МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСТИТЕТ»
ФИНАНСОВО-ЭКОНОМИЧЕСКИЙ ИНСТИТУТ

ОТЧЕТ ПО ИНДИВИДУАЛЬНОЙ РАБОТЕ

НА ТЕМУ «Парсинг информации о студентах»

Работу выполнил:
Титов Павел Сергеевич
Группа: ЛБ-01
Преподаватель:
Актаев Нуркен Ерболатович

Тюмень, 2022

Содержание

Постановка задачи

Реализовать парсер студентов ТюмГУ и выгрузку полученных данных в csv для анализа среднего балла и формы обучения.
Результат: .csv файл со списком студентов.
У каждого студента выделены следующие поля:

1. Уровень образования (Бакалавр/Магистр)
2. Код специальности
3. ФИО
4. Рейтинг (Средний балл)
5. Год поступления

1 Листинг кода работы с парсером и выгрузки

Импорты

Для обработки данных, их необходимо:

1. Получить (UtmnParser)
2. Обработать (json - Для выгрузки, кеширования и прочего, pydantic - для создания модели студента, tqdm - красивый progress bar)
3. Сохранить/Выгрузить (csv)

```
import csv
import json

from pydantic import BaseModel
from tqdm import tqdm

from app.utmn_parser import UtmnParser
from settings import getLogger, settings
```

Логгирование

Инициализируем в начале файла логгер, чтобы в логах было понятно где выброшено исключение:

```
logger = getLogger(__name__)
```

Исключения

Собственные классы ошибок позволят точнее определять, что пошло не так.

```
class InvalidUsername(Exception):
    pass


class StudentNotFound(Exception):
    pass
```

Модели

Модель студента для удобного представления данных:

```
class Student(BaseModel):
    education_level: str
    specialty_code: str
    entered: int
    fio: str
    rating: float
    entered_upon: str
```

Валидация

Валидация позволит исключить студентов, не имеющих важные для сбора информации поля.

```
def validate_student(student: dict, cached_students: dict):
    username = student['username']
    if username is None:
        raise InvalidUsername

    student_info = cached_students[username]
    if student_info is None:
        raise StudentNotFound
```

Получение нужной информации о студентах

Получение информации делится на несколько этапов:

1. Валидация
2. Получение первичной информации
3. Получение более подробной информации (данные из зачётной книжки)
4. Компоновка данных в модель

```python
def get_students_information(students: list, cached_students: dict):
    for student in students:
        try:
            validate_student(student, cached_students)
        except InvalidUsername:
            logger.error(f'{student} have invalid username')
            continue
        except StudentNotFound:
            logger.error(f'{student} not found')
            continue

        username: str = student['username']
        fio: str = student['displayName']
        student_info = cached_students[username]

        rating: float = 0.00
        entered_upon: str = 'No info'
        entered: int = student_info['main'].get('entered')
        specialty_code: str = student_info['main'].get('specialtyCode')
        education_level: str = student_info['main'].get('educationLevel')

        for studbook in student_info['studbooks']:
            if not studbook.get('active'):
                continue
            rating = student_info.get('rating').get('progress')
            if isinstance(rating, dict):
                rating = rating.get(student_info['defaultStudbook'], 0.00)
            entered_upon = studbook.get('enteredUpon')

        yield Student(
            education_level=education_level,
            specialty_code=specialty_code,
            entered=entered,
            fio=fio,
            rating=round(float(rating), 2),
            entered_upon=entered_upon,
        )
```

Кэширование данных

Чтобы каждый раз не запрашивать подробную информацию о студенте, создадим функцию сохранения.
Принцип работы следующий:

1. Получение уже сохранённой информации
2. «Пробег» по всем студентам (с общей информацией)
3. Сохранение, если подробных данных нет
4. Возврат сохранённых студентов

```python
def cache_students(up: UtmnParser, students: list) -> dict:
    with open('cached_students.json', 'r', encoding='utf8') as f:
        cached_students = json.load(f)

    students_bar = tqdm(desc='Collecting student data', total=len(students))
    for student in students:
        username = student.get('username')
        if not cached_students.get(username):
            student_info = up.get_student(username).get('response')
            cached_students.update({username: student_info})
            with open('cached_students.json', 'w', encoding='utf8') as f:
                json.dump(cached_students, f, ensure_ascii=False)
        students_bar.update(1)
    students_bar.close()

    return cached_students
```

Главная функция

Собираем всё воедино:

1. Создание экземпляра парсера
2. Получение общей информации о студентах направления
3. Получение подробной информации о студентах (из кэша, например)
4. Сортировка + выгрузка данных в файл

```python
def main(study_plan: str, qualification: str, entered: int):
    up = UtmnParser(settings.app.usernameOrEmail, settings.app.password)

    students = up.get_all_students_by_study_plan(
        study_plan=study_plan,
        qualification=qualification,
        entered=entered,
    )

    cached_students = cache_students(up, students)

    students = sorted(
        get_students_information(students, cached_students),
        key=lambda student: (
            student.rating,
            student.education_level,
            student.specialty_code,
            student.entered,
        ),
        reverse=True,
    )
    fieldnames = list(Student.__fields__.keys())
    with open('students.csv', 'w') as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        for student in students:
            writer.writerow(student.dict())
```

Запуск

```python
if __name__ == '__main__':
    study_plan = 'students plan ...'
    qualification = 'qualification level ...'
    entered = 2021
    main(study_plan, qualification, entered)
```

2 Листинг кода парсера

Импорты

1. urlencode - Форматировать параметров для отправки
2. requests - Для запросов к api

```
import json
from urllib.parse import urlencode

import requests
from settings import getLogger
from tqdm import tqdm
```

Логгер

```
logger = getLogger(__name__)
```

Исключения

```
class InvalidTokenException(Exception):
    pass
```

Создания класса парсера

В headers прописывается Content-Type для получения ответа в json формате, и User-Agent для доступа к api.
В init методе производится авторизация.

```
class UtmnParser:
    _api_url: str = 'https://nova.utmn.ru/api/v1'
    _headers: dict = {
        'Content-Type': 'application/json',
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5)
                                                AppleWebKit/537.36 (KHTML,
                                                like Gecko) Chrome/50.0.2661.
                                                102 Safari/537.36',
    }

    def __init__(self, username: str, password: str) -> None:
        self._headers.update({'Authorization': self._get_token(username,
                                                password)})
```

Метод получения токена для авторизации

Метод авторизации работает следующим образом:

1. Формирование данных для отправки (логин+пароль)
2. Отправка запроса на авторизацию
3. Получение токена

```
    def _get_token(self, username: str, password: str) -> str:
        '''Getting token by username and password

        Args:
            username (str): Username or e-mail
            password (str): Password

        Raises:
            InvalidTokenException: Authorization data is not valid

        Returns:
```

```
            str: Received token
        '''
        payload = json.dumps({'usernameOrEmail': username, 'password':
                                                password})
        resp = requests.post(
            f'{self._api_url}/auth/signin', headers=self._headers, data=
                                                    payload
        )
        resp = resp.json().get('response')
        if token := resp.get('token'):
            return token
        raise InvalidTokenException()
```

Метод получения всех студентов направления

Получение делится на два этапа:

1. Отправка запроса для получения информации о кол-ве студентов и страниц.
2. Отправка запроса для получения всех студентов

Далее просто сбор полученных данных.

```
    def get_all_students_by_study_plan(
        self,
        study_plan: str,
        qualification: str,
        entered: int,
    ) -> list:
        '''Getting all students of a given direction

        Args:
            study_plan (str): Full name of the study plan
            qualification (str): Qualification level
            entered (int, optional): Entered Year. The default is 2021.

        Returns:
            dict: query result
        '''
        params = {
            'limit': 1,
            'searchRole': 'student',
            'entered': entered,
            'studyPlan': study_plan,
            'offset': 0,
            'qualification': qualification,
        }
        limit = 20
        resp = requests.get(
            f'{self._api_url}/users?{urlencode(params)}', headers=self.
                                                    _headers
        ).json()
        total = resp.get('response').get('total')

        all_students = [*resp.get('response').get('users')]
        params['limit'] = limit
        students_bar = tqdm(desc='Collecting primary data on students',
                                                total=total)

        for offset in range(1, total + 1, limit):
            params['offset'] = offset
            resp = requests.get(
                f'{self._api_url}/users?{urlencode(params)}', headers=self.
                                                        _headers
```

```
            ).json()
            logger.debug(resp)
            all_students += resp.get('response').get('users')
            students_bar.update(limit)
        students_bar.close()

        return all_students
```

Метод получения подробной информации

Поскольку метод на получение информации о студентах направления возвращает общую информацию, необходимо запросить подробную информацию отдельным методом.

```
    def get_student(self, username: str) -> dict:
        '''Getting detailed information about the student

        Args:
            username (str): student username on vmeste

        Returns:
            dict: query result
        '''
        return requests.get(
            f'{self._api_url}/users/username/{username}', headers=self.
                                                    _headers
        ).json()
```

3 Листинг кода настроек

Настройки приложения:

Логгирование, Модель настроек (данные для авторизации)

```
from logging import INFO, FileHandler, StreamHandler, basicConfig, getLogger
from pathlib import Path

from pydantic import BaseModel, BaseSettings
from yaml import SafeLoader, load

CONFIG_FILE = str(Path(__file__).parent.absolute()) + '/settings.yaml'
LOGFILE_FILE = str(Path(__file__).parent.absolute()) + '/utmn.log'
basicConfig(
    level=INFO,
    format='[%(asctime)s] [%(levelname)s] [%(name)s] [%(funcName)s():%(
                                            lineno)s] %(message)s',
    handlers=[FileHandler(LOGFILE_FILE), StreamHandler()],
)
with open(CONFIG_FILE, 'r') as f:
    cfg = load(f, SafeLoader)


class App(BaseModel):
    usernameOrEmail: str
    password: str


class Settings(BaseSettings):
    app: App


settings = Settings.parse_obj(cfg)
```

4 Файлы настроек и зависимостей

Файл конфигурации

```
app:
  usernameOrEmail: ''
  password: ''
```

Файл с зависимостями для poetry

```
python = '^3.10'
flake8 = '^5.0.4'
pydantic = '^1.9.1'
pyyaml = '^6.0'
requests = '^2.28.1'
tqdm = '^4.64.0'
```

Результат

В результате получаем .csv файл с интересующей нас информацией:

education_level	specialty_code	entered	fio	rating	entered_upon
Бакалавр	02.03.03	2021	student fio	5.0	Бюджетная основа
Бакалавр	02.03.03	2021	student fio	4.71	Бюджетная основа
Бакалавр	02.03.03	2021	student fio	4.14	Бюджетная основа
Бакалавр	02.03.03	2021	student fio	3.29	Полное возмещение затрат
Бакалавр	02.03.03	2021	student fio	0.0	No info