**Тема занятия № 50: Модуль 30. Разработка Web-служб REST. REST framework. Django REST Framework**

1. Ввод и правка данных

**Второй принцип REST:**

Идентификация действия по НТТР-методу согласно второму принципу REST, действие, выполняемое над ресурсом (идентифицируемым уникальным интернет-адресом), обозначается HTTP-методом, указанным в запросе. В веб-службах REST применяются следующие методы:

□ GET — выдача ресурса. Ресурс может представлять собой как перечень каких-либо сущностей, так и отдельную сущность (в нашем случае в качестве сущностей выступают рубрики);

□ POST — создание нового ресурса;

□ PUT — исправление значений всех полей у ресурса. Отметим, что при использовании этого метода фронтенд должен отправить бэкенду значения всех полей;

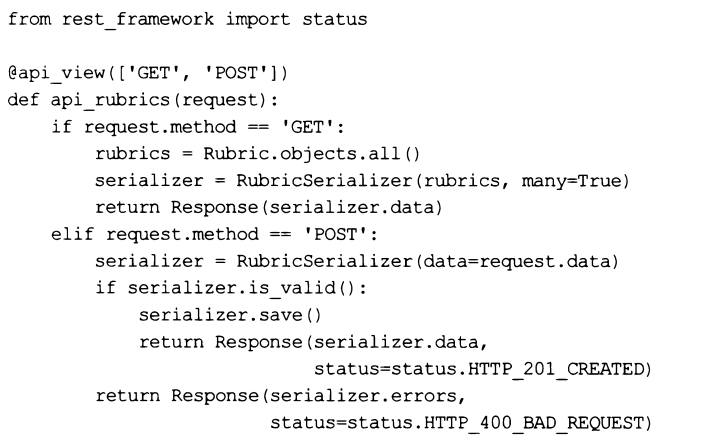
□ PATCH— исправление отдельных полей у ресурса. В этом случае фронтенд может отправить бэкенду значения только тех полей, которые нужно исправить.

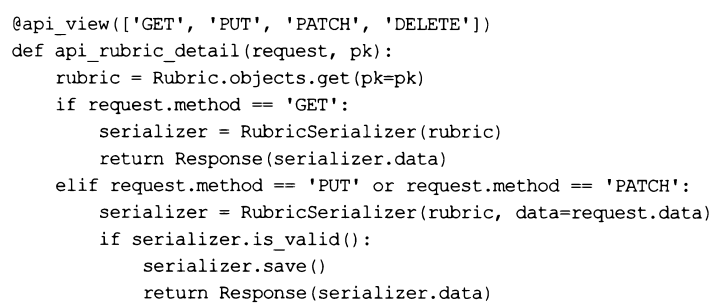
На практике методы PUT и PATCH часто обозначают одно и то же действие — исправление либо всех полей ресурса, либо отдельных его полей (это зависит от конкретной реализации);

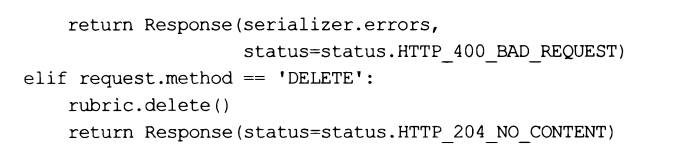
□ DELETE — удаление ресурса.

Обычно создание нового ресурса реализует тот же контроллер, который выдает ресурс-перечень сущностей, а исправление и удаление — тот же контроллер, который выдает ресурс-отдельную сущность. Благодаря этому для реализации всех этих действий достаточно записать всего два маршрута.

Приведен исправленный код контроллеров api\_rubrics () и Api rubric detaiiо, которые получили поддержку добавления, правки и удаления рубрик.







Сохранение и удаление записей с помощью сериализаторов, связанных с моделями, выполняется точно так же, как и в случае применения связанных с моделями форм.

Мы вызываем методы is vaiido, save о и delete о, а все остальное выполняют фреймворк Django и библиотека Django REST framework.

Теперь отметим три важных момента. Во-первых, мы добавили в вызовы декоратора api\_view() обозначения HTTP-методов POST, PUT, PATCH и DELETE, указав тем самым, что контроллеры поддерживают запросы, выполненные с применением этих методов. Если этого не сделать, мы получим ошибку.

Во-вторых, чтобы занести в сериализатор данные, полученные от фронтенда, мы присваиваем их именованному параметру data конструктора класса сериализатора.

Сами эти данные можно извлечь из атрибута data объекта запроса. Вот пример:



В-третьих, после успешного создания рубрики мы отсылаем фронтенду ответ с кодом 201 (ресурс успешно создан). Это выполняется передачей конструктору класса

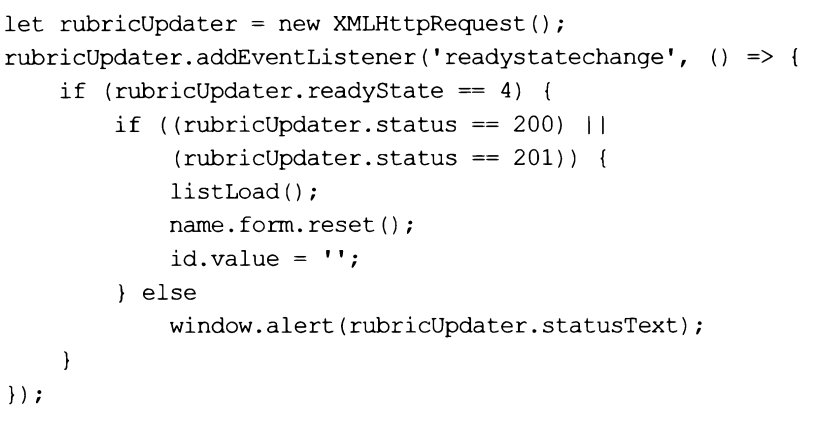
Response числового кода статуса посредством именованного параметра status:

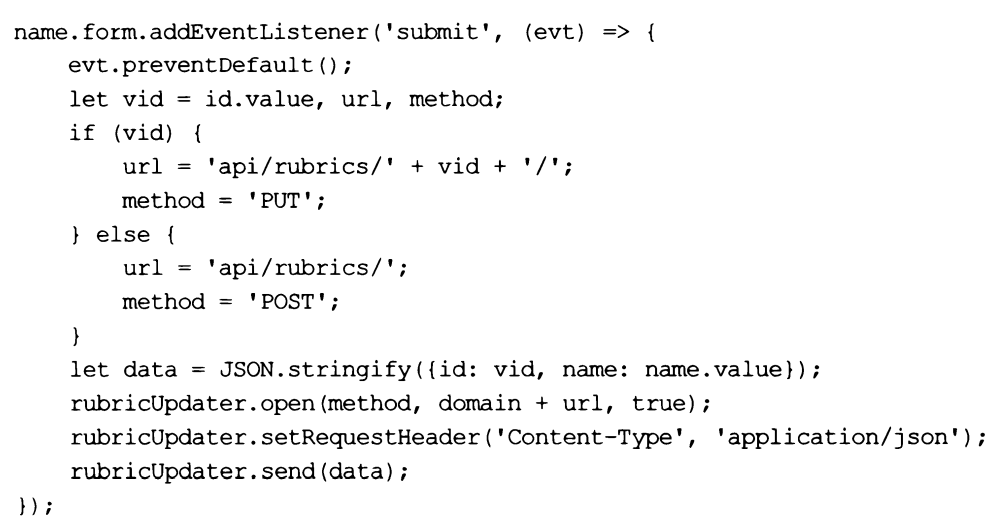


При успешном исправлении рубрики отправленный клиенту ответ будет иметь код статуса по умолчанию: 200. После успешного удаления рубрики клиент получит ответ с кодом статуса 204 (ресурс удален). Если же отправленные данные некорректны, то фронтенд получит ответ с кодом 400 (некорректный запрос).

Указанные коды статуса хранятся в переменных http\_201\_created, http\_204\_no\_CONTENT И HTTP\_400\_BAD\_REQUEST, объявленных В Модуле rest\_framework. Status.

Настало время ’’научить” фронтенд создавать и править рубрики. Откроем файл Rubrics.js и добавим в него следующий код:



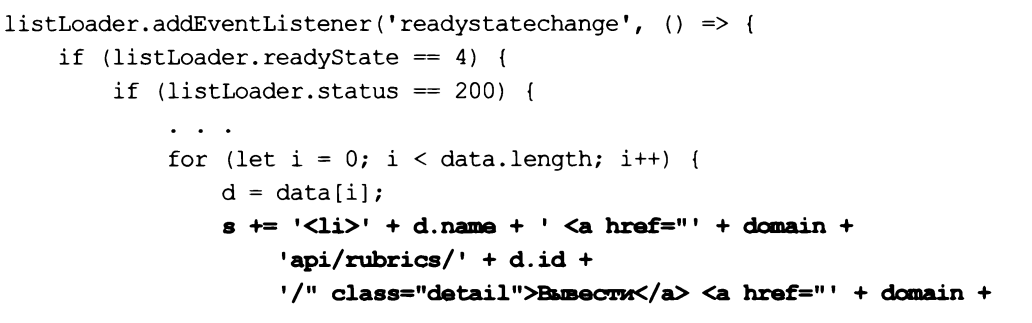


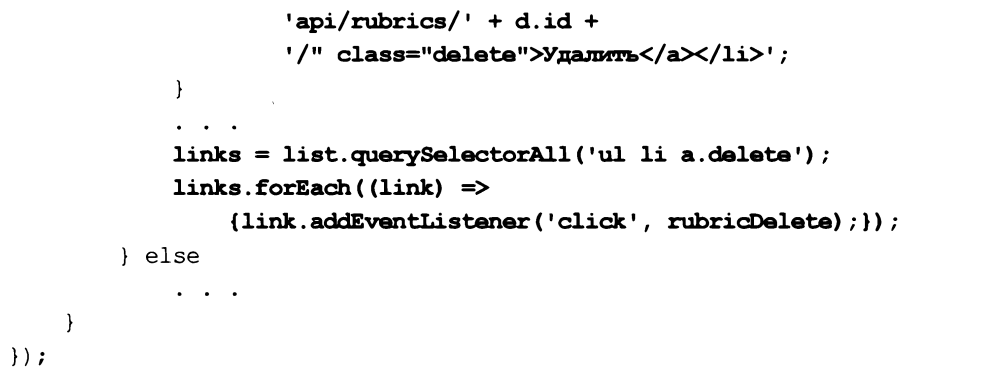
Обработчик события submit веб-формы проверяет, хранится ли в скрытом поле id ключ рубрики. Если скрытое поле хранит ключ, значит, выполняется правка уже имеющей рубрики, в противном случае в список добавляется новая рубрика. Исходя из этого, вычисляется интернет-адрес ресурса, по которому следует выполнитьзапрос, и выбирается HTTP-метод, применяемый для его отсылки. Введенные в форму данные кодируются в формат JSON, для чего они сначала представляются в виде объекта класса object, а потом "пропускаются" через статический метод Stringify о класса json. У готовящегося к отправке запроса в заголовке contenttype указывается тип отсылаемых данных: application/json, —для чего используется метод setrequestheader () класса xmlhttprequest (ЭТО нужно, чтобы django REST framework подобрала подходящий парсер — о парсерах мы поговорим позже). Наконец, готовый запрос, хранящий введенные данные, отсылается вебслужбе.

Обработчик события readystatechange после получения ответа С КОДОМ 200 ИЛИ 201 (т. е. после успешного исправления или добавления рубрики) обновляет перечень рубрик, очищает форму и заносит в скрытое поле id веб-формы "пустую" строку.

Так он сообщает пользователю, что все прошло нормально, и готовит форму для ввода новой рубрики.

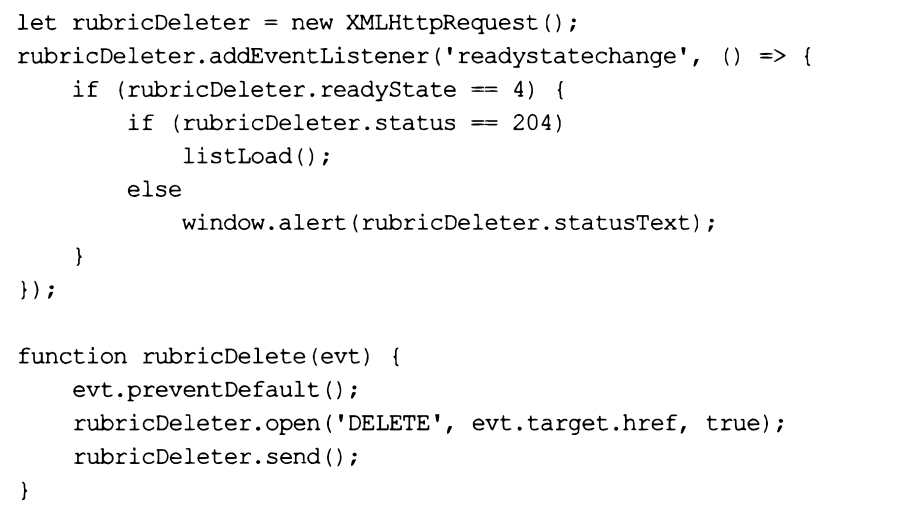
Сразу же добавим в файл rubrics.js код, реализующий удаление рубрик. Сначала сделаем так, чтобы в перечне рубрик выводились гиперссылки Удалить:





К созданным гиперссылкам привязывается обработчик события click— функция Rubricdelete (), которая и запустит удаление выбранной рубрики.

Допишем код, удаляющий рубрики:

****

Здесь обработчик события readystatechange ожидает ответа С КОДОМ 204, чтобы удостовериться, что рубрика была успешно удалена.

Перезапустим отладочный веб-сервер, обновим страницу rubrics.html в веб-обозревателе и проверим, как все работает.

**Парсеры веб-форм**

Получив от фронтенда какие-либо данные, библиотека Django REST framework пытается разобрать их, используя подходящий парсер.

Парсер — это класс, выполняющий разбор переданных фронтендом данных и их преобразование в объекты языка Python, пригодные для дальнейшей обработки.

Парсер задействуется программным ядром библиотеки перед вызовом контроллера — таким образом, последний получит уже обработанные данные.

В составе Django REST framework поставляются три наиболее интересных для нас класса парсеров:

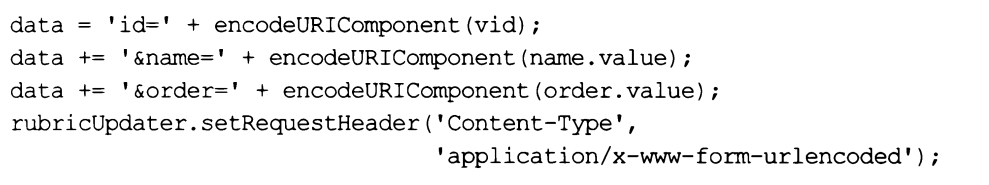
□ jsonparser — обрабатывает данные, представленные в формате JSON (mimeтип application/j son);

□ formparser — обрабатывает данные ИЗ обычных веб-форм (MIME-ТИП application/Х—www—form-urlencoded);

□ muitipartparser— обрабатывает данные из веб-форм, выгружающих файлы (М1МЕ-тип multipart/form-data).

По умолчанию активны все эти три класса парсеров. Библиотека выбирает нужный парсер, основываясь на MIME-типе переданных фронтендом данных, который записывается в заголовке Content-Type запроса. Именно поэтому в коде фронтенда перед отправкой данных необходимо указать их MIME-тип.

Ранее мы пересылали от клиента данные, закодированные в формате JSON, и обрабатывались они парсером jsonparser. Однако мы можем переслать данные в формате обычных форм:



Такие данные будут обработаны парсером formparser.