

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА № 6**

**по дисциплине**

**«Проектирование интеллектуальных систем (часть ½)»**

Студент группы: ИКБО-04-21 Даурбеков М.И. *(Ф.И.О.студента)*

Руководитель Холмогоров В.В.

*(Ф.И.О.преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc167296791)

[1 ТЕОРИТИЧЕСКИЙ РАЗДЕЛ 4](#_Toc167296792)

[1.1 Описание предметной области 4](#_Toc167296793)

[1.2 Существующие подходы 4](#_Toc167296794)

[1.3 Метод решения 8](#_Toc167296795)

[2 АНАЛИТИЧЕСКИЙ РАЗДЕЛ 10](#_Toc167296796)

[2.1 Алгоритмический и математический анализ 10](#_Toc167296798)

[3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 11](#_Toc167296799)

[3.1 Описание программных сущностей 11](#_Toc167296801)

[3.2 Тестирование 11](#_Toc167296802)

[ЗАКЛЮЧЕНИЕ 12](#_Toc167296803)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc167296804)

[ПРИЛОЖЕНИЯ 14](#_Toc167296805)

# 

# АННОТАЦИЯ

В данной работе описано создание back-end составляющей для рекомендательной системы по подбору образовательных курсов во время участия в хакатоне от цифрового прорыва, который был проведен в Центральном федеральном округе.

# ВВЕДЕНИЕ

Сегодня пользователям сложно ориентироваться в большом числе обучающих курсов на образовательных платформах, а также прорабатывать свою личную траекторию развития для получения необходимого уровня знаний, соответствующего желаемой профессии.

Создание MVP на основе искусственного интеллекта, который сможет определить, какую программу обучения стоит изучить, может значительно упростить процесс трудоустройства. Потенциальные сотрудники будут понимать, какие им необходимы навыки и знания для успешного трудоустройства на заинтересовавшую вакансию.

# ТЕОРИТИЧЕСКИЙ РАЗДЕЛ

* 1. Постановка задачи

Необходимо разработать MVP на основе искусственного интеллекта, который сможет определить, какую программу обучения стоит изучить, чтобы приобрести необходимые навыки и знания для успешного трудоустройства. Решение должно анализировать требования, предъявляемые кандидатам в вакансиях, и предлагать персонализированные рекомендации образовательных курсов, продуктов или других обучающих материалов, соответствующих потребностям рынка труда.

* 1. Существующие подходы

На данный момент каждый желающий трудоустроится самостоятельно ищет подходящую вакансию, зачастую ошибаясь в необходимых навыках и знаниях, тем самым тратя свое время и время работодателя.

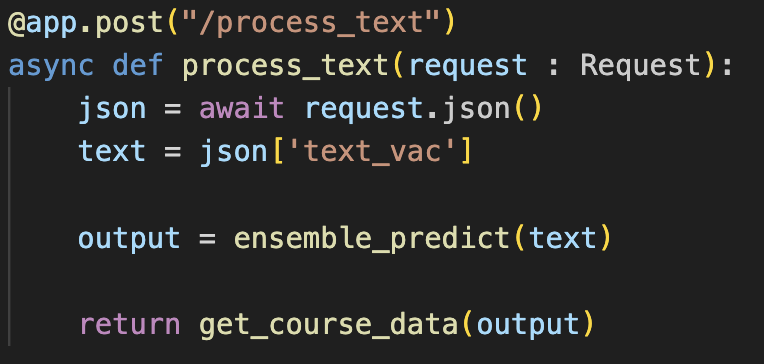
# АНАЛИТИЧЕСКИЙ РАЗДЕЛ

1. 1. Алгоритмический анализ

В ходе участия хакатона был разработан back-end на ЯП python с помощью фрейворка FastAPI. который имеет 3 метода, которые возвращают информацию о наиболее подходящем по описанию курсе на портале GeekBrains.

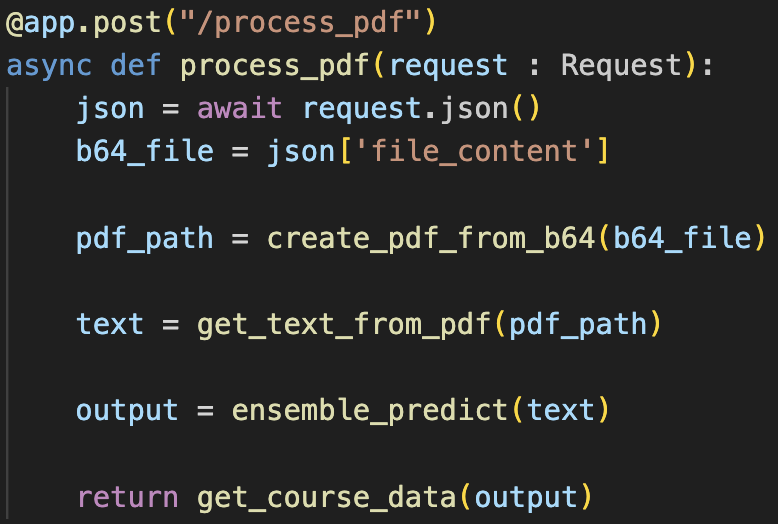
Методы:

1. /process\_text
2. /process\_pdf
3. /process\_url

****

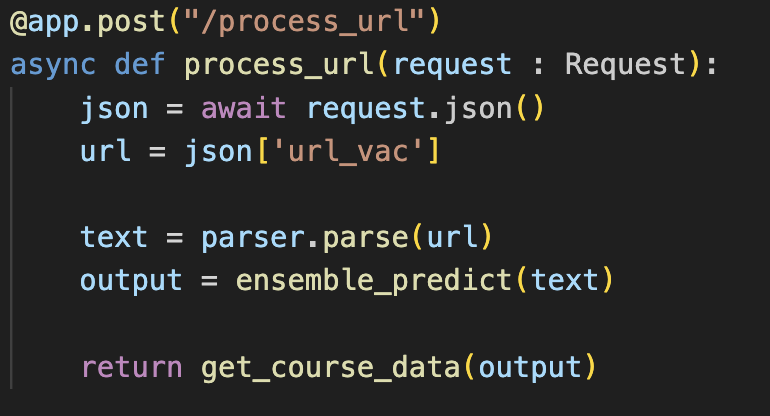
**Рисунок 1 – Метод /process\_text**

Метод /process\_text не имеет никакой предобработки входящий данных, а сразу отправляет текст как описание вакансии для обработки Bert моделью.

****

**Рисунок 2 – Метод /process\_pdf**

Метод /process\_pdf сперва сохраняет входящий поток файла в формате Base64 в временную папку в формате pdf и с помощью библиотеки py\_pdf\_parser достает текстовое содержание файла, которое отправляется в Bert модель.

****

**Рисунок 3 – Метод /prcess\_url**

Метод /process\_url получает на вход ссылку на вакансию на портале hh.ru, алгоритм обрабатывает страницу с помощью библиотеки selenium и находит на странице вакансии описание требований от разработчика. Это описание отправляется в модель Bert для поиска наилучшего курса на портале GeekBrains.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы было осуществлено знакомство с фреймворком FastAPI и написание рабочего веб-сервиса, который предоставляет наиболее подходящие курсы обучения. Были получены навыки работы в команде и публичной презентации продукта. Подтверждение участия представлено в приложении Б.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сорокин А. Б. Свёрточные нейронные сети: примеры реализаций – Москва: МИРЭА, 2020.

2. Цифровой прорыв - https://hacks-ai.ru/

3. FastAPI - https://fastapi.tiangolo.com/

4. Заказчик - https://gb.ru

# ПРИЛОЖЕНИЯ

Приложение А – Исходный код

Приложение Б – Сертификат

**Приложение А**

Исходный код

*Листинг 1 – код main.py*

from fastapi import FastAPI, Request

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel

from pdf import get\_text\_from\_pdf, create\_pdf\_from\_b64

from hh\_parse import Parser

from text\_classifier import ensemble\_predict

from course\_config import get\_course\_data

parser = Parser()

def lifespan(app: FastAPI):

global parser

yield

del parser

app = FastAPI(lifespan=lifespan)

app.add\_middleware( CORSMiddleware, allow\_origins=['\*'] )

@app.post("/process\_text")

async def process\_text(request : Request):

json = await request.json()

text = json['text\_vac']

output = ensemble\_predict(text)

return get\_course\_data(output)

@app.post("/process\_pdf")

async def process\_pdf(request : Request):

json = await request.json()

b64\_file = json['file\_content']

pdf\_path = create\_pdf\_from\_b64(b64\_file)

text = get\_text\_from\_pdf(pdf\_path)

output = ensemble\_predict(text)

return get\_course\_data(output)

@app.post("/process\_url")

async def process\_url(request : Request):

json = await request.json()

url = json['url\_vac']

text = parser.parse(url)

output = ensemble\_predict(text)

return get\_course\_data(output)

*Листинг 2 – код pdf.py*

from pathlib import Path

from py\_pdf\_parser.loaders import load\_file as LoadPDF

from base64 import b64decode

pdf\_directory\_path = "pdf\_buffer/"

Path(pdf\_directory\_path).mkdir(exist\_ok=True)

def get\_text\_from\_pdf(file\_path: str):

pdf = LoadPDF(file\_path)

return ' '.join([element.text() for element in pdf.elements])

def create\_pdf\_from\_b64(bytes, file\_name: str = 'file.pdf'):

file\_path = pdf\_directory\_path + file\_name

with open(file\_path, "wb") as file:

file.write(b64decode(bytes))

return file\_path

*Листинг 3 – код text\_classifier*

import torch

import torch.nn as nn

from transformers import BertModel, BertTokenizer

import pickle

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import pandas as pd

import nltk

from nltk.corpus import stopwords

import re

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

class TextClassifier(nn.Module):

def \_\_init\_\_(self, n\_classes):

super(TextClassifier, self).\_\_init\_\_()

self.bert = BertModel.from\_pretrained('DeepPavlov/rubert-base-cased', return\_dict=False)

self.dropout = nn.Dropout(p=0.1)

self.classifier = nn.Linear(self.bert.config.hidden\_size, n\_classes)

def forward(self, input\_ids, attention\_mask):

outputs = self.bert(input\_ids, attention\_mask=attention\_mask)

output = self.dropout(outputs[1])

return self.classifier(output)

n\_classes = 12

model = TextClassifier(n\_classes)

model.load\_state\_dict(torch.load('model\_CFO\_sd.pth', map\_location=device))

model.to(device)

model.eval()

with open('label\_encoder.pkl', 'rb') as file:

le = pickle.load(file)

def clean\_text(text):

text = re.sub(r'[^\w\s]', ' ', text)

text = re.sub(r'\s+', ' ', text).lower()

words = text.split()

words = [word for word in words if word not in stop\_words]

return ' '.join(words)

nltk.download('stopwords')

stop\_words = set(stopwords.words('russian'))

tokenizer = BertTokenizer.from\_pretrained('DeepPavlov/rubert-base-cased')

datasetTMP = pd.read\_excel('datasetTMP.xlsx')

datasetTMP = datasetTMP.drop(columns="Название продукта")

tfidf = TfidfVectorizer()

tfidf\_matrix = tfidf.fit\_transform(datasetTMP['Описание'])

def recommend\_product(user\_description, dataset):

tfidf\_desc = TfidfVectorizer()

tfidf\_matrix\_desc = tfidf\_desc.fit\_transform(dataset["Описание"])

cleaned\_query = clean\_text(user\_description)

query\_vector\_desc = tfidf\_desc.transform([cleaned\_query])

cos\_sim\_desc = cosine\_similarity(query\_vector\_desc, tfidf\_matrix\_desc)

enhanced\_cos\_sim = cos\_sim\_desc \* 2 # Увеличиваем влияние TF-IDF

best\_idx = enhanced\_cos\_sim.argmax()

return dataset.iloc[best\_idx]['Ссылка на продукт']

def predict\_url(description):

encoded = tokenizer.encode\_plus(

description, add\_special\_tokens=True, max\_length=512,

padding='max\_length', truncation=True, return\_attention\_mask=True,

return\_tensors='pt')

input\_ids = encoded['input\_ids'].to(device)

attention\_mask = encoded['attention\_mask'].to(device)

with torch.no\_grad():

outputs = model(input\_ids, attention\_mask)

\_, predicted\_index = torch.max(outputs, dim=1)

return le.inverse\_transform([predicted\_index.item()])[0]

def ensemble\_predict(description):

predicted\_url = predict\_url(description)

recommended\_product = recommend\_product(description, datasetTMP)

return predicted\_url if predicted\_url == recommended\_product else recommended\_product

*Листинг 4 – код hh\_parse.py*

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.common.exceptions import TimeoutException

import validators

xpath\_vacancy\_archive\_button = '//\*[@id="HH-React-Root"]/div/div/div[4]/div[1]/div/div/div/div/div/div[3]/div/button'

xpath\_vacancy\_description = "//div[@data-qa='vacancy-description']"

class Parser():

def \_\_init\_\_(self):

options = webdriver.SafariOptions()

self.driver = webdriver.Safari(options=options)

def \_\_del\_\_(self):

self.driver.quit()

def parse(self, url : str):

if not validators.url(url):

raise Exception(f'invalid url: {url}')

try:

self.driver.get(url)

except:

raise Exception(f'бляяяя')

try:

button = self.driver.find\_element(By.XPATH, xpath\_vacancy\_archive\_button)

button.click()

except:

pass

try:

description = self.driver.find\_element(By.XPATH, xpath\_vacancy\_description)

return description.text

except:

raise Exception(f'could not find vacancy description at {url}')

*Листинг 5 – код course\_config.py*

import pandas as pd

courses = pd.read\_csv('course\_data.csv')

def get\_course\_data(course\_link : str):

info = courses.loc[courses['Ссылка на продукт'] == course\_link].iloc[0]

return {

'parsedBody': {

'url\_vac': info['Ссылка на продукт'],

'time': info['Длительность'],

'cost': info['стоимость обучения'],

'forma': info['Формат обучения'],

'desc': info['Профессия'],

}

}

**Приложение Б**

Сертификат

