Методы сбора и обработки данных при помощи Python

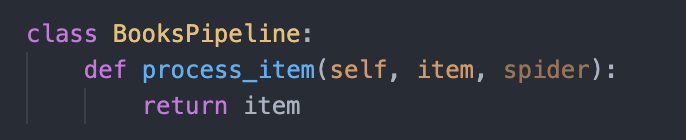
Scrapy



На этом уроке

1. Познакомимся с фреймворком Splash
2. Научимся сохранять данные из Scrapy в mongoDB и Sqlite

Если мы хотим импортировать все данные в базу данных, всю логику этого процесса надо описывать в файле pipelines.py.

Посмотрим на этот файл из нашего предыдущего проекта - books. Как видите, тут уже есть встроенный класс BooksItemPipeline

И как видите, тут уже есть встроенный метод process\_item(). В этом классе моггу быть еще два метода: open\_spider() и close\_spider(), оба принимают аргументы self и spider. Open вызывается один раз, когда паук начинает свою работу, то есть открывается, close вызывается тогда, когда паук закончил свою работу и закрылся.

Теперь нам надо перейти в настройки - файл settings.py. Там находим строчку ITEM\_PIPELINES и раскомментируем ее

В этом словаре мы можем указать все пайплайны, которые мы хотим использовать, прописав путь к каждому на подобие того, что у нас уже записано. То есть будет взят класс BooksPipeline из папки pipelines, которая находится внутри директории books. Цифра отвечает за приоритет.

Допустим, мы создадим еще один пайплайн, который будет отвечать за фильтрацию дубликатов. Назовем его books.pipelines.FilterDuplicates. И поставим ему значение 100ю Это значит, что этот пайплайн будет обрабатываться в первую очередь. Чем меньше цифра, тем выше приоритет.

Теперь давайте попробуем сохранять наши данные в MongoDB. На предыдущих уроках мы уже обсуждали, как скачать, установить и запустить локальный сервер MongoDB. Запускаем его еще раз. для этого идем в папку с mongodb/bin и вводим команду для старта сервера:

**brew services start mongodb-community@5.0**

Наш сервер запустился. теперь идем в файл pipelines.py. Меняем название класса на MongoDBPipeline, затем импортируем библиотеку pymongo: **from pymongo import MongoClient**

Затем создаем метод **open\_spider(self, spider)**: и внутри него создаем клиент **self.client = MongoClient(‘localhost’, 27017)**

Затем создаем объект базы данных: **self.db = self.client[‘Books’]**

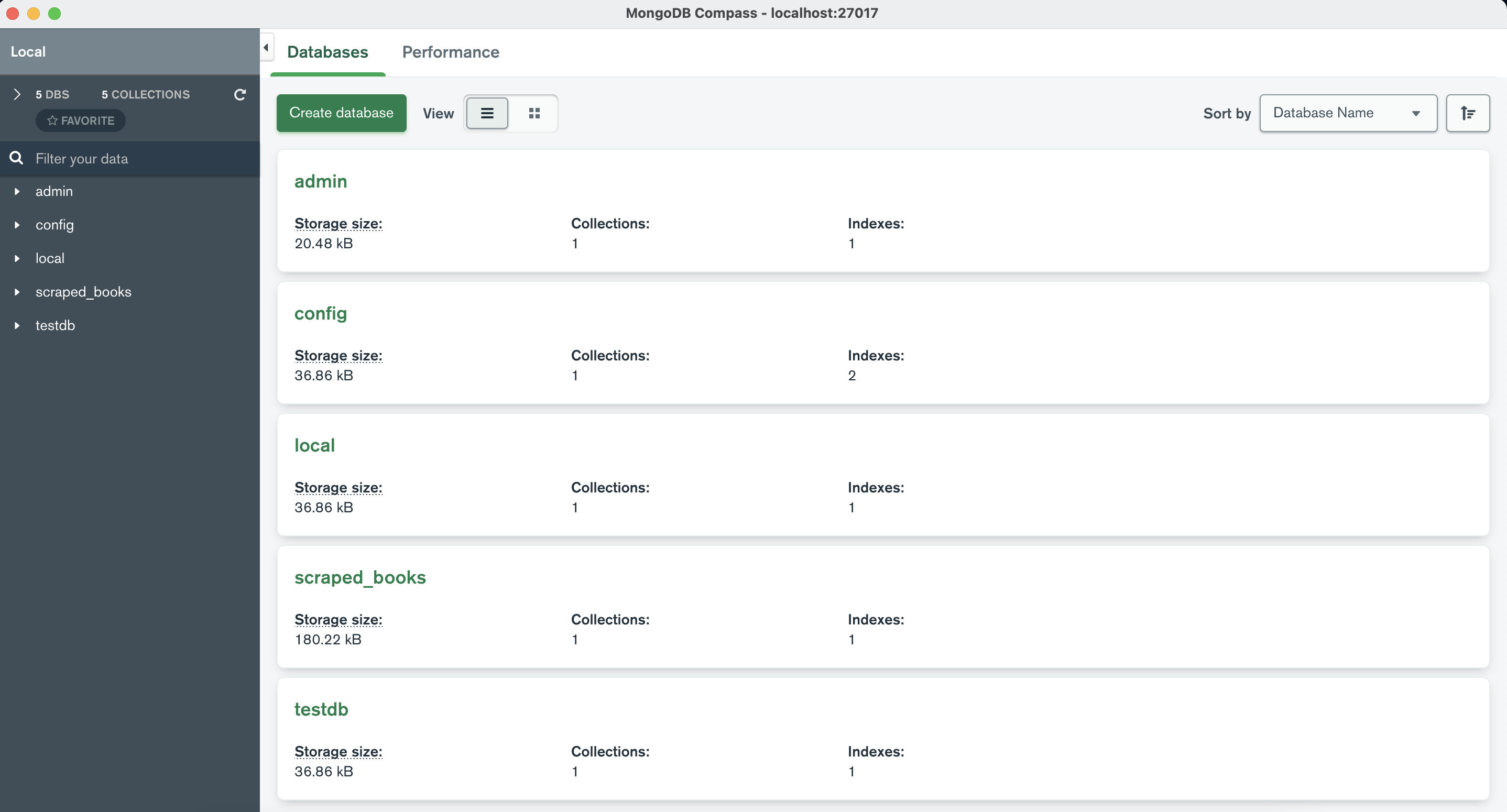
Теперь создаем метод **close\_spider(self, spider):** и внутри мы должны закрыть соединение с базой данных: **self.client.close()**

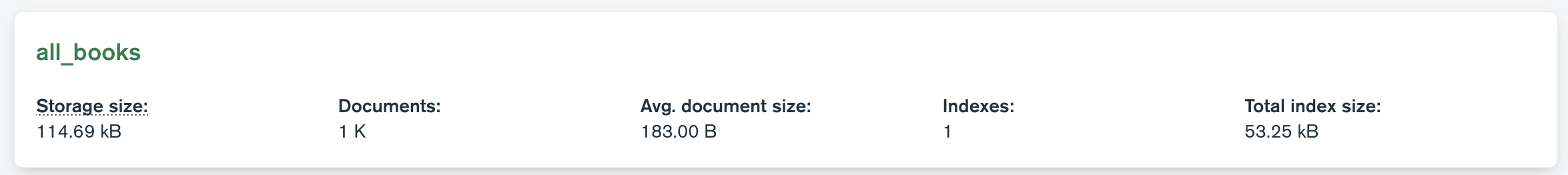
Внутри метода **process\_item()** мы будем вставлять в базу данных каждый элемент, который был спарсен нашим пауком.Для начала мы должны указать название коллекции, в которую будут сложены данные. Давайте создадим переменную в начале нашего класса **collection\_name = ‘all\_books’**. И внутри функции process\_item укажем эту коллекцию: **self.db[self.collection\_name]** и добавим метод **insert\_one()**. Как вы видите, у нас в функции insert\_item() уже есть аргумент item, который нам и надо сохранять в базу. Укажем его **self.db[self.collection\_name].insert\_one(item)**.

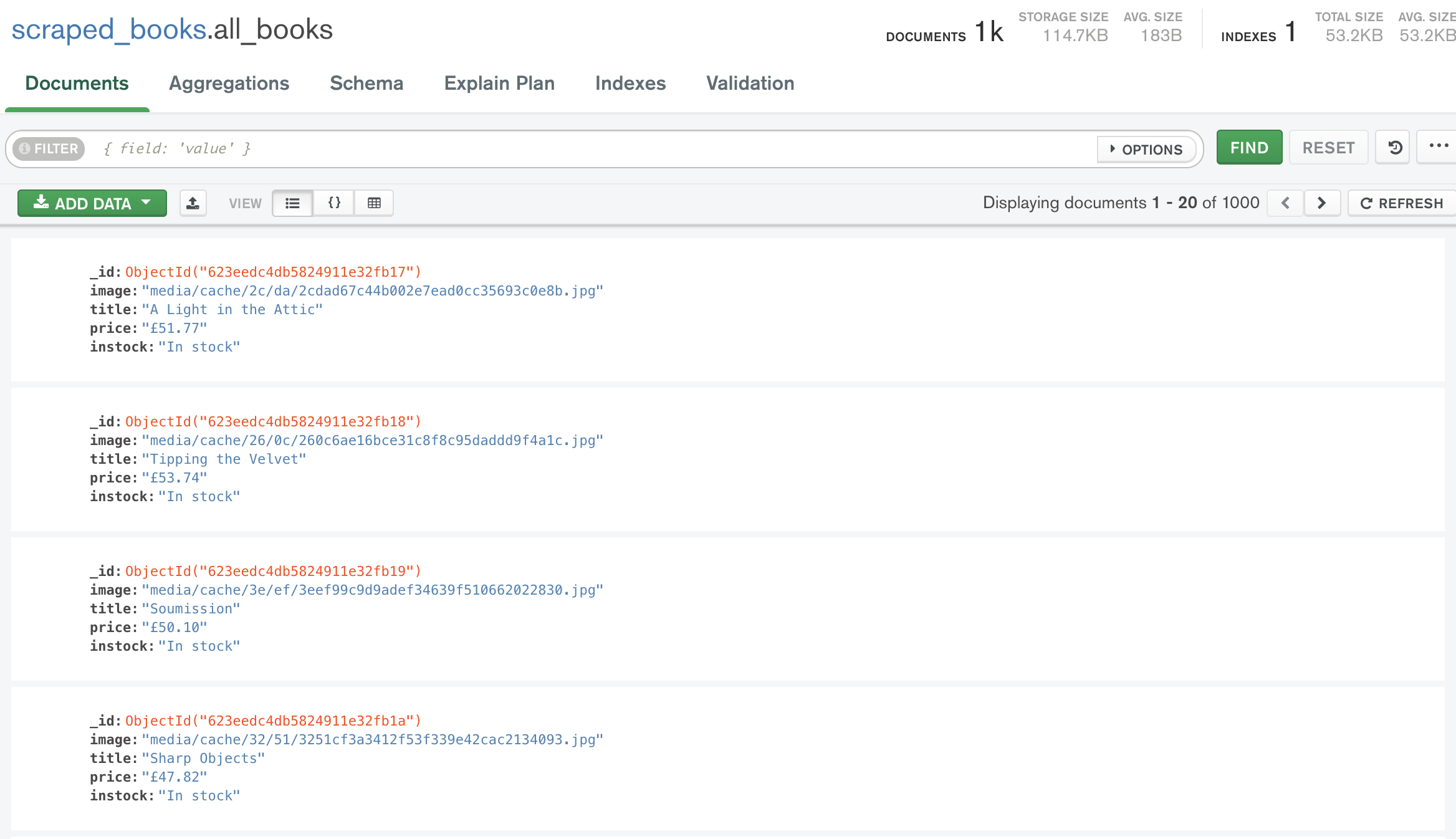
Переименовываем pipeline в файле с настройками: **MongoDBPipeline** и закомментируем переход на следующую страницу в пауке, чтобы убедиться, что всё у нас работает правильно.

Запускаем парсер. Как видите, все выполнилось, никаких ошибок нет. Теперь давайте посмотрим, что же у нас сохранилось и куда. Есть два варианта, мы можем пойти в терминал, набрать там **mongosh**, а затем **show dbs.** Как видите, у нас есть есть база данных Books. Перейдем в нее: **use Books** и посмотрим, что там у нас лежит **db.all\_books.find().** Как видите, у нас показан список элементов в базе данных. Посмотрим, сколько у нас всего элементов: **db.all\_books.countDocuments()**

Так как мы отключили переход на следующую страницу, то у нас всего 20 книг - всё верно. Теперь давайте удалим коллекцию и базу данных, включим переход на следующую страницу в парсере, данные будем собирать в другую базу, назовем ее **scraped\_books** и посмотрим, что получится. Но уже немного другим способом.

Парсер закончил работать. Проверяем: собралось 1000 книг. Так, теперь открываем MongoDB Compass, советую вам скачать эту программу, она бесплатная и удобная для исследования ваших баз данных. При открытии необходимо указать ссылку, по которой соединиться с вашей базой. Так как у нас запущен локальный сервер, указываем: mongodb://localhost:27017****

Как видите, у нас тут есть база данных scraped\_books. Кликаем на нее, тут всего одна коллекция - all\_books. И вы видите статистику: 1К документов, общий объем и так далее:

При клике на коллекцию мы попадаем в сами данные, тут можно просматривать и фильтровать данные, посмотреть схему, индексы, копировать и удалять данные.

Теперь давайте научимся сохранять данные в sqlite3. Для этого импортируем (**import sqlite3**) в файле pipelines и скопируем уже существующий пайплайн, чтобы было удобнее работать, назовем его **SQLitePipeline**.

Создаем переменную connection в методе **open\_spider(): self.connection = sqlite3.connect(‘books.db’)**. В методже connect передадим название базы данных. Пусть будет books. Теперь создадим курсор: **self.cursor = self.connection.cursor()**

Теперь напишем запрос для создания таблицы. Для простоты сделаем все поля текстовые.

**create\_table\_query = '''**

**CREATE TABLE all\_books(**

**image TEXT,**

**title TEXT,**

**price TEXT,**

**instock TEXT**

**)**

**'''**

Вообще при создании таблиц со схемами надо уделять большое внимание типам данных, которые вы собираетесь хранить, а так же первичным ключам. Но это отдельные темы для изучения. Теперь выполним запрос на создание таблицы: **self.cursor.execute(self.create\_table\_query)**

Сохраняем изменения в базу: **self.connection.commit()**

В методе close\_spider мы меняем **client** на **connection**, так как нам тоже надо закрыть соединение с базой, как и в случае с MongoDB.

Теперь создадим запрос на запись данных в таблицу:

**insert\_query = '''**

**INSERT INTO all\_books(image, title, price, instock)**

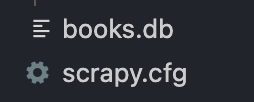
**VALUES(?,?,?,?)**

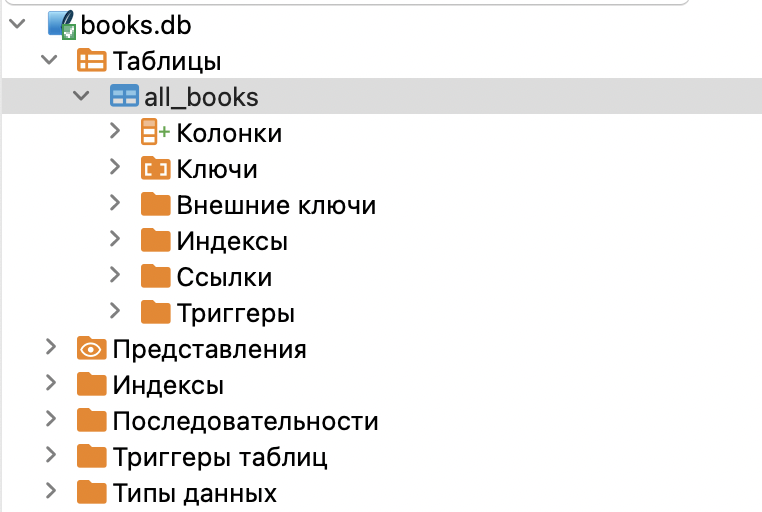
**‘''**

И выполним этот запрос в методе insert\_item(). Только вместо простой переменно item укажем кортеж следующим способом

**self.cursor.execute(self.insert\_query, (item.get(‘image’), item.get(‘title’), item.get(‘price’), item.get(‘instock’)))**

И сохраним данные: **self.connection.commit()**

Копируем и меняем имя пайплайна в настройках. И запускаем нашего паука. Смотрите, у нас появился файл books.db:

Просмотреть базу данных можно с помощью любой программы. У меня это DBeaver. Открываем и нажимаем на создать новое соединение, находим нашу базу данных.



Теперь давайте немного изменим наш код, так как если вы запустите его еще раз, то у вас будет ошибка, что таблица с таким именем уже существует. Поэтому запрос на создание таблицы нам надо обернуть в try except:

**try:**

**self.cursor.execute(self.create\_table\_query)**

**self.connection.commit()**

**except sqlite3.OperationalError:**

**pass**

Теперь, если мы запустим паука еще раз, ошибки не будет, а данные будут записаны в существующую таблицу. Давайте остановим выполнение кода. И посмотрим, сколько книг у нас теперь в таблице: **select count(\*) from all\_books ab**

На этом наше знакомство с пайплайнами в Scrapy завершено. Приступаем к знакомству с фреймворком Splash.

Splash

Часто при парсинге вы можете столкнуться с тем, что, прописав все пути до нужных элементов, вы получите пустые списки и массивы. Если вы проинспектируете html-ответ, который вам вернулся, вы можете увидеть, что разметка существенно отличается от той, которую вы только что изучали на сайте. Это всё связано с работой javascipt’a. Давайте разберемся на примере.

Идем на сайт [quotes.toscrape.com/js](http://quotes.toscrape.com/js). Вот у нас тут такая разметка. Теперь давайте отключим выполнение javascript’ов на сайте. Для этого достаточно нажать F1 и выбрать «Отключить JavaScript». сайт перезагрузится, если нет, перезагрузите его сами. И смотрите, у нас получился совсем другой сайт - в нем нет никакого наполнения. То есть при загрузке сайта выполняется какой-то скрипт, который загружает все данные на страницу. Это сделано в том числе и для защиты от парсинга.

К сожалению, ни requests, ни scrapy это обойти не могут, они загружают html ссылки без выполнения скриптов. Но есть чудесная библиотека - Splash Открываем ее домашнюю страницу - <https://github.com/scrapy-plugins/scrapy-splash>

Эта библиотека позволяет загружать ссылку с выполнением javascript’ов и возвращать html. Сейчас мы научимся это делать. У меня сплеш предустановлен, вам необходимо следовать инструкции. После запуска докера запустится и сплеш. Мы можем открыть его по этому адресу. Давайте для начала попробуем получить html страницы на страничке splash, а потом поговорим про его использование вместе со scrapy.

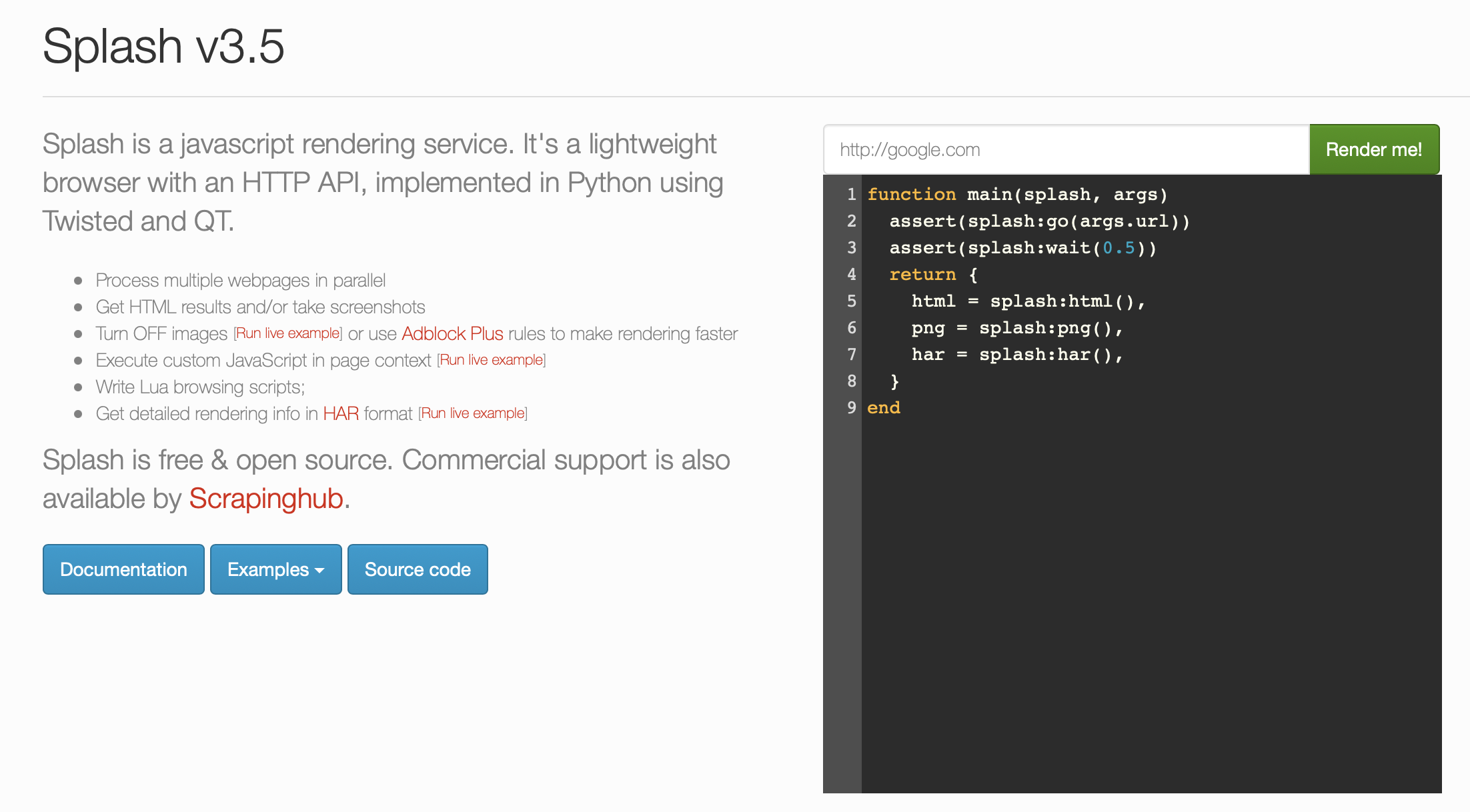
Итак, переходим по ссылке - localhost:8050 - откроется страничка splash. Здесь вы видите пример работы на языке lua, на котором написан Splash.

Давайте для начала попробуем получить html обычной страницы [quotes.toscrape.com](http://quotes.toscrape.com). Для этого в окошко вместо [google.com](http://google.com) вставляем нашу ссылку на сайт. Затем стираем здесь всё и пишем:

**url = args.url**

**splash:go(url)**

**return splash:html()**



И нажимаем кнопку Render! Как видите, нам вернулся ответ, мы можем скопировать его в редактор или просмотреть здесь. В принципе, и так видно, что нам вернулся html с цитатами. Теперь давайте изменим нашу ссылку, добавив к ней слеш js. Снова запускаем код. Смотрите, тут нам вернулся точно такой же html, как и с первой ссылкой. Значит, сплеш смог выполнить скрипты и получить полноценную страницу с кодом.

Давайте напишем еще несколько команд. Во-первых, splash:go(url) завернем в assert - это предотвратит splash от выполнения дальнейших строчек кода, если url не откроется. Так же мы можем добавить таймаут вручную, для этого напишем: **assert(splash:wait(1))** - в таком случае сплеш будет ждать 1 секунду и затем выполнит оставшуюся часть кода.

Теперь, при выполнении скрипта сплеш использует свой юзер агент, многие сайты могут заблокировать выполнение скриптов и в принципе работу сплеша, как только увидят, что юзер агент - сплеш. Давайте поменяем это в нашем коде. Мы можем сделать это двумя способами. Давайте найдем какой-нибудь юзер агент.

Первый способ: **splash:set\_user\_agent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.80 Safari/537.36»)**

Второй способ:

**headers = {**

**['User-Agent'] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.80 Safari/537.36"**

**}**

**splash:set\_custom\_headers(headers)**

Так же сайт может не возвращать html, потому что splash использует по умолчанию настройку инкогнито, сайт это распознает и блокирует сплеш от парсинга. Отключить эту настройку можно следующим образом:

**splash.private\_mode\_enabled = false**

Мы с вами будем парсить простой сайт, так что все эти модификации нам не нужны, но о них полезно помнить.

Теперь, когда мы познакомились со сплешем, давайте научимся запускать его из скрейпи. Создадим новый проект, назовем его splash\_quotes

**scrapy startproject splash\_quotes**

**cd splash\_quotes**

**scrapy genspider quotes [quotes.toscrape.com](http://quotes.toscrape.com)**

Старт юрлс нам не нужен, пока закомментируем, потому что ссылка отсюда нам еще понадобится. Теперь импортируем библиотеку scrapy\_splash: **from scrapy\_splash import SplashRequest.** И с главной страницы splash добавляем в настройки следующие строчки кода. Давайте скопируем их просто. Splash url можем поменять на localhost. Обратите внимание, что для правильно работы сплеш в скрейпи, у вас должен быть запущен сплеш на докере, иначе ничего не сработает.

Так, теперь возвращаемся к нашему пауку. Сначала создадим переменную script, в которую скопируем наш код из локального сплеша:

**script = '''**

**function main(splash, args)**

**url = args.url**

**assert(splash:go(url))**

**assert(splash:wait(1))**

**return splash:html()**

**end**

**‘''**

Теперь создадим функцию start\_requests, которая будет запускать выполнение скрипта на сплеше, а ответ передавать в следующую функцию. Итак:

**def start\_requests(self):**

**yield SplashRequest(**

**url='<http://quotes.toscrape.com/js>',** - указываем ссылку на сайт, который хотим спарсить и удаляем start\_urls

**callback=self.parse,** - после выполнения скрипта передаем ответ в функцию parse

**endpoint=‘execute',** - выполнить скрипт

**args={**

**'lua\_source': self.script** - указываем, какой скрипт выполнить

**}**

**)**

Теперь давайте соберем цитату и автора в функции parse:

**quotes = response.xpath("//div[@class='quote']")**

**for quote in quotes:**

**yield {**

**'author': quote.xpath(".//small[@class='author']/text()").get(),**

**'quote': quote.xpath(".//span[@class='text']/text()").get()**

**}**

Запускаем скрипт. Как видите, у нас собрались первые 10 цитат с сайта. Используя сплеш можно обходить такие вот блокировки. В последнем уроке мы с вами вернемся к сплешу, когда будем говорить о логине на сайт при его помощи. На этом сегодняшний урок завершен.

**Домашнее задание**

1. Собрать все цитаты с сайта [quotes.toscrape.com/js](http://quotes.toscrape.com), плюс провалиться в информацию о каждом авторе и собрать информацию о нем: день рождения и описание.
2. Полученные данные сохранить в БД на ваш выбор (mongoDB, SQLite3) с использованием пайплайна. Скриншот монги или .db файл приложить вместе с кодом.