Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

Лабораторная работа №4

«Технологии машинного обучения в облачной системе Yandex Cloud»

ДИСЦИПЛИНА: «Облачные технологии»

Выполнил: студент гр. ИУК4-82Б _____ (Сафронов Н.С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Ameличев Г.Э.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель работы: приобретение практических навыков по работе с сервисом Yandex Cloud.

Постановка задачи

Решите задачу согласно варианту с использованием платформы Yandex Cloud.

Вариант 4

С помощью сервиса Yandex Cloud Translate перевести на русский язык машинно-сгенерированный текст на английском языке.

Результаты выполнения работы

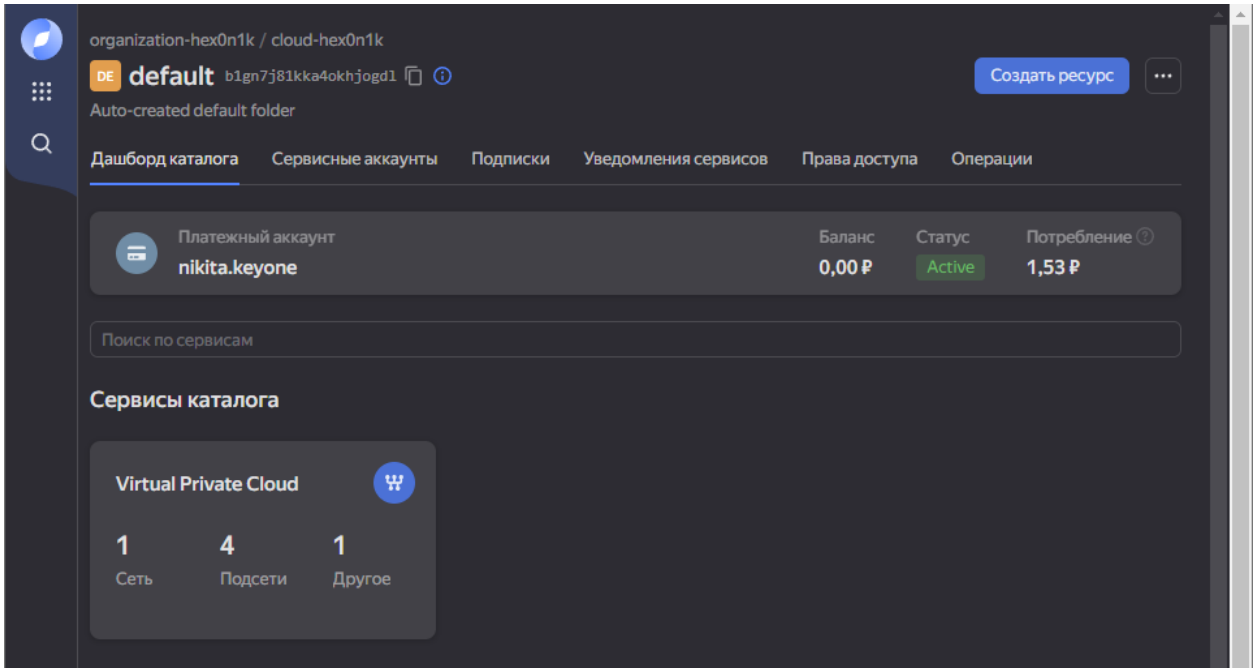

Рисунок 1 – Созданная учётная запись

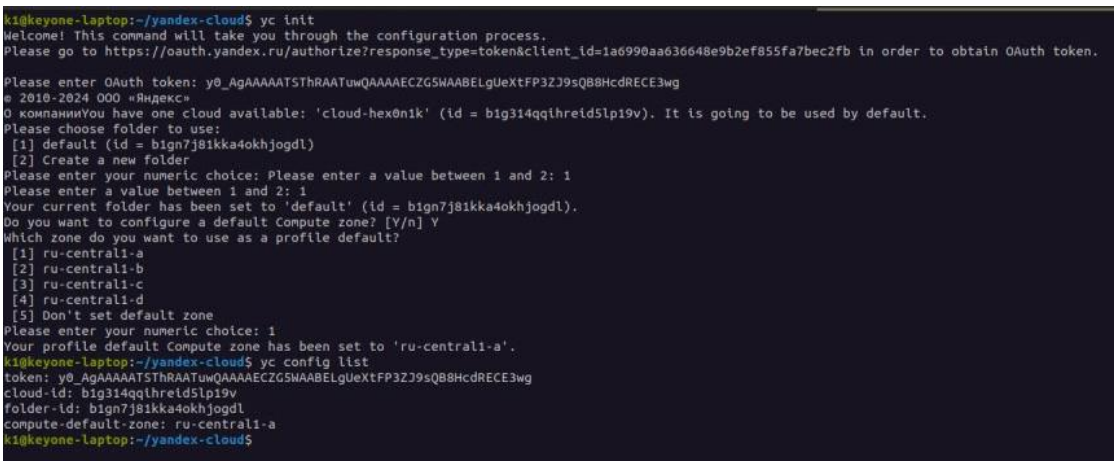

Рисунок 2 – Создание профиля

Листинг программы

Файл main.py:

```
from wrappers.cli import CliWrapper
from services.translate import TranslationService
from services.gpt import GptService
import sys


if __name__ == '__main__':
    cli_wrapper = CliWrapper()
    config = cli_wrapper.parse_config(sys.argv[1:])

    gpt_service = GptService(
        token=config['token'],
        folder_id=config['folder_id'],
        api_url=config['gpt_api_url']
    )

    print(f'[GPT] Started generation for prompt
{config["prompt"]}...')
    text = gpt_service.get_generated_text(config['prompt'])
    print('[GPT] Text was successfully generated:', text)

    translations_service = TranslationService(
        token=config['token'],
        target_language=config['target_language'],
        folder_id=config['folder_id'],
        api_url=config['translations_api_url']
    )

    print(
        f'[Translation] Started translation to
{config["target_language"]}',
        'for generated text...'
    )
    translation = translations_service.get_translation(text)
```

```
    print('[Translation] Text was successfully translated:',
translation)
```

Файл models/base.py:

```
import typing as t

ModelType = t.TypeVar('ModelType')


def load_model(cls: t.Type[ModelType], data: t.Mapping) ->
ModelType:
    required = cls.__dict__.get('__required_keys__',
frozenset())
    optional = cls.__dict__.get('__optional_keys__',
frozenset())

    model = cls(
        **{key: data[key] for key in required},
        **{key: data[key] for key in optional if key in data},
    )

    return model
```

Файл models/config.py:

```
""".""
import os
import typing as t


class Config(t.TypedDict):
    """Конфигурация для сервиса перевода Yandex Cloud"""
    token: str
    folder_id: str
    target_language: str
    translations_api_url: str
    gpt_api_url: str
    prompt: str


default_config = Config(
    token=os.environ.get('YC_TOKEN', ''),
    folder_id=os.environ.get('YC_FOLDER_ID', ''),
    target_language=os.environ.get('YC_FOLDER_ID', ''),

translations_api_url='https://translate.api.cloud.yandex.net/tra
nslate/v2/'
                         'translate',

gpt_api_url='https://llm.api.cloud.yandex.net/foundationModels/v
1/'
                'completion',
    prompt=os.environ.get('YC_GPT_PROMPT', ''),
)
```

Файл models/translation.py:

```
"""."""
import typing as t


class Translation(t.TypedDict):
    """."""
    text: str
    detectedLanguageCode: str
```

Файл services/gpt.py:

```
import requests
import logging
from models.translation import Translation
from models.base import load_model


class GptService:
    """."""
    def __init__(
        self,
        token: str,
        folder_id: str,
        api_url: str,
    ):
        self._token = token
        self._folder_id = folder_id
        self._api_url = api_url
        self._logger = logging.Logger('GPT')

    def _request_generation(self, prompt: str) -> dict:
        body = {
            "modelUri": f"gpt://{self._folder_id}/yandexgpt-
lite/latest",
            "completionOptions": {
                "maxTokens": 500,
            },
            "messages": [
                {
                    "role": "user",
                    "text": prompt,
                },
            ]
        }

        headers = {
            'Content-Type': 'application/json',
            'Authorization': f'Bearer {self._token}',
        }

        response = requests.post(
            url=self._api_url,
```

```
            json=body,
            headers=headers,
            timeout=5,
        )

        if not response.ok:
            self._logger.error(
                f'Unable to generate text,
code={response.status_code}'
            )
            raise RuntimeError('Unable to generate')

        return response.json()

    def get_generated_text(self, prompt: str) -> dict:
        response = self._request_generation(prompt)
        single_result =
response.get('result').get('alternatives')[0]

        return single_result.get('message').get('text')
```

Файл services/translate.py:

```
import requests
import logging
from models.translation import Translation
from models.base import load_model


class TranslationService:
    """."""
    def __init__(
        self,
        token: str,
        folder_id: str,
        target_language: str,
        api_url: str,
    ):
        self._token = token
        self._folder_id = folder_id
        self._target_language = target_language
        self._api_url = api_url
        self._logger = logging.Logger('Translate')

    def _request_translate(self, text: str) -> dict:
        body = {
            'targetLanguageCode': self._target_language,
            'folderId': self._folder_id,
            'texts': text,
        }

        headers = {
            'Content-Type': 'application/json',
            'Authorization': f'Bearer {self._token}',
```

```
        }

        response = requests.post(
            url=self._api_url,
            json=body,
            headers=headers,
            timeout=5,
        )

        if not response.ok:
            self._logger.error(
                f'Unable to generate text,
code={response.status_code}'
            )
            raise RuntimeError('Unable to translate')

        return response.json()

    def get_translations(self, text: str) -> list[Translation]:
        raw_translations =
self._request_translate(text).get('translations')

        return [load_model(Translation, x) for x in
raw_translations]

    def get_translation(self, text: str) -> Translation:
        return self.get_translations(text)[0].get('text')
```

Файл wrappers/cli.py:

```
import argparse
import copy
from models.config import default_config, Config


class CliWrapper:
    def _init_parser(self) -> argparse.ArgumentParser:
        parser = argparse.ArgumentParser(
            description='Yandex Cloud Translations API'
        )

        parser.add_argument('prompt', type=str)
        parser.add_argument('--token', type=str, required=False)
        parser.add_argument('--folder_id', type=str,
required=False)
        parser.add_argument('--target_language', type=str,
required=False)

        return parser

    def __init__(self):
        self._parser = self._init_parser()

    def parse_config(self, data: str | list[str]) -> Config:
```

```
    raw_config = vars(self._parser.parse_args(data))

    config = copy.deepcopy(default_config)

    if token := raw_config.get('token'):
        config['token'] = token

    if folder_id := raw_config.get('folder_id'):
        config['folder_id'] = folder_id

    if target_language := raw_config.get('target_language'):
        config['target_language'] = target_language

    config['prompt'] = raw_config.get('prompt')

    return config
```

Результат выполнения программы

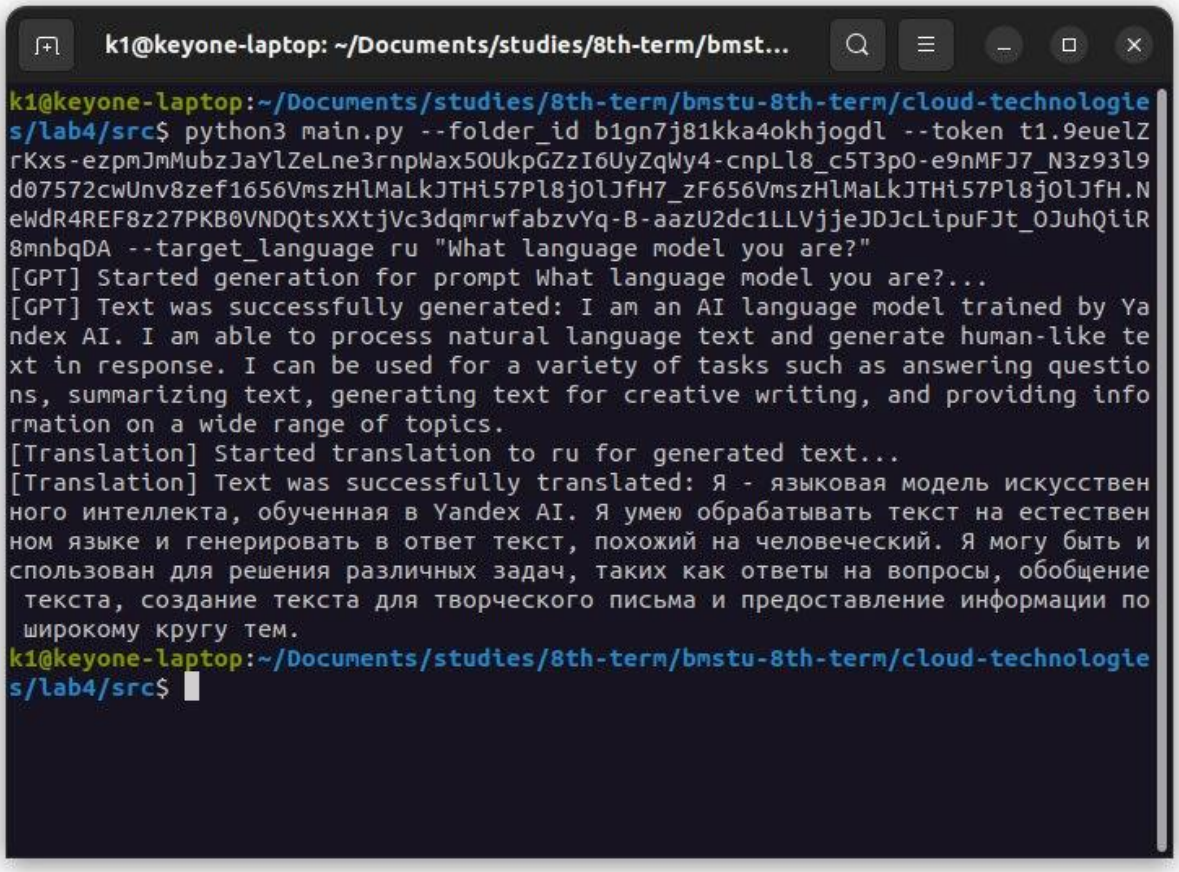

Рисунок 3 – Результат выполнения программы

Вывод: в ходе выполнения лабораторной работы были получены практические навыки по работе с сервисом Yandex Cloud.