(low, low): 1.0,	(low, computer): 0.0,
(machine, machine): 1.0,	(machine, computer): 0.837,
(binary, binary): 1.0,	(binary, source): 0.128,
(developer, developer): 1.0,	(developer, source): 0.619,
(write, write): 1.0,	(write, write): 1.0,
(programming, programming): 1.0,	(programming, programming): 1.0,
(object, object): 1.0]	(object, target): 1.0]

Calculating Similarity

Finally, the cosine similarity between vectors is computed using formula 1:

$$similarity(S_1, S_2) = \frac{\vec{V_1} \cdot \vec{V_2}}{\|\vec{V_1}\|\|\vec{V_1}\|} \tag{1}$$

For example sentences $similarity(S_1, S_2) = 0.841$.

5 Step by Step Example of Training

Here we will give an example of a training assuming that it consists of 5 questions. If question had been already asked during a training it will be marked grey.

Step 1

The state: score = 0, max = 0, debt = 0.

Debt is 0 so the next question is picked randomly. It is "Java" which has score 5.

Predefined answer is "A general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible."

User answer is "A high-level, interpreted programming language that conforms to the ECMAScript specification. It is a language that is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm."

Similarity between sentences is 0.547 so the answer is wrong.

Step 2

The state: score = 0, max = 5, debt = 5.

Debt is more than 0 so the next question is chosen as most relevant. It is "C" which has score 3.

Predefined answer is "A general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations."

User answer is "A general-purpose programming language, supporting structured programming, with a static type system that prevents many unintended operations."

Similarity between sentences is 0.966 so the answer is right.

Step 3

The state: score = 3, max = 8, debt = 2.

Debt is more than 0 so the next question is chosen as most relevant. It is "Compiler" which has score 4.

Predefined answer is "A software program that transforms high-level source code that is written by a developer in a high-level programming language into a low level object code (binary code) in machine language, which can be understood by the processor."

User answer is "A computer program that transforms computer code written in one programming language (the source language) into another programming language (the target language)."

Similarity between sentences is 0.841 so the answer is right.

Step 4

The state: score = 7, max = 12, debt = 0.

Debt is 0 so the next question is picked randomly. It is "Stack" which has score 3.

Predefined answer is "In computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed."

User answer is "A geological landform consisting of a steep and often vertical column or columns of rock in the sea near a coast, formed by wave erosion."

Similarity between sentences is 0.294 so the answer is wrong.

Step 5

The state: score = 7, max = 15, debt = 3.

Debt is more than 0 so the next question is chosen as most relevant. It is "Map" which has score 4.

Predefined answer is "An abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection."

User answer is "An abstract data type that stores data in the form of key and value pairs where every key is unique."

Similarity between sentences is 0.806 so the answer is right.

Final grade

The state: score = 11, max = 19, debt = 0.

$\frac{score}{max} = \frac{11}{19} \approx 0.58$. It is more than 50% of correct answers so the final grade is C.

References

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0131873210.

Liu, Hongzhe and Pengfei Wang (June 2013). "Assessing Sentence Similarity Using WordNet based Word Similarity". In: *Journal of Software* 8. DOI: 10.4304/jsw.8.6.1451-1458.

Shima, Hideki (2013). *WordNet Similarity for Java*. URL: https://code.google.com/archive/p/ws4j/ (visited on 02/01/2019).