**Тема занятия № 49: Модуль 30. Разработка Web-служб REST. REST framework. Django REST Framework**

1. Установка и подготовка к работе Django REST framework

Многие современные веб-сайты предоставляют программные интерфейсы, предназначенные для использования сторонними программами: настольными, мобильными приложениями и другими веб-сайтами. С помощью таких интерфейсов, называемых веб-службами, сторонние программы могут получать информацию или, наоборот, заносить ее на сайт.

Интерфейсы подобного рода строятся согласно принципам REST (Representational State Transfer, репрезентативная передача состояния). К числу этих принципов относится, в частности, идентификация запрашиваемых интернет-ресурсов посредством обычных интернет-адресов. Так, для получения перечня рубрик сторонней программе нужно обратиться по интернет-адресу, скажем, http://www.bboard.ru/Api/rubrics/.

Совокупность веб-служб (бэкенд) отсылает сторонним программам данные, закодированные в каком-либо компактном формате, обычно JSON. Сторонние клиентские программы (фронтенды) получают эти данные и обрабатывают нужным им образом, например, выводят на экран.

Веб-службу можно реализовать исключительно средствами Django. Ранее класс jsonresponse, отправляющий клиенту данные в формате JSON.

Однако для этих целей удобнее применять библиотеку Django REST framework.

Она самостоятельно извлечет данные из базы, закодирует в JSON, отправит фронтенду, получит данные от фронтенда, проведет их валидацию, занесет в базу и даже реализует разграничение доступа.

Внимание!

Django REST framework— это полноценный фреймворк, базирующийся на Django. Полная документация по Django REST framework находится по интернет-адресу:

Http://www.django-rest-framework.org/.

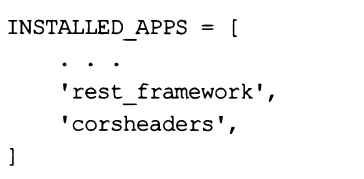
Установка этой библиотеки выполняется подачей команды:



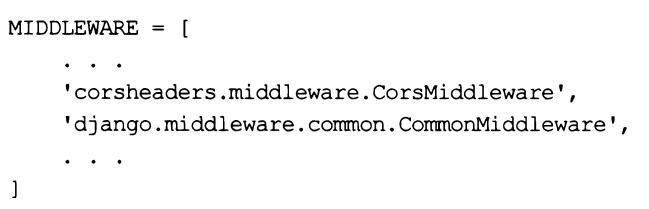
Изначально Django обрабатывает лишь запросы, пришедшие с того же домена, на котором располагается веб-служба. Чтобы разрешить фреймворку обрабатывать запросы с других доменов, понадобится дополнительная библиотека django-corsheaders. Установить ее можно подачей команды:



Программными ядрами библиотек Django REST framework и django-cors-headers являются приложения rest framework и corsheaders соответственно. Их необходимо добавить в список зарегистрированных в проекте (параметр installed\_apps модуля settings.py из пакета конфигурации):

****

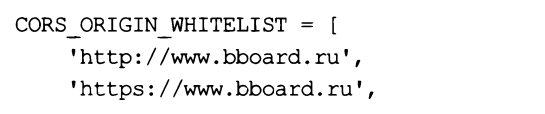
Кроме того, в список зарегистрированных в проекте (параметр middleware) нужно добавить посредник corsheaders.middleware.corsmiddleware, расположив его перед посредником django.middleware. Common. Commonmiddleware:

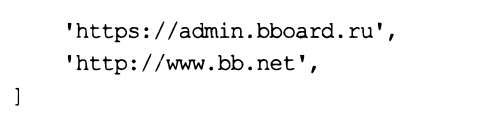


Библиотека Django REST framework для успешной работы не требует обязательного указания каких-либо настроек. Необходимые настройки библиотеки django-corsheaders весьма немногочисленны:

□ cors origin allow all — если True, то Django будет обрабатывать запросы, приходящие с любого домена. Если False, то будут обрабатываться только запросы с текущего домена и с доменов, заданных в параметрах cors origin whitelist и CORS\_ORIGIN\_REGEX\_WHITELIST. По умолчанию — False;

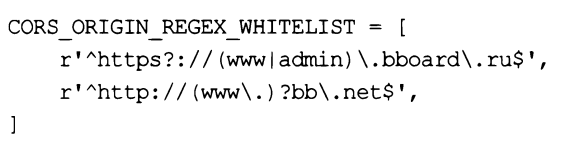
□ cors origin whitelist — список или кортеж доменов, запросы с которых разрешено обрабатывать. Домены задаются в виде строк. Пример:

****

****

Значение по умолчанию — ’’пустой” список;

□ cors\_origin\_regex\_whitelist — список или кортеж с регулярными выражениями, с которыми должны совпадать ’’разрешенные” домены:

****

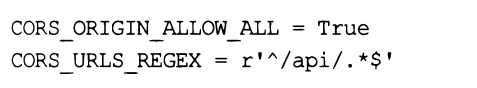
Значение по умолчанию — ’’пустой” список;

□ cors urls regex— регулярное выражение, с которым должен совпадать путь,

Запрос, по которому будет допущен к обработке, в виде строки. По умолчанию:

”А. \*$" (регулярное выражение, совпадающее с любым путем).

Например: чтобы разрешить обработку запросов, приходящих с любых доменов, но только к тем путям, что включают префикс api, следует добавить в модуль Settings.py пакета конфигурации такие выражения:

****

На заметку!

Полное руководство по библиотеке django-cors-headers находится здесь:

Https://github.com/ottoyiu/django-cors-headers/.

1. Введение в Django REST framework. Вывод данных

**Сериализаторы**

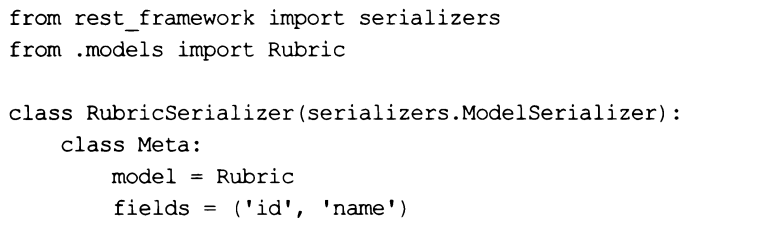
Сериализатор в Django REST framework выступает аналогом формы. Сериализаторы, связанные с моделями, самостоятельно извлекают данные из модели и "умеют” сохранять в ней данные, полученные от фронтенда.

Код сериализаторов обычно записывается в модуле serializers.py пакета приложения.

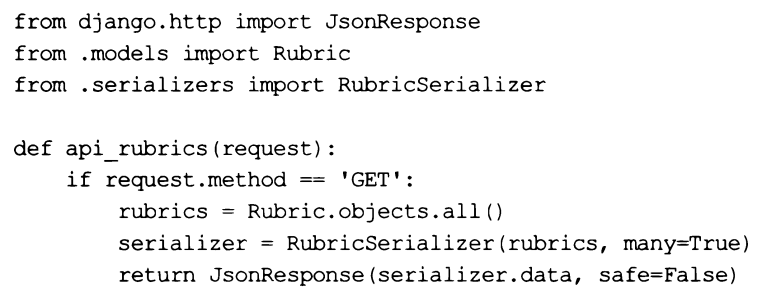
Этот модуль изначально отсутствует, и его придется создать самостоятельно.

Класс сериализатора, связанного с моделью, должен быть производным от класса Modelserializer ИЗ модуля rest\_framework. Serializers. В остальном ОН мало отличается от формы, связанной с моделью.

Приведен КОД сериализатора rubricserializer, связанного с моделью Rubric и обрабатывающего рубрики.

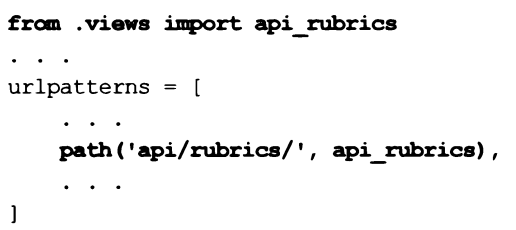
****

Проверим наш первый сериализатор в действии. Добавим в модуль views.py контроллер-функцию api\_rubrics (), который будет выдавать клиентам список рубрик, закодированный в формат JSON. Код контроллера приведен.

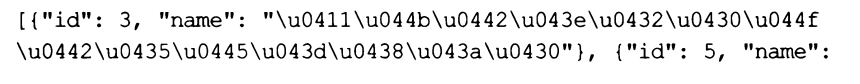
****

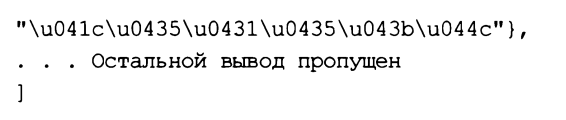
Конструктору класса сериализатора мы передали набор записей, который следует сериализовать, и параметр many со значением True, говоря тем самым, что сериализовать нужно именно набор записей, а не единичную запись. А ответ мы сформировали с помощью класса jsonresponse.

Наконец, добавим в список маршрутов уровня приложения (модуль uris.py пакета приложения) маршрут, указывающий на только что написанный нами контроллер:

****

Запустим отладочный веб-сервер Django, откроем веб-обозреватель и перейдем по интернет-адресу http://localhost:8000/api/rubrics/. Мы увидим на экране список рубрик в формате JSON.





**Веб-представление JSON**

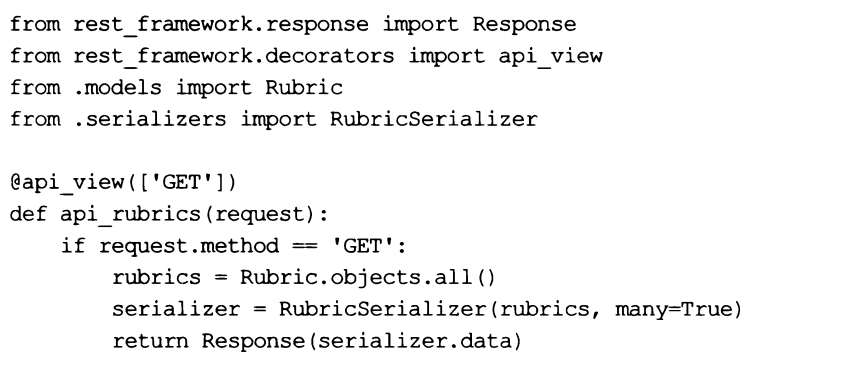
Если активно веб-представление JSON, то Django REST framework будет выводить JSON-данные на обычной веб-странице отформатированными для удобства чтения и с некоторыми дополнительными сведениями. Используем его, чтобы проверить, действительно ЛИ сериализатор rubricserializer выводит нам список рубрик.

Чтобы задействовать веб-представление, достаточно:

□ указать у контроллера-функции декоратор api\_view (Допустимые НТТР-методы>) ИЗ модуля rest\_framework.decorators. Допустимые HTTP-методы задаются в виде списка со строковыми наименованиями этих методов;

□ для формирования ответа вместо класса jsonresponse использовать класс Response ИЗ модуля rest fгamework.response. Конструктор ЭТОГО класса вызывается В формате: Response(<отправляемые данные>).

Приведен полный код обновленной версии контроллера-функции Api rubrics (), которая реализует веб-представление.

****

Сохраним исправленный код и попробуем наведаться по тому же интернет-адресу Http://localhost:8000/api/rubrics/. На этот раз веб-обозреватель покажет нам веб представление JSON.

Здесь выведены, прежде всего, сами JSON-данные в удобном для изучения виде и сведения о полученном ответе (код статуса, MIME-тип содержимого и пр.). Кнопка GET позволит вывести обычный JSON-код— для этого достаточно щелкнуть на расположенной в ее правой части стрелке, направленной вниз, и выбрать в появившемся на экране меню пункт json. Вернуть веб-представление данных можно выбором в том же меню пункта api или нажатием непосредственно кнопки GET, не затрагивая стрелки. Кнопка OPTIONS выводит сведения о самой веб-службе.

**Вывод данных на стороне клиента**

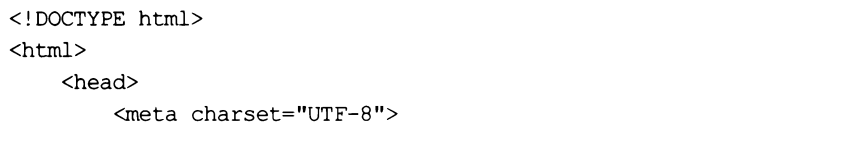
Полученные от бэкенда JSON-данные можно обработать и вывести на веб-странице средствами DOM и AJAX.

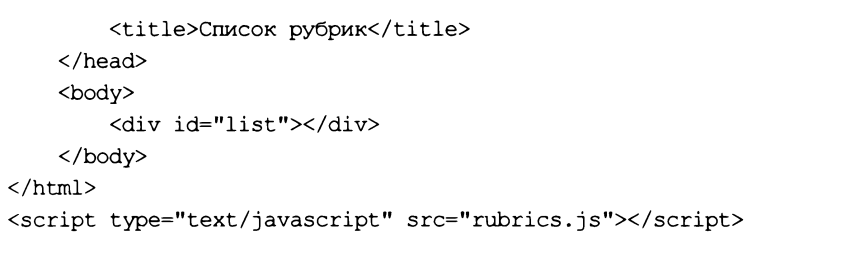
Внимание!

Веб-обозреватель может загружать данные по технологии AJAX только с веб-сервера, но не с локального диска. Кроме того, многие веб-обозреватели блокируют AJAX-загрузку данных на страницах, открытых с локального диска.

Для обслуживания тестового фронтенда использовался сторонний веб-сервер Apache HTTP Server с настройками по умолчанию.

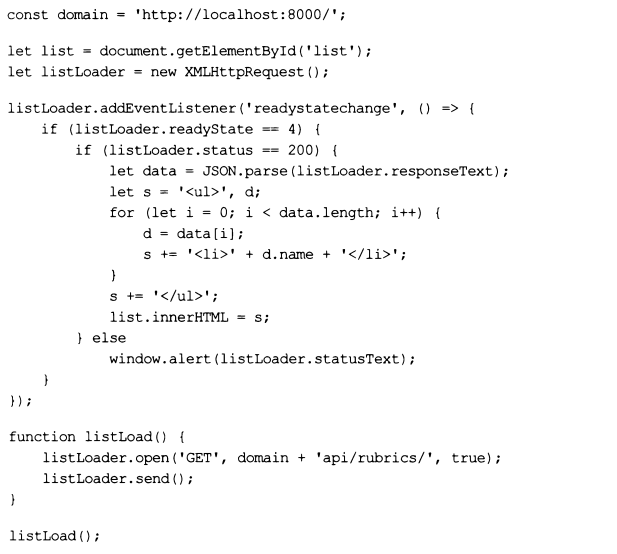
Приведен код веб-страницы rubrics.html, на которой будет выводиться перечень рубрик, полученный от веб-службы.





Блок (блочный контейнер, тег <div>) с якорем list будет использован для вывода маркированного списка с перечнем рубрик.

Далее код файла веб-сценария rubrics.js, загружающего и выводящего перечень рубрик в виде маркированного списка.



Код, запускающий загрузку перечня рубрик, оформлен в виде функции listloado.

Это позволит впоследствии, после добавления, правки или удаления рубрики, выполнить обновление перечня простым вызовом этой функции.

Сохраним файлы rubrics.html и rubrics.js в корневой папке стороннего веб-сервера.

Запустим отладочный веб-сервер Django и сторонний веб-сервер. В веб-обозревателе выполним обращение по интернет-адресу http://localhost/rubrics.html.

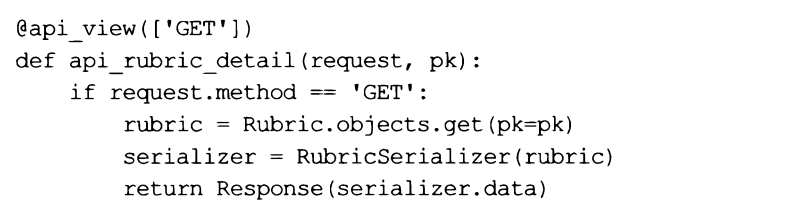
На открывшейся странице будет выведен перечень рубрик.

**Первый принцип REST:**

Идентификация ресурса по интернет-адресу согласно первому принципу REST, любой ресурс, выдаваемый бэкендом, идентифицируется интернет-адресом — как и обычная веб-страница. Например, у нас ресурс ’’перечень рубрик” идентифицируется интернет-адресом /api/rubrics/ — именно по нему фронтенд обращался к бэкенду для получения этого перечня.

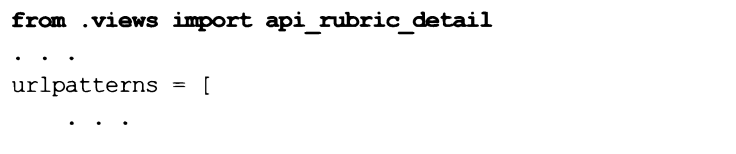
Логично ресурс ’’сведения о рубрике” с заданным ключом идентифицировать интернет-адресом формата /api/rubrics/<KTW4 рубрики>1.

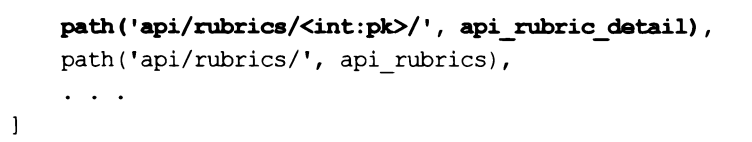
Приведен код контроллера, выдающий фронтенду сведения о рубрике с указанным ключом. Этот код мы добавим в модуль views.py.

****

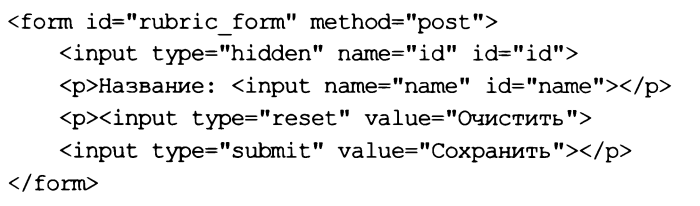
В вызове конструктора класса сериализатора rubricserializer не следует указывать параметр many со значением True, т. К. Сериализовать нужно лишь одну запись.

Добавим в список маршрутов уровня приложения маршрут, который укажет на новый контроллер:



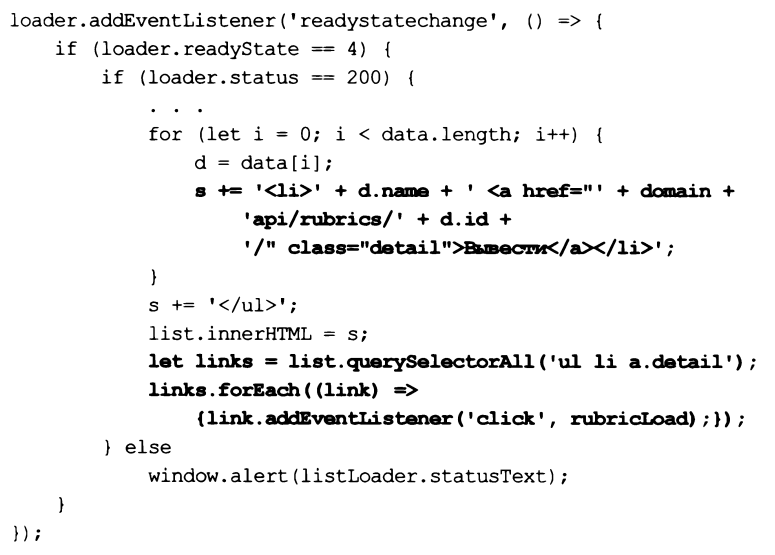


На страницу rubrics.html, непосредственно под блоком list, поместим веб-форму, в которой будут выводиться сведения о выбранной рубрике. Эту форму мы потом используем для добавления и правки рубрик. Вот HTML-код, создающий ее:

****

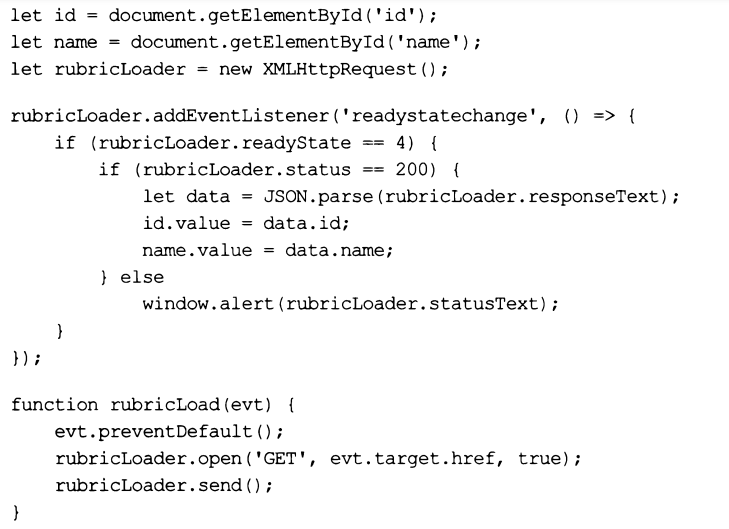
В форме присутствует скрытое поле id, хранящее ключ исправляемой рубрики. Он понадобится нам впоследствии.

Теперь исправим веб-сценарий, хранящийся в файле rubrics.js. Сначала сделаем так, чтобы рядом с названием каждой рубрики присутствовала гиперссылка Вывести, выводящая сведения о рубрике в только что созданной веб-форме. Вот правки, которые нам нужно внести в код:

****

Здесь нужно пояснить три момента. Во-первых, мы привязали к каждой из созданных гиперссылок стилевой класс detail — это упростит задачу привязки к гиперссылкам обработчика события click. Во-вторых, записали интернет-адреса для загрузки рубрик непосредственно в тегах <а>, создающих гиперссылки, — это также упростит нам дальнейшее программирование. В-третьих, привязали к созданным гиперссылкам обработчик события click---функцию rubricload ().

Теперь допишем в файл rubrics.js код, выводящий сведения о выбранной рубрике:

****

Функция rubricload о — обработчик события click гиперссылок Вывести — извлекает из атрибута href тега <а> гиперссылки, на которой был выполнен щелчок мышью, интернет-адрес и запускает процесс загрузки с этого адреса сведений о рубрике. Полученные сведения — название рубрики — выводятся в веб-форме.

Запустим отладочный веб-сервер Django и сторонний веб-сервер и перейдем по интернет-адресу http://localhost/rubrics.html. Когда на странице появится перечень рубрик, щелкнем на гиперссылке Вывести любой из них и проверим, выводятся ли в веб-форме сведения об этой рубрике.