BIGDATA TEAM

HW #03: Asset Web Service

Deadline: 11.01.2021, 08:00
Регистрация проблем до 07.01 ,08:00, в форме по ссылке

---

1. Описание задания

В данном ДЗ нужно написать финансово-аналитический Web Service, который позволит мониторить изменение курса валют и их влияние на инвестиционные продукты. Для решения задания вам пригодятся умения:

- парсить HTML (например с помощью bs4, lxml и/или XPath);
- использовать паттерны проектирования (например Composite);
- тестировать и поднимать Web-сервисы на Python с помощью Flask и pytest.

Бонусные задачи (не на оценку, но для наработки практики):

- писать и организовать шаблоны Jinja2;
- логировать, мониторить и запускать Web-сервис на PaaS платформе (например Heroku).

2. Критерии оценивания

Балл за задачу складывается из:

- **30%** - правильная реализация парсинга
- **60%** - правильная реализация API Web-сервиса
- **10%** - качество кода (pylint[1]), точная формула:
    - 10% x min([lint_quality / 8.0], 1.0)

Бонусы и штрафы:

- **100%** за плагиат в решениях (всем участникам процесса)
- **100%** за использование буквосочетаний “exec”, “eval”, “pylint” (Слова с такими сочетаниями (например, “execute”) тоже провоцируют этот штраф)
- **5%** за каждую повторную посылку

[1] pylint версии 2.5.3

3. Задача: Asset Web Service

Корреспондент спрашивает Рокфеллера как он разбогател.

- Когда я был маленький у меня было 2 цента. Я мог пойти в кино, но я купил грязное яблоко, помыл его и продал за 4 цента. На эти деньги я мог купить гамбургер, но я купил 2 грязных яблока, помыл их и продал за 8 центов. На эти деньги я мог пойти в ресторан, но я купил 4 грязных яблока, помыл их и продал за 16 центов, а потом умер мой папа и оставил мне 100 млн долларов.

В предыдущих модулях вы уже познакомились с консольным приложением asset.py, который предоставлял возможность рассчитать инвестиционную привлекательность актива за интересующий период в будущем:

- https://github.com/big-data-team/python-course/asset.py

В рамках этого задания предлагается расширить функциональность библиотеки по работе с активами в разной валюте, учитывающую текущие оценки курсов валют на сайте Центрального Банка Российской Федерации (cbr.ru).

В HTML-документах, которые будут использованы при тестировании (как парсинга, так и работы веб-сервиса) могут быть удалены некоторые непопулярные указанные на сайте и добавлены выдуманные валюты для проверки того, что парсер и код веб-сервиса не привязан к конкретному набору валют.

Ваша программа не должна в ходе работы печатать что-либо в stdout. Это может привести к неинформативным ошибкам в отчете по тестированию.

3.1 Парсинг

Ваш код должен содержать функции по парсингу выдачи следующих страниц (обратите внимание, что в версии на английском языке):

- https://www.cbr.ru/eng/currency_base/daily/
  (функция parse_cbr_currency_base_daily)
  (Snapshot: github../cbr_currency_base_daily.html)

- https://www.cbr.ru/eng/key-indicators/
  (функция parse_cbr_key_indicators)
  (Snapshot: github../cbr_key_indicators.html)

У функции парсинга один аргумент - строка, представляющая из себя содержимое HTML-документа. Результатом парсинга является словарь (точность сравнения дробных чисел 10e-8):

- {“буквенный код”: <курс валют в обмене на 1 у.е.>}

Не используйте requests.get во время парсинга html-документа, иначе парсинг будет посчитан некорректным.

3.2 Flask app

Flask приложение должно называться app и создаваться с помощью конструкции Flask(__name__) (тестирующая система использует конструкцию *from task_<Surname>_<Name>_asset_web_service import app*), все обращения в интернет должны производиться с помощью вызова “requests.get” (именно такие запросы будут Mock’аться в тестах). Гарантируется, что надежные ответы дают обращения только к следующим публичным сущностям класса requests.models.Response: “status_code”, “encoding”, “text”, “content” (последние три только в случае status_code == 200). Другие могут выдавать правдоподобные ответы, но полагаться на них не стоит.

Проверяться будут только указанные в условии случаи неожиданного поведения, неправильного ввода и т.п. Тем не менее, выявлять и обрабатывать все такие ситуации - полезное упражнение.

Пожалуйста, учитывайте, что как и реальный пользователь тестирующая система может за одну сессию переходить по нескольким адресам вашего веб-приложения.

Приложение должно реализовывать следующие route’ы (во всех случаях у запроса HTTP метод GET):

- Обработчик ошибки 404 (возвращает код 404 и текст “This route is not found”) в случае обращения по не существующему route (проверяйте в тестах код возврата)
- В случае недоступности cbr.ru, необходимо возвращать ошибку 503. Для этой ошибки должен быть зарегистрирован обработчик, который будет

возвращать сообщение “CBR service is unavailable”. Для тестирования этого поведения необходимо запатчить обращения “requests.get”

- Json API: /cbr/daily - сделать запрос на страницу “daily” и получить значения курсов валют в формате {“char_code”: rate}
- Json API: /cbr/key_indicators - сделать запрос на страницу “key-indicators” и получить значения для курса валют USD, EUR и драгоценных металлов[2].
- /api/asset/add/char_code/name/capital/interest (не забудьте про указание converter’ов Flask) - добавить актив в валюте “char_code” с именем “name”, размером капитала “capital” и оценочной инвестиционной годовой доходностью “interest” (в процентах, записанных дробным числом; то есть в качестве interest может быть указано число 0.5, что будет означать 50%). Для хранения всех активов вам пригодится шаблон проектирования Composite и глобальное хранилище активов (например, сохраните в переменную app.bank[3]). Запрос должен возвращать код возврата 200 и сообщение "Asset '{name}' was successfully added". В случае попытки добавления актива с именем (name), который уже существует в базе, то система должна выдавать код возврата 403.
- Json API: /api/asset/list - вернуть список всех доступных активов, каждый актив представить списком [“char_code”, “name”, “capital”, “interest”] Сортировка списков по умолчанию (то есть главенствующую роль в сортировке играет char_code).
- /api/asset/cleanup - очистить список активов. Запрос должен возвращать код возврата - 200.
- Json API: /api/asset/get?name=name_1&name=name_2 - вернуть список всех перечисленных активов, каждый актив представить списком [“char_code”, “name”, “capital”, “interest”]. Сортировка списков по умолчанию (то есть главенствующую роль в сортировке играет char_code).
- Json API: /api/asset/calculate_revenue?period=period_1&period=period_2 - посчитать оценочную инвестиционную доходность за указанные периоды времени (вернуть словарь {“period”: “revenue”}), где для валют

[2] Фильтрацию по длине char_code делать нельзя (в тестах будут и длинные имена), можно использовать только структуру HTML и указанные классы тегов для парсинга нужных значений

[3] Вопросы thread-safe и корректной работы в проде через multiprocessing - хорошие, но выходят за рамки текущего учебного модуля. Хочется узнать про это больше - пишите комментарии, будем готовить доп. материалы.

USD, EUR и драгоценных металлов делать обращения на страницу “key-indicators” (в противном случае реализация будет посчитана неверной), а остальные со страницы “daily”. (Точность сравнения дробных чисел 10e-8.) Также учитывайте, что может быть и рублевый актив.

Бонусная задача #1 (не оценивается):

- Json API: /api/asset/calculte_country_tax?period=p1&period=p2 - предположим, что все инвестиционные продукты в цепочке заработка используют человеческий труд (оплатили человеческий труд, произвели продукт и продали продукт с учетом наценки). Произведем оценочные вычисления по косвенному вкладу в бюджет страны различными отчислениями: Пенсионный Фонд России (ПФР) - 22% (к зарплате сотрудника), Фонд Социального Страхования (ФСС) - 2.9% (к зарплате сотрудника), Федеральный фонд обязательного медицинского страхования (ФФОМС) - 5.1% (к зарплате сотрудника), на травматизм - от 0,2 до 8,5 % (к зарплате сотрудника, для простоты будем считать 0.2%). ИТОГО - 30.2% страховых взносов, а также Налог на доходы физических лиц (НДФЛ) в размере 13%, то есть, учитывая оба фактора, в бюджет государства будет уплачено от капитала (1 - 1 / 1.302 * (1.0 - 0.13)) ≈ 0.332 (или 33.2%)[4]. Для простоты вычислений, налоги компании и НДС учитывать не будем. Вызов данного API должен возвращать словарь {“period”: “country_tax”}, для удобства выведите сразу два числа в словаре {“period”: [“revenue”, “country_tax”]}.

Бонусная задача #2 (не оценивается):

- Добавьте логирование в формате удобном для парсинга (например, чтобы можно было легко построить метрику флуктуации курса валют или числа последовательных неответов от cbr.ru) и разместите приложение через gunicorn на PaaS платформе Heroku (бесплатных dino-hours должно хватить с головой, чтобы разместить один экспериментальный Web-сервис). Можете хвастаться друзьям и коллегам запущенным Web-сервисом, но просьба не выкладывать исходный код в открытый доступ, чтобы другие слушатели самостоятельно решали задачу.

[4] Если считать от чистой зарплаты (на руки), то сверху к этой зарплате будет добавлено (1 / 0.87 * 1.302 - 1) ≈ 49.7% для уплаты в государственные фонды

- Инструкция (на примере Django, но вы должны легко справиться с переводом на Flask): https://devcenter.heroku.com/articles/getting-started-with-python

4. Общие рекомендации

При решении задач старайтесь следовать следующим рекомендациям:

- держите уровень покрытия кода тестами на уровне 80+%, следуйте TDD (сначала тесты, потом реализация);
- отделяйте фазу рефакторинга от фазы добавления новой функциональности, т.е.
  - фиксируем функциональность, все тесты зеленые;
  - проводим рефакторинг;
  - по окончании фазы рефакторинга снова все тесты зеленые;
- следите за скоростью выполнения unit-test'ов, несколько секунд - это хорошо, в противном случае нужно уменьшать размер тестируемых датасетов или разделять тесты на фазы (см. видео про mark.slow);
- следите за качеством кода и проверяйте "глупые" ошибки с помощью pylint, следите за поддерживаемостью и читаемостью кода;

5. Правила оформления задания

Ваша программа не должна в ходе работы печатать что-либо в stdout. Это может привести к неинформативным ошибкам в отчете по тестированию.

Оформление задания:

- Выполненное ДЗ запакуйте в архив **MADEPY20Q4_<Surname>_<Name>_HW3.zip**, например, для Алексея Драля -- MADEPY20Q4_Dral_Alexey_HW3.zip. Если ваше решение лежит в папке my_solution_folder, то для создания архива hw.zip на Linux и Mac OS выполните команду:
  - `zip -r hw.zip my_solution_folder/*`
- На Windows 7/8/10: необходимо выделить необходимое для сдачи содержимое директории my_solution_folder/ нажать правую кнопку мыши на одном из выделенных объектов, выбрать в открывшемся меню

"Отправить >", затем "Сжатая ZIP-папка". Теперь можно переименовать архив.

- Перед проверкой убедитесь, что дерево вашего архива выглядит так:
  - `| MADEPY20Q4_<Surname>_<Name>_HW3.zip`
  - `| ---- task_<Surname>_<Name>_asset_web_service.py`
  - `| ---- test_<Surname>_<Name>_asset_web_service.py`[5]
  - При несовпадении дерева вашего архива с представленным деревом ваше решение не будет возможным автоматически проверить, а значит, и оценить его.
- Для того, чтобы сдать задание необходимо:
  - Зарегистрироваться и залогиниться в сервисе Everest
  - Перейти на страницу приложения "BigData Team | MADE Python Grader"
  - Выбрать вкладку Submit Job (если отображается иная).
  - Выбрать в качестве “Task” значение: “HW3: Asset Web Service”[6]
  - Загрузить в качестве “Task solution” архив с решением
  - В качестве Sender ID указать свой индивидуальный id слушателя (если после объявления о том, что всем id разосланы, вы свой не получили, свяжитесь с нами).

Любые вопросы / комментарии / предложения:

- по работе тестирующей системы просьба писать на почту grader@bigdatateam.org;
- по заданиям и в целом по курсу - в Discord-канал курса (#python).

[5] Тесты вашего приложения, которые можно запустить с помощью “PYTHONPATH=. pytest test_….py”.
[6] Сервисный ID: python.asset_web_service