README.md

Задание

Scoring API Tests

Задание: протестировать HTTP API сервиса скоринга. Шаблон уже есть в test.py, само API реализовывалось в прошлой части ДЗ.

1. Необходимо разработать модульные тесты, как минимум, на все типы полей и функциональные тесты на систему. Разрешается пользоваться любым тестовым фреймворком: unittest, nosetests, pytest.
2. Обязательно необходимо реализовать через декоратор функционал запуска кейса с разными тест-векторами (Либо, в качестве альтернативы, сделать тоже самое через фикстуры в pytest). Если берете готовый cases, то его надо допилить так, чтобы припадении теста было ясно какой кейс упал.

```
@cases([
    {"account": "horns&hoofs", "login": "h&f", "method": "online_score", "token": "", "arguments": {}},
    {"account": "horns&hoofs", "login": "h&f", "method": "online_score", "token": "sdd", "arguments": {}},
    {"account": "horns&hoofs", "login": "admin", "method": "online_score", "token": "", "arguments": {}},
])
def test_bad_auth(self, request):
    ...
```

3. store, который был захардкожен в None, наконец обретает смысл! нужно реализовать в store.py общение с любым клиент-серверным key-value хранилищем (tarantool, memcache, redis, etc.) согласно интерфейсу заданному в обновленном scoring.py. Обращение к хранилищу не должно падать из-за разорванного соединения (т.е. store должен пытаться переподключаться N раз преждем чем сдаться) и запросы не должны залипать (нужно использовать timeout'ы где возможно).
    - У store есть отдельно `get` и `cache_get`. В реальной системы, аналогом которой является API из ДЗ, есть отдельный кеш и отдельный key-value storage. Методы названы по-разному, чтобы показать что есть разница. В данном случае, внутри можно реализовать это и как хождение в одно и тоже хранилище. Важно то, как это будет протестировано с учетом разных требований для разных функций.
4. Естественно нужно протестировать этот новый функционал. Обратите внимание, функции `get_score` не важна доступность store'а, она использует его как кэш и, следовательно, должна работать даже если store сгорел в верхних слоях атмосферы. `get_interests` использует store как персистентное хранилище и если со store'ом что-то случилось может отдавать только ошибки.
5. Структура тестов https://realpython.com/python-testing/#writing-integration-tests

Цель задания: применить знания по тестированию, полученные на занятии. В результате получится прокачать навык дизайна тест-кейсов, разработки модульных и функциональных тестов, создания mock'ов.

Критерии успеха: задание **обязательно**, критерием успеха является работающий согласно заданию код, для которого проверено соответствие pep8, написана минимальная документация с примерами запуска, в README, например. Далее успешность определяется code review.

Deadline

Задание нужно сдать через неделю. То есть ДЗ, выданное в понедельник, нужно сдать до следующего занятия в понедельник. Код, отправленный на ревью в это время, рассматривается в первом приоритете. Нарушение делайна (пока) не карается, пытаться сдать ДЗ можно до конца курсы. Но код, отправленный с опозданием, когда по плану предполагается работа над более актуальным ДЗ, будет рассматриваться в более низком приоритете без гарантий по высокой скорости проверки

Обратная связь

Студент коммитит все необходимое в свой github/gitlab репозитарий. Далее необходимо зайти в ЛК, найти занятие, ДЗ по которому выполнялось, нажать "Чат с преподавателем" и отправить ссылку. После этого ревью и общение на тему ДЗ будет происходить в рамках этого чата.