**Тема занятия № 39: Модуль 21. Разграничение доступа. Расширенные инструменты**

1. Настройки проекта, касающиеся разграничения доступа

Подсистема разграничения доступа, реализованная в Django, предоставляет ряд инструментов, полезных при программировании на низком уровне.

Немногочисленные настройки, затрагивающие работу подсистемы разграничения доступа, записываются в модуле settings.py пакета приложения:

□ auth password validators — список валидаторов, применяемых при валидации пароля, который пользователь заносит при регистрации. Каждый элемент списка задает один валидатор и должен представлять собой словарь с элементами name (задает имя класса валидатора в виде строки) и options (словарь с дополнительными параметрами валидатора).

Валидаторы, приведенные в списке, задействуются в формах для смены и сброса пароля, в командах создания суперпользователя и смены пароля. Во всех прочих случаях они никак не используются.

Значение по умолчанию — "пустой” список, однако сразу при создании проекта

Этому параметру присваивается список из всех валидаторов, поставляемых в составе Django с параметрами по умолчанию;

□ authentication backends — список имен классов, реализующих аутентификацию и авторизацию, представленных в виде строк. По умолчанию — список с единственным элементом "django.contrib.auth.backends.modelbackend" (ЭТОТ класс реализует аутентификацию и авторизацию пользователей из списка, хранящегося в модели);

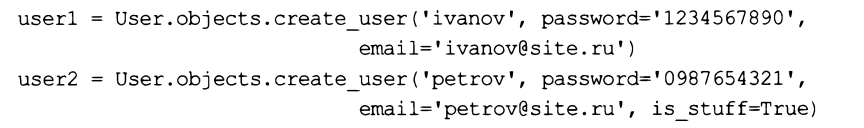
□ auth user model— имя класса модели, хранящей список зарегистрированных пользователей, в виде строки. По умолчанию: "auth.user" (стандартная модель User).

1. Работа с пользователями

**Создание пользователей**

Для создания пользователя применяются два описанных далее метода, поддерживаемые классом диспетчера записей usermanager, который используется в модели User:

□ create\_user (<имя>, password=<naponb>[, етаи=<адрес электронной почты>][, дополнительные поля>]) — создает и сразу сохраняет нового пользователя с указанными именем, паролем и адресом электронной почты (если он задан). Также могут быть указаны значения для дополнительных полей, которые будут сохранены в модели пользователя. Созданный пользователь делается активным (полю is active присваивается значение True) и возвращается в качестве результата. Примеры:

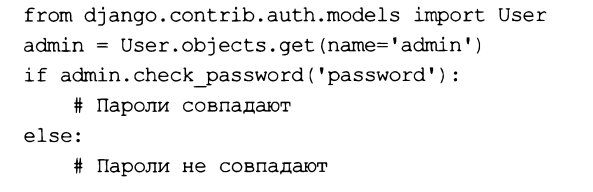
****

□ create\_superuser (<имя>, <адрес электронной почты>, <пароль>[, Дополнительные поля>]) — создает и сразу сохраняет нового суперпользователя с указанными именем, паролем и адресом электронной почты (если ОН задан). Также могут быть указаны значения для дополнительных полей модели пользователя. Созданный суперпользователь делается активным (полю is active присваивается значение True) и возвращается в качестве результата.

**Работа с паролями**

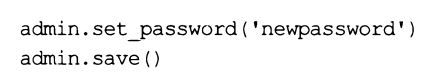
Еще четыре метода, поддерживаемые моделью user, предназначены для работы с паролями:

О check\_password(<пароль>} — возвращает True, если заданный пароль совпадает с хранящимся в списке, и False — в противном случае:

****

□ set\_password (<новый пароль>) —задает для текущего пользователя новый пароль.

Сохранение пользователя не выполняет. Пример:

****

□ set unusabie password () — задает для текущего пользователя недействительный пароль. При проверке такого пароля функцией check\_password() последняя всегда будет возвращать False. Сохранение пользователя не выполняет.

Недействительный пароль указывается у тех пользователей, для которых процедура входа на сайт выполняется не средствами Django, а какой-либо сторонней библиотекой — например, Python Social Auth, описываемой далее;

□ has\_usabie\_password () — возвращает True, если текущий пользователь имеет действительный пароль, и False, если его пароль недействителен (была вызвана функция set\_unusable\_password () ).

1. Аутентификация и выход с сайта

Аутентификация, т. е. вход на сайт, с применением низкоуровневых инструментов выполняется в два этапа: поиск пользователя в списке и собственно вход. Здесь нам понадобятся три функции, объявленные В модуле django.contrib.auth.

Для поиска пользователя по указанным им на странице входа имени и паролю применяется функция authenticate ():



Запрос должен быть представлен экземпляром класса httprequest. Если пользователь с указанными именем и паролем существует в списке, функция возвращает представляющую его запись модели user. В противном случае возвращается None. собственно ВХОД выполняется ВЫЗОВОМ функции login (<запрос>, <пользователь>) .

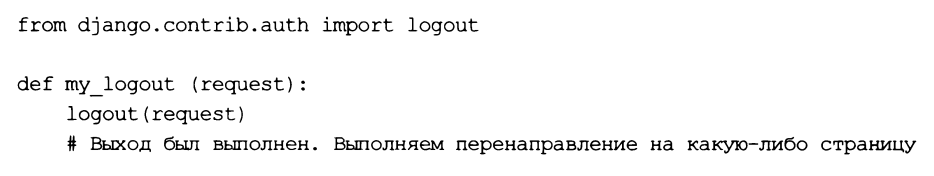
Запрос должен быть представлен экземпляром класса httprequest, а пользователь.

От имени которого выполняется вход, — записью модели user.

Вот пример кода, получающего в POST-запросе данные из формы входа и выполняющего вход на сайт:

****

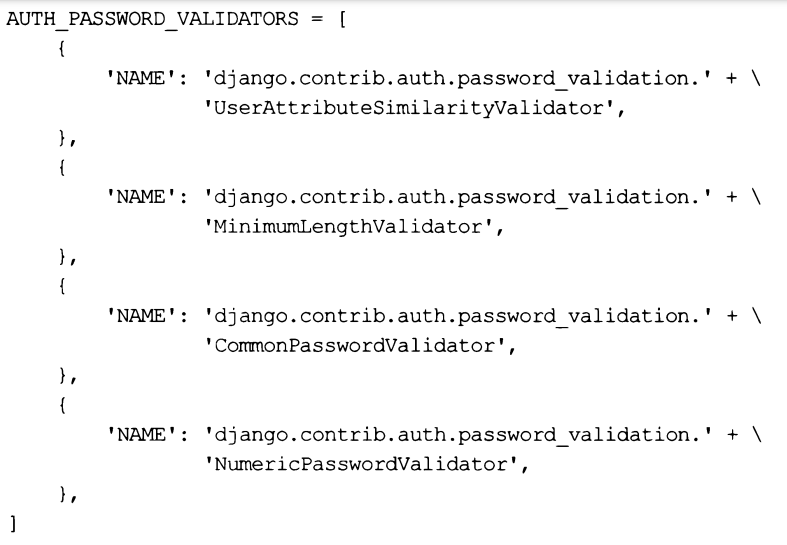
Выход с сайта выполняется вызовом функции logout(<запрос>). Запрос должен быть представлен экземпляром класса httprequest. Пример:



1. Валидация паролей

Ранее описывался параметр auth\_password\_validators настроек проекта, задающий набор валидаторов паролей. Эти валидаторы будут работать в формах для смены и сброса пароля, в командах создания суперпользователя и смены пароля.

Значение этого параметра по умолчанию — "пустой" список. Однако сразу при создании проекта для него задается такое значение:

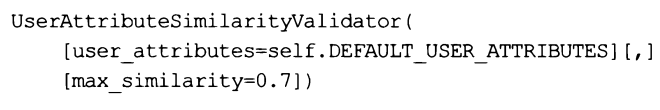


Это список, включающий все (четыре) стандартные валидаторы, поставляемые в составе Django.

**Стандартные валидаторы паролей**

Все стандартные валидаторы реализованы в виде классов и объявлены в модуле Django.contrib.auth.password\_validation:

□ userattributesimilarityvalidator () — позволяет удостовериться, ЧТО пароль в достаточной степени отличается от остальных сведений о пользователе. Формат вызова конструктора:

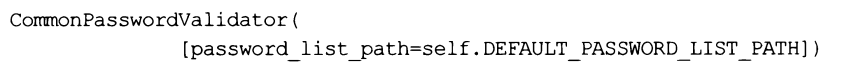
****

Необязательный параметр user attributes задает последовательность имен полей модели user, из которых будут браться сведения о пользователе для сравнения с паролем, имена полей должны быть представлены в виде строк. По умолчанию берется кортеж, хранящийся в атрибуте default user attributes этого же класса И перечисляющий ПОЛЯ username, first\_name, last name И email.

Необязательный параметр max similarity задает степень схожести пароля со значением какого-либо из полей, указанных в последовательности userattributes. Значение параметра должно представлять собой вещественное число от о (будут отклоняться все пароли без исключения) до 1 (будут отклоняться только пароли, полностью совпадающие со значением поля). По умолчанию установлено значение 0.7;

□ minimumlengthvalidator ([min\_length=8] ) — проверяет, не оказались ЛИ длина пароля меньше заданной в параметре min length (по умолчанию 8 символов);

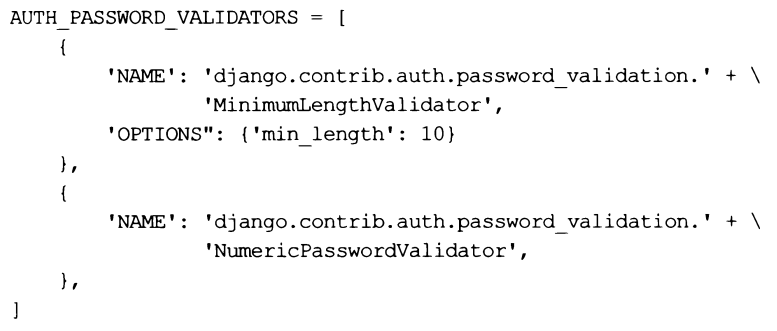
□ commonpasswordvalidator () — проверяет, не входит ли пароль в указанный перечень наиболее часто встречающихся паролей. Формат вызова конструктора:

****

Необязательный параметр password\_iist\_path задает полный путь к файлу со списком недопустимых паролей. Этот файл должен быть сохранен в текстовом формате, а каждый из паролей должен находиться на отдельной строке. По умолчанию используется файл с порядка 1000 паролей, путь к которому хранится в атрибуте default\_password\_list\_path класса;

□ numericpasswordvaiidator — проверяет, не содержит ли пароль одни цифры.

Вот пример кода, задающего новый список валидаторов:

****

Новый СПИСОК содержит ТОЛЬКО валидаторы minimumlengthvalidator и Numericpasswordvaiidator, причем для первого указана минимальная длина пароля 10 символов.

**Написание своих валидаторов паролей**

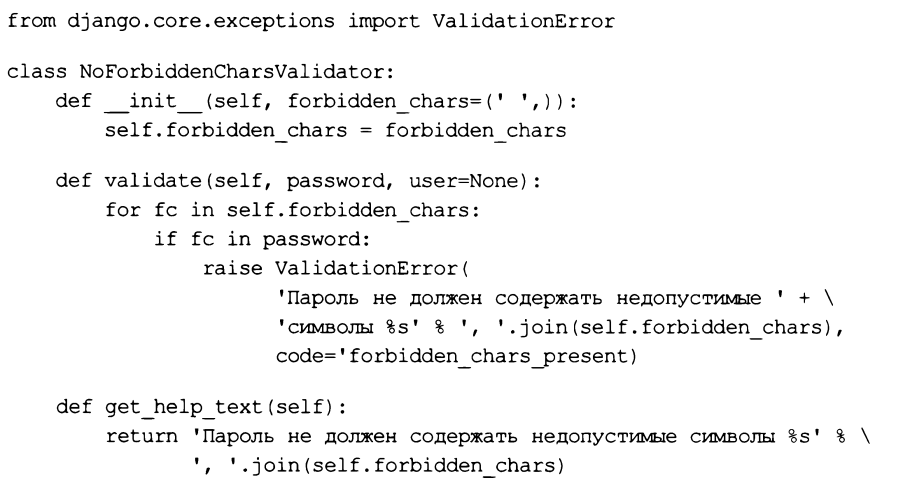
Валидаторы паролей обязательно должны быть реализованы в виде классов, поддерживающих два метода:

□ validate (self, password, user=None) — выполняет вэлидацию пароля, получаемого с параметром password. С необязательным параметром user может быть получен текущий пользователь.

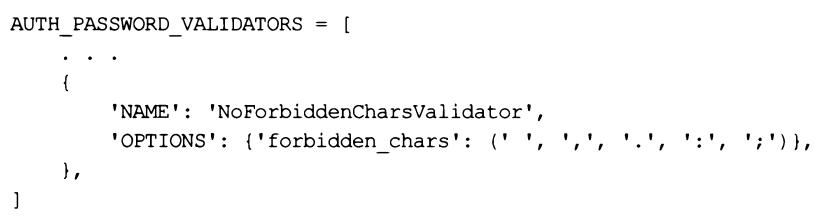
Метод не должен возвращать значения. Если пароль не проходит валидацию, то следует возбудить исключение validationerror ИЗ модуля django. Core.exceptions;

□ get heip text (self) — должен возвращать строку с требованиями к вводимому паролю.

Приведен код валидатора noforbiddencharsvaiidator, проверяющего, не содержатся ли в пароле недопустимые символы, заданные в параметре Forbidden\_chars.

****

Такой валидатор может быть использован наряду со стандартными:

****

**Выполнение валидации паролей**

Валидаторы из параметра auth\_password\_validators используются в ограниченном количестве случаев. Чтобы осуществить валидацию пароля там, где нам нужно (например, в написанной нами самими форме), мы прибегнем к набору функций из модуля django.contrib.auth.password\_validation:

□ validate\_password(<пароль>[, user=None][, password\_validators=None]) —выполняет валидацию пароля. Если пароль не проходит валидацию, возбуждает исключение validationerror;

□ password\_validators\_help\_texts ( [password\_validators=None] ) — возвращает список строк, содержащий требования к вводимым паролям от всех валидаторов. Строка с такими требованиями возвращается методом get heip text () валидатора;

□ password\_validators\_help\_texts\_html ( [password\_validators=None] ) — ТО же самое, что и password vaiidators heip textо, но возвращает HTML-код, создающий маркированный список со всеми требованиями;

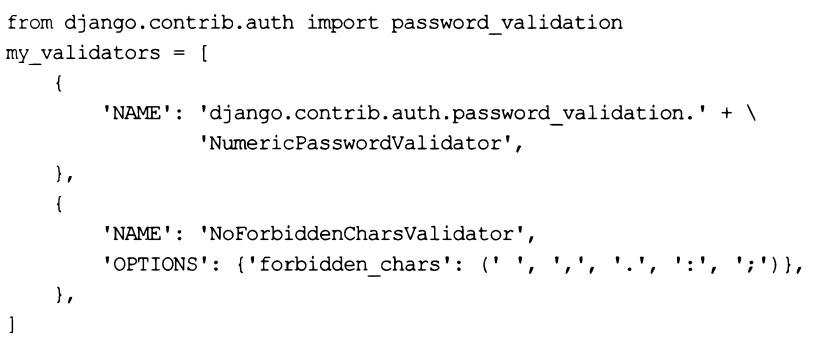
□ password\_changed (<пароль>[, user=None] [, password\_validators=None]) —сообщает всем валидаторам, что пароль пользователя изменился.

Вызов этой функции следует выполнять сразу после каждой смены пароля, если для этого не использовалась функция set password () после выполнения функции set passwordo функция password—Changed () вызывается автоматически.

Необязательный параметр user, принимаемый большинством функций, задает пользователя, чей пароль проходит валидацию. Это значение может понадобиться некоторым валидаторам.

Необязательный параметр password validators, поддерживаемый всеми этими функциями, указывает список валидаторов, которые будут заниматься валидацией пароля. Если он не указан, используется список из параметра auth\_password\_Validators настроек проекта.

Чтобы сформировать свой список валидаторов, следует применить функцию Get\_password\_validators (<настройки валидаторов>). Настройки валидаторов указываются в том же формате, что и значение параметра auth\_password\_validators настроек проекта. Пример:

****

1. Библиотека Python Social Auth: регистрация и вход через социальные сети

В настоящее время очень и очень многие пользователи Интернета являются подписчиками какой-либо социальной сети, а то и не одной. Неудивительно, что появились решения, позволяющие выполнять регистрацию в списках пользователей различных сайтов и вход на них посредством социальных сетей. Одно из таких решений — дополнительная библиотека Python Social Auth.

Python Social Auth позволяет выполнять вход посредством более чем 100 социальных сетей и интернет-сервисов, включая ’’вконтакте”, Facebook, Twitter, github, Instagram и др. Отметим, что она поддерживает не только Django, но и ряд других веб-фреймворков, написанных на Python.

На заметку!

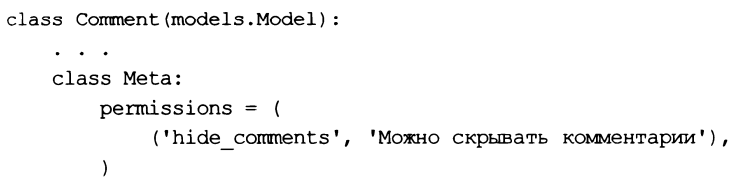
Полное руководство по Python Social Auth располагается здесь:

Https://python-social-auth.readthedocs.io/.

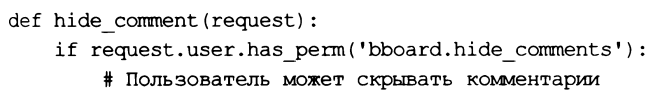
1. Создание своих прав пользователя

Ранее описывались права, определяющие операции, которые пользователь может выполнять над записями какой-либо модели. Изначально для каждой модели создаются четыре стандартных права: на просмотр, добавление, правку и удаление записей.

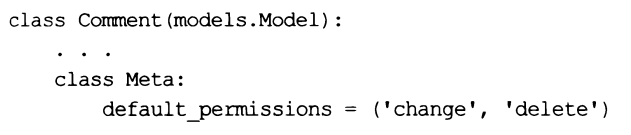
Для любой модели можно создать дополнительный набор произвольных прав. Для этого мы воспользуемся параметром permissions, задаваемым для самой модели — во вложенном классе Meta. В качестве его значения указывается список или кортеж, каждый элемент которого описывает одно право и также представляет собой кортеж из двух элементов: обозначения, используемого самим Django, и наименования, предназначенного для вывода на экран. Пример:

****

Обрабатываются эти права точно так же, как и стандартные. В частности, можно программно проверить, имеет ли текущий пользователь право скрывать комментарии:

****

Можно также изменить набор стандартных прав, создаваемых для каждой модели самим Django. Стандартные права указываются в параметре модели default permissions в виде списка или кортежа, содержащего строки с их наименованиями: "view” (просмотр), "add” (добавление), "change” (правка) и "delete" (удаление). Вот пример указания у модели только прав на правку и удаление записей:

****

Значение параметра default\_permissions ПО умолчанию: (’ view’, ’ add’, ’ change ’, ’Delete’) —т. е. полный набор стандартных прав.