**Тема занятия № 19: Разграничение доступа**

1. Как работает подсистема разграничения доступа

К внутренним данным сайта, хранящимся в его информационной базе, следует допускать только посетителей, записанных в особом списке — зарегистрированных пользователей, или просто пользователей. Также нужно учитывать, что какому-либо пользователю может быть запрещено работать с определенными данными — иначе говоря, принимать во внимание права, или привилегии, пользователя.

Допущением или недопущением посетителей к работе с внутренними данными сайта на основе того, зарегистрирован ли он в списке пользователей, и с учетом его прав, в Django-сайте занимается подсистема разграничения доступа.

**Как работает подсистема разграничения доступа**

Если посетитель желает получить доступ к внутренним данным сайта (например, чтобы добавить объявление или создать новую рубрику), то он предварительно должен войти на особую веб-страницу и занести в представленную там форму свои регистрационные данные: имя и пароль. Подсистема разграничения доступа проверит, имеется ли пользователь с такими именем и паролем в списке пользователей (т. е. является ли он зарегистрированным пользователем). Если такой пользователь в списке обнаружился, подсистема помечает его как выполнившего процедуру аутентификации, или входа, на сайт. В противном случае посетитель получит сообщение о том, что его в списке нет и данные сайта ему недоступны.

Когда посетитель пытается попасть на страницу для работы с внутренними данными сайта, подсистема разграничения доступа проверяет, выполнил ли он процедуру входа и имеет ли он права на работу с этими данными, — выполняет авторизацию.

Если посетитель прошел авторизацию, то он допускается к странице, в противном случае получает соответствующее сообщение.

Закончив работу с внутренними данными сайта, пользователь выполняет процедуру выхода с сайта. При этом подсистема разграничения доступа помечает его как не выполнившего вход. Теперь, чтобы снова получить доступ к внутренним данным сайта, посетитель вновь должен выполнить вход.

Но как посетители заносятся в список пользователей? Во-первых, их может добавить туда (выполнить их регистрацию) один из пользователей, имеющих права на работу с этим списком, — такое обычно практикуется в корпоративных решениях с ограниченным кругом пользователей. Во-вторых, посетитель сможет занести себя в список самостоятельно, зайдя на страницу регистрации и введя необходимые данные— в первую очередь, свои регистрационное имя и пароль. Этот способ применяется на общедоступных интернет-ресурсах.

На тот случай, если какой-либо из зарегистрированных пользователей забыл свой пароль, на сайтах часто предусматривают процедуру восстановления пароля. Забывчивый пользователь заходит на особую страницу и вводит свой адрес электронной почты. Подсистема разграничения доступа ищет в списке пользователя с таким адресом и отправляет ему особое письмо с гиперссылкой, ведущей на страницу, где пользователь сможет задать новый пароль.

1. Подготовка подсистемы разграничения доступа

**Настройка подсистемы разграничения доступа**

Настройки подсистемы разграничения доступа записываются, как обычно, в модуле settings.py пакета конфигурации.

Чтобы эта подсистема успешно работала, нужно сделать следующее:

□ проверить, записаны ли в списке зарегистрированных в проекте приложений (параметр INSTALLED\_APPS) приложения django.contrib.auth И django.contrib.Contenttypes;

□ проверить, записаны ли в списке зарегистрированных посредников (параметр MIDDLEWARE) посредники dj ango. Contrib. Sessions .middleware. Sessionmiddleware и django.contrib.auth.middleware.authenticationmiddleware.

Впрочем, во вновь созданном проекте все эти приложения и посредники уже занесены в соответствующие списки.

Кроме того, на работу подсистемы влияют следующие параметры:

□ login url— интернет-адрес, на который будет выполнено перенаправление после попытки попасть на страницу, закрытую от неавторизованных посетителей (гостей). Также можно указать имя маршрута. Значение по умолчанию: "/accounts/login/".

Обычно в этом параметре указывается интернет-адрес страницы входа на сайт;

□ login redirect url — интернет-адрес, на который будет выполнено перенаправление после успешного входа на сайт. Также можно указать имя маршрута. Значение ПО умолчанию: "/accounts/profile/".

Если переход на страницу входа на сайт был вызван попыткой попасть на страницу, закрытую от неавторизованных посетителей, то Django автоматически выполнит перенаправление на страницу входа, добавив к ее интернет-адресу getпараметр next, в котором запишет интернет-адрес страницы, на которую хотел попасть посетитель. После успешного входа будет выполнено перенаправление на интернет-адрес, сохраненный в этом GET-параметре. Если же такого параметра обнаружить не удалось, перенаправление выполнится по интернет-адресу из параметра login\_redirect\_url;

□ logout redirect url— интернет-адрес, на который будет выполнено перенаправление после успешного выхода с сайта. Также можно указать имя маршрута или значение None— в этом случае будет выведена страница выхода с сайта.

Значение по умолчанию — None;

□ password reset timeout days — число дней, в течение которых будет действителен интернет-адрес сброса пароля, отправленный посетителю в электронном письме;

□ authentication backends— список путей к модулям бэкендов, выполняющих аутентификацию и авторизацию, которые задаются в виде строк {бэкенд — это серверная часть какой-либо программной подсистемы). Значение по умолчанию:

[ ”dj ango. Contrib. Auth. Backends . Modelbackend" ] (единственный поставляемый в составе Django аутентификационный бэкенд, хранящий список пользователей в таблице базы данных).

Еще с несколькими параметрами, касающимися работы низкоуровневых механизмов аутентификации и авторизации.

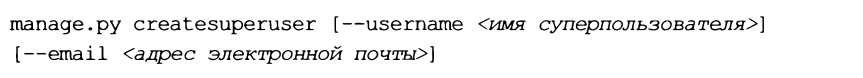
Внимание!

Перед использованием подсистемы разграничения доступа требуется хотя бы раз выполнить миграции. Это необходимо для того, чтобы Django создал в базе данных таблицы списков пользователей, групп и прав.

**Создание суперпользователя**

Суперпользователь— это зарегистрированный пользователь, имеющий права на работу со всеми данными сайта, включая список пользователей.

Для создания суперпользователя применяется команда createsuperuser утилиты Manage.py:



После отдачи этой команды утилита manage.py запросит имя создаваемого пользователя, его адрес электронной почты и пароль, который потребуется ввести дважды.

Поддерживаются два необязательных ключа командной строки:

□ —username — задает имя создаваемого суперпользователя. Если указан, то утилита manage.py не будет запрашивать имя;

О —email — задает адрес электронной почты суперпользователя. Если указан, то утилита manage.py не будет запрашивать этот адрес.

**Смена пароля пользователя**

В процессе разработки сайта может потребоваться сменить пароль у какого-либо из пользователей (вследствие забывчивости или по иной причине). Для такого случая утилита manage.py предусматривает команду changepassword:



После ее отдачи будет выполнена смена пароля пользователя с указанным именем или, если таковое не указано, текущего пользователя (выполнившего вход на сайт в данный момент). Новый пароль следует ввести дважды — для надежности.

1. Работа со списками пользователей и групп

Административный веб-сайт Django предоставляет удобные средства для работы со списками пользователей и групп (разговор о них пойдет позже). Оба списка находятся в приложении Пользователи и группы на главной странице административного сайта.

**Список пользователей**

Для каждого пользователя из списка пользователей мы можем указать следующие сведения:

□ имя, которое он будет вводить в соответствующее поле ввода в форме входа;

□ пароль.

При создании пользователя на странице будут присутствовать два поля для указания пароля. В эти поля нужно ввести один и тот же пароль (это сделано для надежности).

При правке существующего пользователя вместо поля ввода пароля будет выведен сам пароль в закодированном виде. Сменить пароль можно, щелкнув на расположенной под закодированным паролем гиперссылке;

□ настоящее имя (необязательно);

□ настоящая фамилия (необязательно);

□ адрес электронной почты;

□ является ли пользователь активным. Только активные пользователи могут выполнять вход на сайт;

□ имеет ли пользователь статус персонала. Только пользователи со статусом персонала имеют доступ к административному сайту Django. Однако для получения доступа к страницам, не относящимся к административному сайту, в том числе закрытым для гостей, статус персонала не нужен;

□ является ли пользователь суперпользователем;

□ список прав, имеющихся у пользователя.

Для указания списка прав предусмотрен элемент управления в виде двух списков и четырех кнопок в левом списке выводятся все доступные для пользователей права. Представляющие их пункты списка имеют следующий вид:

<приложение> | <моделъ> | Сап <оперсщия> <модель>приложение всегда выводится в виде своего псевдонима, модель — либо в виде имени класса, либо как заданное для него название (указывается в параметре Verbose name модели). Операция обозначается словом view (просмотр), add (добавление), change (правка) или delete (удаление).

Рассмотрим несколько примеров:

□ auth | пользователь | Can add user — право добавлять пользователей (auth — это псевдоним приложения, реализующего систему разграничения доступа);

□ auth | группа | Can delete group — право удалять группы пользователей;

□ bboard | Объявление | Can add bb — право добавлять объявления;

□ bboard | Рубрика | Can change Рубрика — право править рубрики.

В правом списке показываются права, уже имеющиеся у пользователя. Выводятся они в точно таком же виде.

Оба списка (и левый, и правый) предоставляют возможность выбора произвольного числа пунктов:

□ чтобы предоставить пользователю какие-либо права, следует выбрать их в левом списке и нажать расположенную между списками кнопку со стрелкой вправо;

□ чтобы удалить права, данные пользователю ранее, нужно выбрать их в правом списке и нажать расположенную между списками кнопку со стрелкой влево;

□ чтобы дать пользователю какое-либо одно право, достаточно найти его в левом списке и щелкнуть на нем двойным щелчком;

□ чтобы удалить у пользователя какое-либо одно право, следует найти его в правом списке и щелкнуть на нем двойным щелчком;

□ чтобы дать пользователю все доступные права, нужно нажать находящуюся под левым списком кнопку Выбрать все;

□ чтобы удалить у пользователя все права, нужно нажать находящуюся под правым списком кнопку Удалить все.

Внимание!

Пользователь может выполнять только те операции над внутренними данными сайта, на которые он явно получил права. Модели, на которые он не имеет никаких прав, при этом вообще не будут отображаться в административном сайте.

Однако суперпользователь может выполнять любые операции над любыми моделями, независимо от того, какие права он имеет.

**Группы пользователей. Список групп**

На сайтах с большим числом зарегистрированных пользователей, выполняющих разные задачи, для быстрого указания прав у пользователей можно включать последних в группы. Каждая такая группа объединяет произвольное количество пользователей и задает для них одинаковый набор прав. Любой пользователь может входить в любое число групп.

Для каждой группы на Django-сайте указываются ее имя и список прав, которые будут иметь входящие в группу пользователи. Список прав для группы задается точно так же и с помощью точно такого же элемента управления, что и аналогичный параметр у отдельного пользователя.

Для указания групп, в которые входит пользователь, применяется такой же элемент управления.

Внимание!

При проверке прав, предоставленных пользователю, принимаются в расчет как права, заданные непосредственно для него, так и права всех групп, в которые он входит.

На заметку!

Для своих нужд Django создает в базе данных таблицы authuser (список пользователей), auth group (СПИСОК групп), authjpemission (СПИСОК прав), auth user groups (связующая таблица, реализующая связь "многие-со-многими" между списками пользователей и групп), auth user user permissions (связующая между СПИСКАМИ пользователей и Прав) и auth groupjpermissions (связующая между СПИСКАМИ групп и Прав).

1. Аутентификация и служебные процедуры

Для выполнения аутентификации, т. е. входа на сайт, и различных служебных процедур (выхода, смены и сброса пароля) Django предлагает ряд контроллеров-классов, объявленных в модуле django. Contrib. Auth. Views.

**Контроллер loginview: вход на сайт**

Контроллер-класс Loginview, наследующий от formview. Реализует вход на сайт. При получении запроса по HTTP-методу GET он выводит на экран страницу входа с формой, в которую следует занести имя и пароль пользователя.

При получении POST-запроса (т. к. после отправки формы) он ищет в списке пользователя с указанными именем и паролем. Если такой пользователь обнаружился, выполняется перенаправление по интернет-адресу, взятому из GET- или postпараметра next, или, если такой параметр отсутствует, из параметра login\_Redirectjjrl настроек проекта. Если же подходящего пользователя не нашлось, то страница входа выводится повторно.

Класс Loginview поддерживает следующие атрибуты:

□ tempiate name — путь к шаблону страницы входа в виде строки (по умолчанию:

’’Registration/login.html");

□ redirect fieid name — имя GET- или POST-параметра, из которого будет извлекаться интернет-адрес для перенаправления после успешного входа, в виде строки (по умолчанию: "next”);

□ redirect authenticated user— если True, ТО пользователи, уже выполнившие вход, при попытке попасть на страницу входа будут перенаправлены по интернет-адресу, взятому из GET- или POST-параметра next или параметра Login redirect url настроек проекта. Если False, то пользователи, выполнившие вход, все же смогут попасть на страницу входа.

Значение этого атрибута по умолчанию — False, и менять его на True следует с осторожностью, т. к. Это может вызвать нежелательные эффекты. В частности, если пользователь попытается попасть на страницу с данными, для работы с которыми у него нет прав, он будет перенаправлен на страницу входа. Но если атрибут redirectjauthenticated\_user имеет значение True, ТО сразу же после ЭТОГО будет выполнено перенаправление на страницу, с которой пользователь попал на страницу входа. В результате возникнет зацикливание, которое закончится аварийным завершением работы сайта;

□ extra context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

□ success uri aiiowed hosts— множество, задающее хосты, на которые можно выполнить перенаправление после успешного входа, в дополнение к текущему хосту (по умолчанию — ’’пустое” множество);

□ authentication\_fonn— ссылка на класс формы входа (по умолчанию— класс Authenticationform ИЗ модуля dj ango. Contrib. Auth. Forms).

Контекст шаблона, создающий страницу входа, содержит следующие переменные:

□ form —форма для ввода имени и пароля;

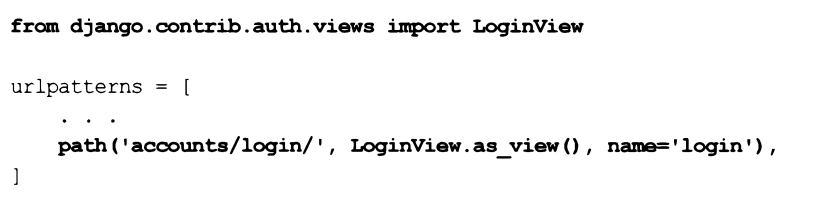
□ next— интернет-адрес, на который будет выполнено перенаправление после успешного входа.

Внимание!

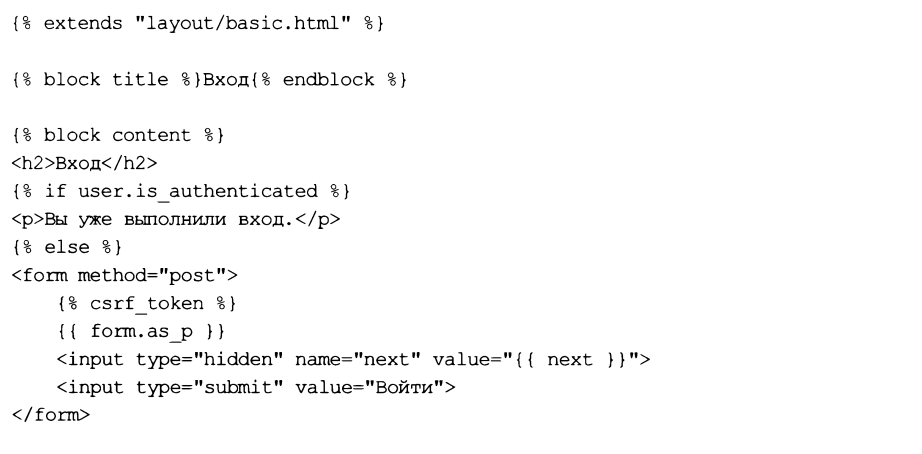
Шаблон registration\login.html изначально не существует ни в одном из зарегистрированных в проекте приложений, включая встроенные во фреймворк. Поэтому мы можем просто создать такой шаблон в одном из своих приложений.

Шаблоны, указанные по умолчанию во всех остальных контроллерах-классах, которые будут рассмотрены в этом разделе, уже существуют во встроенном приложении Django.contrib.admin, реализующем работу административного сайта Django. Поэтому в остальных контроллерах следует задать другие имена шаблонов (в противном случае будут использоваться шаблоны административного сайта).

Чтобы реализовать процедуру аутентификации (входа), достаточно добавить в список маршрутов уровня проекта (он объявлен в модуле uris.py пакета конфигурации) такой элемент:

****

Код простейшего шаблона страницы входа registration\login.html приведен.





Поскольку разработчики Django предостерегают от автоматического перенаправления со страницы входа пользователей, уже выполнивших вход, придется использовать другие средства предотвращения повторного входа на сайт. В контекст любого шаблона помещается переменная user, хранящая объект текущего пользователя.

Атрибут is authenticated этого объекта хранит True, если пользователь уже вошел на сайт, и False — если еще нет. С учетом этого можно вывести на странице либо форму входа, либо сообщение о том, что вход уже был выполнен.

Также в форме входа следует создать скрытое поле с именем next и занести в него интернет-адрес для перенаправления при успешном входе.

**Контроллер logoutview’. Выход с сайта**

Контроллер-класс Logoutview, наследующий ОТ Templateview, реализует выход с сайта при получении GET-запроса, после чего осуществляет перенаправление на интернет-адрес, указанный в GET-параметре next или, если такового нет, в атрибуте next page. Если значение этого атрибута равно None, то он выводит страницу с сообщением об успешном выходе.

Класс поддерживает атрибуты:

□ next page— интернет-адрес, на который будет выполнено перенаправление после успешного выхода с сайта (значение по умолчанию берется из параметра Logout redirect url настроек сайта). Также можно указать имя нужного маршрута.

Если задать этому атрибуту значение None, то перенаправление выполняться не станет, а вместо этого на экран будет выведена страница с сообщением об успешном выходе;

□ template name— путь к шаблону страницы сообщения об успешном выходе в виде строки (по умолчанию: ”registration/logged\_out.html").

Эта страница будет выведена, только если в текущем интернет-адресе отсутствует GET-параметр next и значение атрибута next page равно None;

□ redirect fieid name — имя GET-параметра, из которого будет извлекаться интернет-адрес для перенаправления после успешного выхода, в виде строки (значение по умолчанию: "next”);

□ extra context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

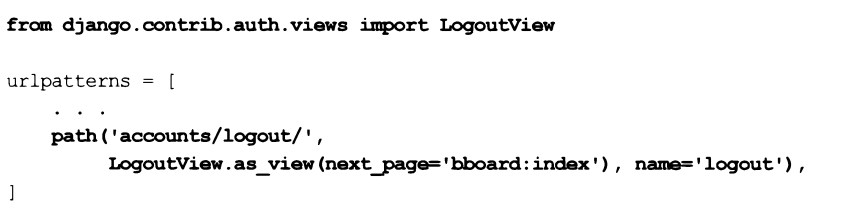
□ success uri aiiowed hosts — множество, задающее хосты, на которые можно выполнить перенаправление после успешного выхода, в дополнение к текущему хосту (по умолчанию — ’’пустое” множество).

В контексте шаблона создается переменная title, в которой хранится сообщение об успешном выходе.

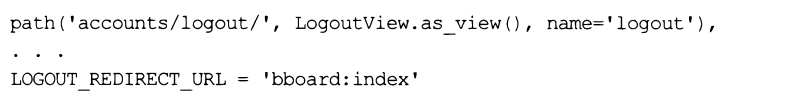
Внимание!

Шаблоны, указанные по умолчанию в этом и во всех остальных контроллерах-классах, рассматриваемых в этом разделе, уже существуют во встроенном приложении Django.contrib.admin, реализующем работу административного сайта Django. Поэтому придется указать для этих классов другие имена шаблонов (в противном случае будут использоваться шаблоны административного сайта).

Реализовать выход можно добавлением в список маршрутов уровня проекта элемента следующего вида:



Интернет-адрес перенаправления или, как в нашем случае, имя маршрута также можно записать в настройках сайта:

****

**Контроллер passwordchangeview: Смена пароля**

Контроллер-класс passwordchangeview, наследующий ОТ класса formview, выполняет смену пароля у текущего пользователя. При получении GET-запроса он выводит на экран страницу с формой, где нужно ввести старый пароль и, дважды, новый пароль. При получении POST-запроса он сохраняет введенный новый пароль и перенаправляет пользователя на страницу с сообщением об успешной смене пароля.

Вот атрибуты, поддерживаемые этим классом:

□ template name — путь к шаблону страницы с формой для смены пароля в виде строки (по умолчанию: "registration/password\_change\_form.html");

□ success uri — интернет-адрес, по которому будет выполнено перенаправление после успешной смены пароля (по умолчанию— по маршруту с именем Pas swoгd\_change\_done);

□ extra context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

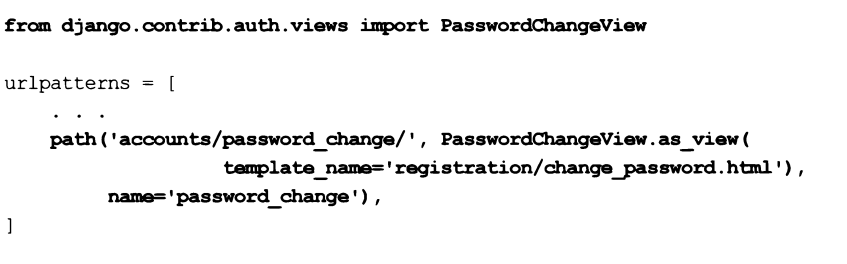
□ form class —ссылка на класс формы для ввода нового пароля (по умолчанию — класс passwordchangeform ИЗ модуля dj ango. Contrib. Auth. Forms).

Контекст шаблона содержит переменные:

□ form—форма для ввода нового пароля;

□ title — текст вида "Смена пароля", который можно использовать в заголовке страницы.

Реализовать смену пароля можно добавлением в список маршрутов уровня проекта такого элемента:



**Контроллер passwordchangedoneview: Уведомление об успешной смене пароля**

Контроллер-класс passwordchangedoneview, наследующий ОТ Templateview, ВЫВОДИТ страницу с уведомлением об успешной смене пароля.

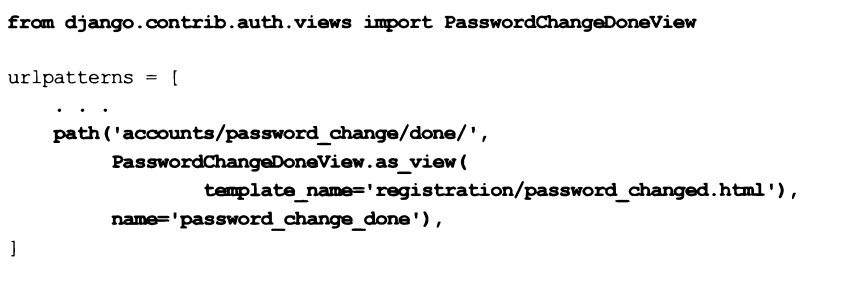
Он поддерживает атрибуты:

□ template name — путь к шаблону страницы с уведомлением в виде строки (по умолчанию: "registration/password\_change\_done.html");

□ extra context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст.

В контексте шаблона создается переменная title, в которой хранится текст уведомления.

Чтобы на сайте выводилось уведомление о смене пароля, достаточно добавить в список маршрутов уровня проекта такой элемент:

****

**Контроллер passwordresetview: Отправка письма для сброса пароля**

Контроллер-класс passwordresetview, ПРОИЗВОДНЫЙ ОТ formview, инициирует процедуру сброса пароля. При получении GET-запроса он выводит страницу с формой, в которую пользователю нужно занести свой адрес электронной почты. После получения POST-запроса он проверит существование этого адреса в списке пользователей и, если такой адрес есть, отправит по нему электронное письмо с гиперссылкой на страницу собственно сброса пароля.

Этот класс поддерживает следующие атрибуты:

□ template name— путь к шаблону страницы с формой для ввода адреса в виде строки (по умолчанию: "registration/password\_reset\_form.html”);

□ subject template name— путь к шаблону темы электронного письма (по умолчанию: ”registration/password\_reset\_subject. Txt”).

Внимание!

Шаблон темы электронного письма не должен содержать символов возврата каретки и перевода строки.

□ email template name — путь к шаблону тела электронного письма в формате обычного текста (по умолчанию: ”registration/password\_reset\_email .html”);

□ htmi emaii tempiate name — путь к шаблону тела электронного письма в формате HTML. Если None (это значение по умолчанию), письмо в формате HTML отправляться не будет;

□ success uri — интернет-адрес, по которому будет выполнено перенаправление после успешной отправки электронного письма (по умолчанию — по маршруту с именем password\_reset\_done);

□ from emaii— адрес электронной почты отправителя, который будет вставлен в отправляемое письмо (значение по умолчанию берется из параметра Default\_from\_email настроек проекта);

□ extra context — дополнительное содержимое контекста шаблона для страницы с формой. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

□ extra email context— дополнительное содержимое контекста шаблона для электронного письма. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

□ form ciass — ссылка на класс формы для ввода адреса (по умолчанию — класс Passwordresetform ИЗ модуля django. Contrib. Auth. Forms);

□ token generator — экземпляр класса, выполняющего формирование электронного жетона безопасности, который будет включен в интернет-адрес страницы сброса пароля (по умолчанию — экземпляр класса passwordresettokengenerator ИЗ МОДУЛЯ django.contrib.auth.tokens).

Контекст шаблона страницы содержит следующие переменные:

□ form—форма для ввода адреса электронной почты;

□ title — текст вида "Сброс пароля", который можно использовать в заголовке страницы.

В контексте шаблона темы и тела электронного письма создаются переменные:

□ protocol —обозначение протокола ("http" или "https");

□ domain — строка с комбинацией IP-адреса (или доменного имени, если его удастся определить) и номера TCP-порта, через который работает веб-сервер;

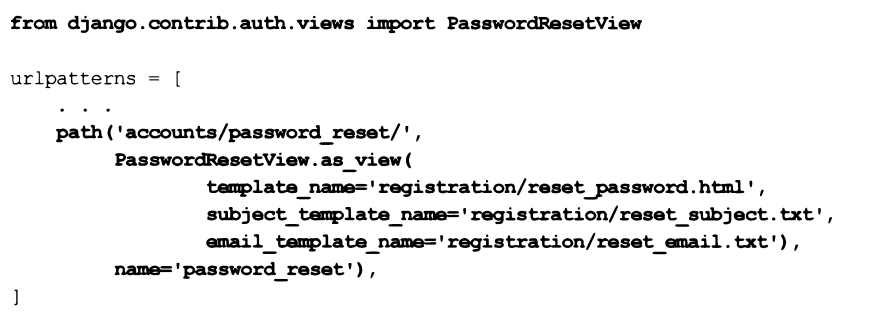
□ uid —закодированный ключ пользователя;

□ token — электронный жетон безопасности, выступающий в качестве электронной подписи;

□ email — адрес электронной почты пользователя, по которому высылается это письмо;

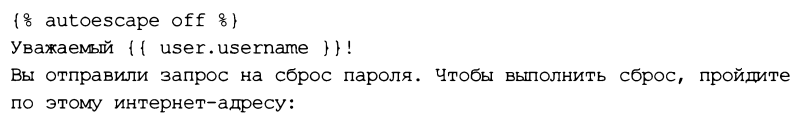
□ user — текущий пользователь, представленный экземпляром класса user.

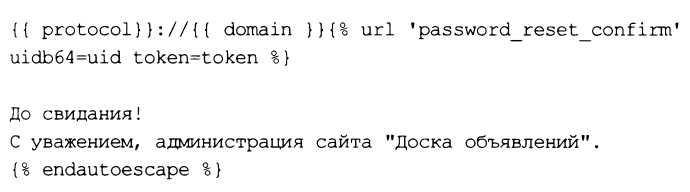
Сброс пароля реализуется добавлением в список маршрутов уровня проекта таких элементов:

****

Код шаблона registration\reset\_subject.txt, создающего тему электронного письма, код шаблона registration\reset\_email.txt, который создаст тело письма.







Код реализует интернет-адрес для перехода на страницу сброса пароля формируется путем обратного разрешения на основе маршрута с именем passwordreset confirm, который будет написан чуть позже.

**Контроллер passwordresetdoneview. Уведомление об отправке письма для сброса пароля**

Контроллер-класс passwordresetdoneview, наследующий ОТ Templateview, ВЫВОДИТ страницу с уведомлением об успешной отправке электронного письма для сброса пароля.

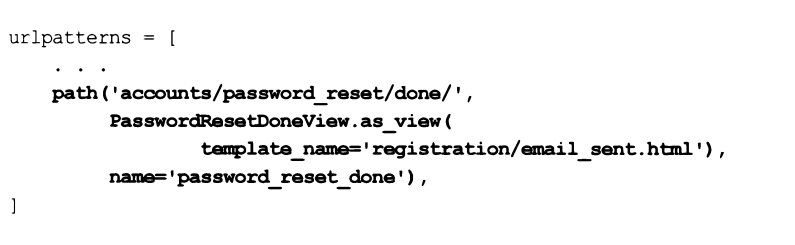
Он поддерживает атрибуты:

□ template name — путь к шаблону страницы с уведомлением в виде строки (по умолчанию: "registration/password\_reset\_done.html");

□ extra\_context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст.

В контексте шаблона создается переменная title, в которой хранится текст уведомления.

Чтобы на сайте выводилось уведомление об отправке письма, нужно добавить в список маршрутов уровня проекта такой элемент:



**Контроллер passwordresetconfirmview: Собственно сброс пароля**

Контроллер-класс passwordresetconfirmview, наследующий ОТ formview, выполняет сброс пароля. Он запускается при переходе по интернет-адресу, отправленному в письме с сообщением о сбросе пароля. С URL-параметром uidb64 он получает закодированный ключ пользователя, а с URL-параметром token— электронный жетон безопасности, значения обоих параметров должны быть строковыми. Получив GET-запрос, он выводит страницу с формой для задания нового пароля, а после получения POST-запроса производит смену пароля и выполняет перенаправление на страницу с уведомлением об успешном сбросе пароля.

Вот атрибуты, поддерживаемые этим классом:

□ template name — путь к шаблону страницы с формой для задания нового пароля в виде строки (по умолчанию: "registration/password\_reset\_confirm.html");

□ post reset login — если True, то после успешного сброса пароля будет автоматически выполнен вход на сайт, если False, то этого не произойдет (по умолчанию — False);

□ successor1 — интернет-адрес, по которому будет выполнено перенаправление после успешной смены пароля (по умолчанию — по маршруту с именем Password\_reset\_complete);

□ extra context — дополнительное содержимое контекста шаблона для страницы с формой. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

□ form class —ссылка на класс формы для сброса пароля (по умолчанию — класс Setpasswordform ИЗ модуля django. Contrib. Auth. Forms);

□ token generator— экземпляр класса, выполняющего формирование электронного жетона безопасности, который был включен в интернет-адрес, ведущий на страницу сброса пароля (по умолчанию — экземпляр класса Passwordresettokengenerator ИЗ модуля django.contrib.auth. Tokens);

□ reset url token (начиная c Django 3.0) — строковый фрагмент, который при выводе страницы с формой для задания нового пароля будет подставлен в интернет-адрес вместо электронного жетона (это делается для безопасности — чтобы никто не смог подсмотреть жетон и использовать его для атаки на сайт). По умолчанию: "set-password”.

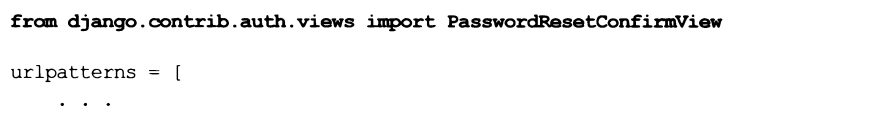
Контекст шаблона содержит следующие переменные:

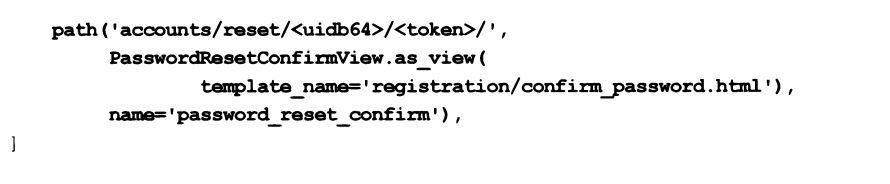
□ form —форма для ввода нового пароля;

□ vaiidiink— если True, то интернет-адрес, по которому пользователь попал на эту страницу, действителен и еще ни разу не использовался, если False, то этот интернет-адрес скомпрометирован;

□ title-- текст вида "Введите новый пароль", который МОЖНО использовать в Заголовке страницы.

Вот такой элемент нужно добавить в список маршрутов уровня проекта, чтобы на сайте заработал сброс пароля:





**Контроллер passwordresetcompleteview: Уведомление об успешном сбросе пароля**

Контроллер-класс passwordresetcompleteview, наследующий ОТ Templateview, ВЫВОДИТ страницу с уведомлением об успешном сбросе пароля.

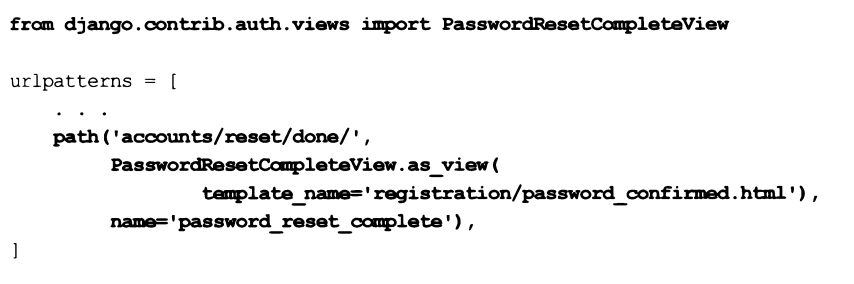
Он поддерживает атрибуты:

□ template name— путь к шаблону страницы с уведомлением в виде строки (по умолчанию: "registration/password\_reset\_complete.html”);

□ extra context — дополнительное содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст.

В контексте шаблона создается переменная title, в которой хранится текст уведомления.

Чтобы на сайте выводилось уведомление об успешном сбросе пароля, следует добавить в список маршрутов уровня проекта такой элемент:



1. Получение сведений о пользователях

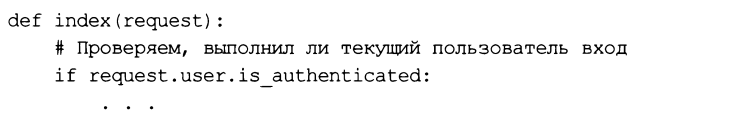
**Получение сведений о текущем пользователе**

Любой зарегистрированный пользователь представляется в Django экземпляром класса user из модуля django.contrib.auth.models. Этот класс является моделью.

Доступ к экземпляру класса user, представляющему текущего пользователя, можно получить:

□ в контроллере — из атрибута user объекта текущего запроса (экземпляра класса Request), который передается контроллеру-функции и методам контроллера-класса в первом параметре, обычно имеющем имя request.

Пример:

****

□ в шаблоне — из переменной контекста user, создаваемой обработчиком контекста dj ango.contrib.auth.context\_processors.auth:

****

Класс user поддерживает довольно большой набор полей, атрибутов и методов.

Начнем знакомство с полей:

□ username — регистрационное имя пользователя, обязательно к заполнению;

□ password — пароль в закодированном виде, обязательно к заполнению;

□ email — адрес электронной почты;

□ first name — настоящее имя пользователя;

□ last name — настоящая фамилия пользователя;

□ is active---True, если Пользователь является активным, И False — в противном случае;

□ is staff— True, если пользователь имеет статус персонала, и False— в противном случае;

□ is superuser — True, если пользователь является суперпользователем, и False — в противном случае;

□ groups — группы, в которые входит пользователь. Хранит диспетчер обратной СВЯЗИ, дающий доступ К записям связанной модели Group из модуля Django.contrib.auth.models, В которой хранятся все группы;

□ last login — дата и время последнего входа на сайт;

□ date joined — дата и время регистрации пользователя на сайте.

Полезных атрибутов класс user поддерживает всего два:

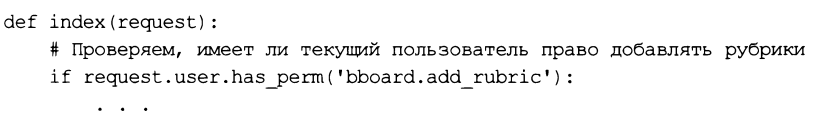
□ is authenticated— True, если текущий пользователь ВЫПОЛНИЛ ВХОД, И False, если не выполнил (т. Е. Является гостем);

□ is anonymous — True, если текущий пользователь не выполнил вход на сайт (т. е. является гостем), и False, если выполнил.

Теперь рассмотрим методы этого класса:

□ has\_perm (<право>[, obj=None]) — возвращает True, если текущий пользователь имеет указанное право, и False— в противном случае, право задается в виде строки формата "<приложение>.<операция>\_<модель>", где приложение указывается его псевдонимом, операция— строкой "view” (просмотр), "add" (добавление), "change" (правка) или "delete" (удаление), а модель —именем ее класса.

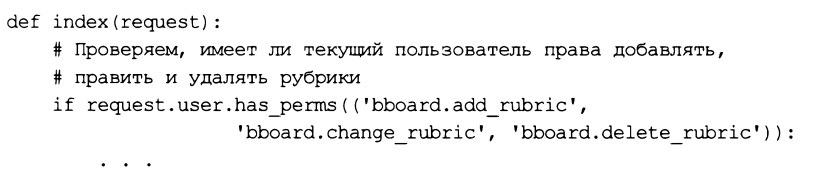
Пример:



Если в параметре obj указана запись модели, то будут проверяться права пользователя на эту запись, а не на саму модель. Эта возможность поддерживается не всеми бэкендами аутентификации (так, стандартный бэкенд django.contrib.Auth.backends.modelbackend ее не поддерживает).

Если пользователь неактивен, метод всегда возвращает False;

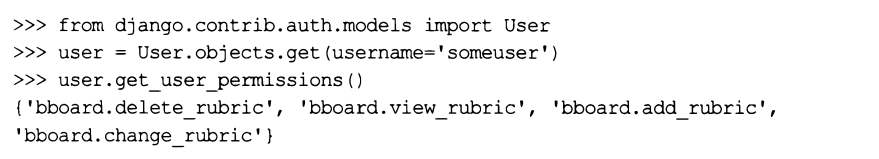
□ has\_pems (^последовательность прав>[г obj=None]) — ТО же самое, ЧТО И has\_Репп(), но возвращает True только в том случае, если текущий пользователь имеет все права ИЗ заданной последовательности'.

****

□ get user permissions ([obj=None]) (начиная c Django 3.0) — возвращает множество из прав, которыми непосредственно обладает текущий пользователь. Наименования прав представлены строками.

Если в параметре obj указана запись модели, то метод вернет права на эту запись, а не на саму модель (поддерживается не всеми бэкендами аутентификации).

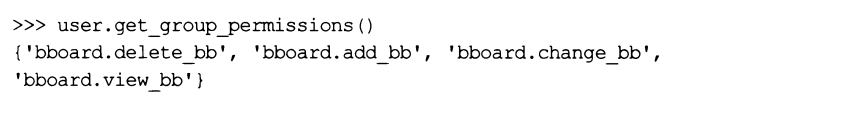
Пример:

****

□ get group permissions ([obj=None]) — возвращает множество из прав групп, в которые входит текущий пользователь. Наименования прав представлены строками.

Если в параметре obj указана запись модели, то метод вернет права на эту запись, а не на саму модель (поддерживается не всеми бэкендами аутентификации).

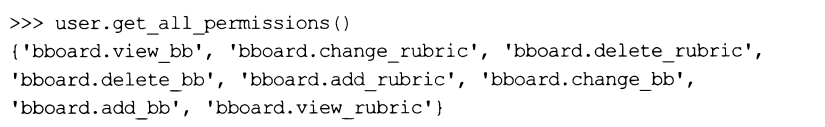
Пример:

****

□ get aii permissions ([obj=None]) — возвращает множество из прав, принадлежащих как непосредственно текущему пользователю, так и группами, в которые он входит. Наименования прав представлены строками.

Если в параметре obj указана запись модели, метод вернет права на эту запись, а не на саму модель (поддерживается не всеми бэкендами аутентификации).

Пример:



□ get username () — возвращает регистрационное имя пользователя;

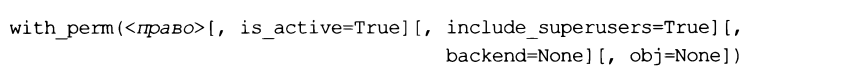
□ get fuii nameo — возвращает строку, составленную из настоящих имени и фамилии пользователя, которые разделены пробелом;

□ get short name () — возвращает настоящее имя пользователя.

Посетитель-гость представляется экземпляром класса anonymoususer из того же модуля django. Contrib. Auth.models. ЭТОТ класс ПОЛНОСТЬЮ аналогичен классу User, поддерживает те же поля, атрибуты и методы. Разумеется, гость не может выполнить вход и не имеет никаких прав.

**Получение пользователей, обладающих заданным правом**

Начиная с Django 3.0, диспетчер записей модели user поддерживает метод With perms, возвращающий перечень зарегистрированных пользователей, которые имеют заданное право:

****

Право указывается в том же формате, который применяется в методах has permo Hhas\_perms().

Если параметру is active задано значение True, то будут возвращены подходящие пользователи только из числа активных (поведение по умолчанию), если False — только из числа неактивных, если None — из числа как активных, так и неактивных.

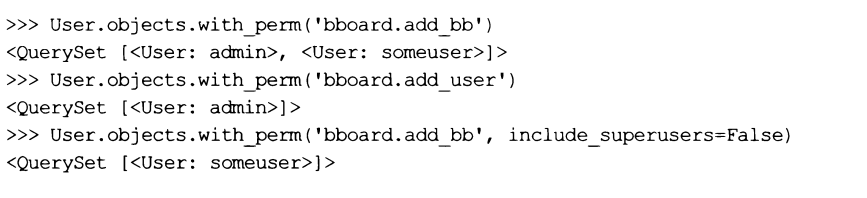
Если параметру include superusers присвоить True, то возвращенный перечень будет включать суперпользователей (поведение по умолчанию), если False — не будет включать.

Параметр backend задает аутентификационный бэкенд, используемый для поиска пользователей, обязательно из числа перечисленных в параметре authentication Backends настроек проекта. Если параметру backend задать значение None, то поиск подходящих пользователей будет выполняться с применением всех зарегистрированных в проекте бэкендов.

Если в параметре obj указана запись модели, то метод будет искать пользователей, обладающих правом на работу с этой записью, а не моделью целиком (поддерживается не всеми бэкендами аутентификации).

Метод with perm о возвращает перечень пользователей, обладающих указанным правом, в виде обычного набора записей (экземпляра класса Queryset).

Примеры:

****

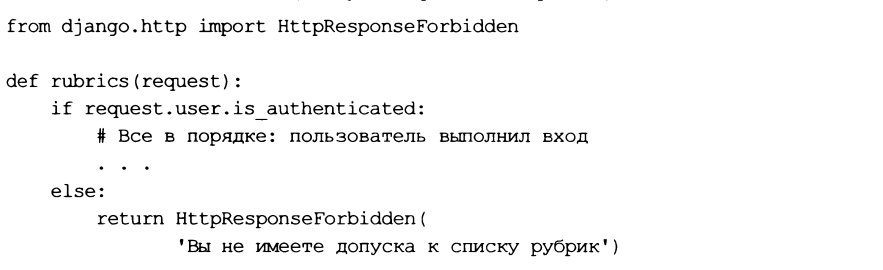
1. Авторизация

Авторизацию можно выполнить как в коде контроллеров (например, чтобы не пустить гостя на закрытую от него страницу), так и в шаблонах (чтобы скрыть гиперссылку, ведущую на закрытую для гостей страницу).

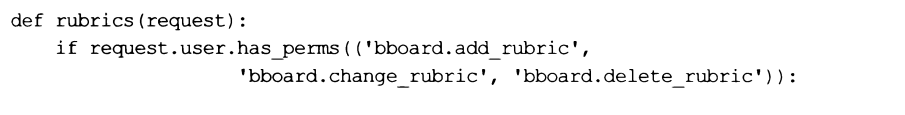
**Авторизация в контроллерах-функциях: Непосредственные проверки**

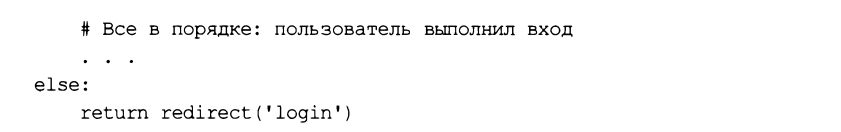
В коде контроллера, применив инструменты, мы можем непосредственно проверить, выполнил ли пользователь вход и имеет ли он достаточные права. И, на основе результатов проверок, пустить или не пустить пользователя на страницу.

Пример проверки, выполнил ли пользователь вход, с отправкой в противном случае сообщения об ошибке 403 (доступ к странице запрещен):

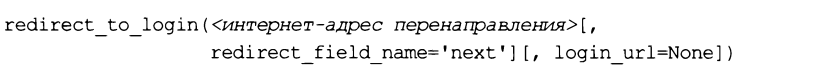
****

Вместо отправки сообщения об ошибке можно перенаправлять посетителя на страницу входа:





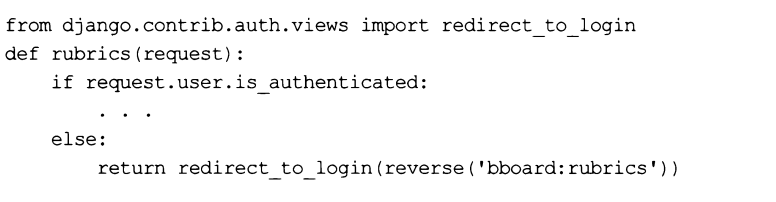
Функция redirect\_to\_login () ИЗ МОДУЛЯ django. Contrib. Auth. Views отправляет посетителя на страницу входа, а после выполнения входа выполняет перенаправление ПО заданному интернет-адресу.

****

Интернет-адрес перенаправления задается в виде строки. Необязательный параметр Redirect fieid name указывает имя GET-параметра, передающего странице входа заданный интернет-адрес (по умолчанию: "next"). Параметр login uri задает интернет-адрес страницы входа или имя указывающего на нее маршрута (по умолчанию — значение параметра login url настроек проекта).

Функция redirect\_to\_iogin() возвращает объект ответа, выполняющий перенаправление. Этот объект нужно вернуть из контроллера-функции.

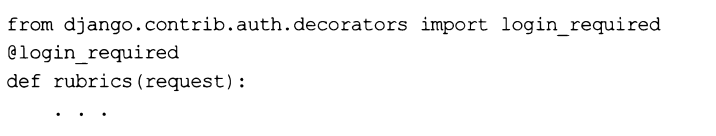
Пример:

****

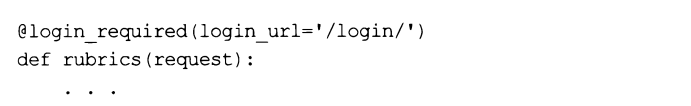
**Авторизация в контроллерах-функциях: Применение декораторов**

Во многих случаях для авторизации удобнее применять декораторы, объявленные в модуле django.contrib.auth.decorators. Они указываются у контроллера, выводящего страницу с ограниченным доступом. Всего этих декораторов три:

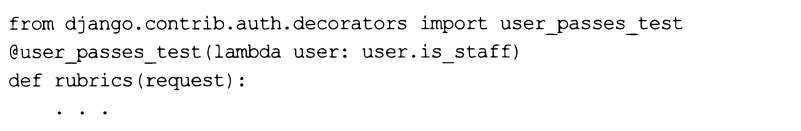
□ login\_required ( [redirect\_field\_name=’next’][,] [login\_url=None] ) — допускает к странице только пользователей, выполнивших вход. Если пользователь не выполнил вход, то выполняет перенаправление по интернет-адресу из параметра Login url настроек проекта с передачей через GET-параметр next текущего интернет-адреса. Пример:



Параметр redirect fieid name позволяет указать другое имя для GET-параметра, передающего текущий интернет-адрес, а параметр login url — другой адрес страницы входа или другое имя указывающего на нее маршрута. Пример:

****

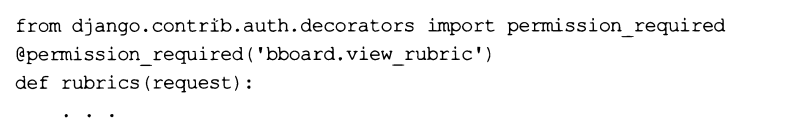
О user\_passes\_test(<проверочная функция>[, redirect\_field\_name='next’][, login\_Uri=None]) — допускает к странице только тех пользователей, кто выполнил вход и в чьем отношении проверочная функция вернет в качестве результата значение True, проверочная функция должна принимать в качестве единственного параметра экземпляр класса user, представляющий текущего пользователя. Вот пример кода, допускающего к списку рубрик только пользователей, имеющих статус персонала:

****

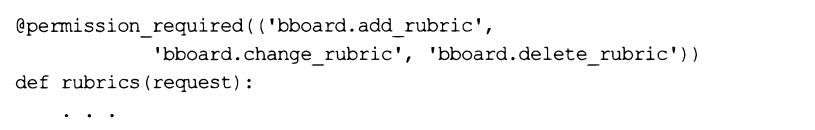
Параметр redirect fieid name позволяет указать другое имя для GET-параметра, передающего текущий интернет-адрес страницы, а параметр login uri — другой интернет-адрес страницы входа или другое имя указывающего на нее маршрута;

□ permission\_required(<права>[г raise\_exception=False][, login\_url=None]) — допускает к странице только пользователей, имеющих заданные права, права указываются в том же формате, который применяется в методах has permo и Has permso. Можно указать:

• одно право:

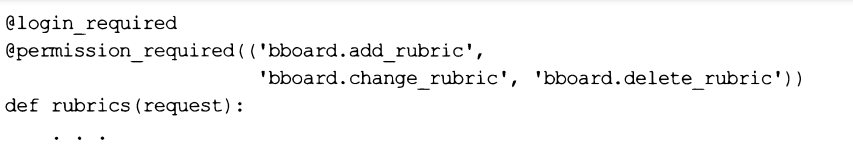
****

• последовательность из произвольного количества прав. В этом случае у текущего пользователя должны иметься все перечисленные в последовательности права. Пример:

****

Параметр login uri позволит задать другой интернет-адрес страницы входа или другое имя маршрута, указывающего на нее.

Если параметру raise exception присвоить значение True, то декоратор вместо перенаправления пользователей, не выполнивших вход, на страницу входа будет возбуждать исключение permissiondenied, тем самым выводя страницу с сообщением об ошибке 403. Если это сообщение нужно выводить только пользователям, выполнившим вход и не имеющим необходимых прав, то следует применить декоратор permission\_required () вместе С декоратором login\_required ():

****

В этом случае пользователи, не выполнившие вход, попадут на страницу входа, а те из них, кто выполнил вход, но не имеет достаточных прав, получат сообщение об ошибке 403.

**Авторизация в контроллерах-классах**

Реализовать авторизацию в контроллерах-классах можно посредством классов-примесей, объявленных в модуле django. Contrib. Auth.mixins. Они указываются в числе базовых в объявлении производных контроллеров-классов, причем в списках базовых классов примеси должны стоять первыми.

Класс accessmixin — базовый для остальных классов-примесей. Он поддерживает ряд атрибутов и методов, предназначенных для указания важных параметров авторизации:

□ login uri — атрибут, задает интернет-адрес или имя маршрута страницы входа (по умолчанию — None);

□ get iogin uri (seif) — метод, должен возвращать интернет-адрес или имя маршрута страницы входа. В изначальной реализации возвращает значение атрибута login uri или, если оно равно None, значение параметра login url настроек проекта;

□ permission denied message— атрибут, хранит строковое сообщение о возникшей ошибке (по умолчанию — ’’пустая” строка);

□ get\_permission\_denied\_message(self) — метод, должен возвращать сообщение об ошибке. В изначальной реализации возвращает значение атрибута permissiondenied—message;

□ redirect—field—name— атрибут, указывает имя GET-параметра, передающего интернет-адрес страницы с ограниченным доступом, на которую пытался попасть посетитель (по умолчанию: "next”);

О get—redirect—field—name (self) — метод, должен возвращать имя GET-параметра, передающего интернет-адрес страницы, на которую пытался попасть посетитель. В изначальной реализации возвращает значение атрибута redirectfield\_name;

□ raise exception — атрибут. Если его значение равно True, то при попытке попасть на страницу гость или пользователь с недостаточными правами получит сообщение об ошибке 403. Если значение параметра равно False, то посетитель будет перенаправлен на страницу входа. Значение по умолчанию — False;

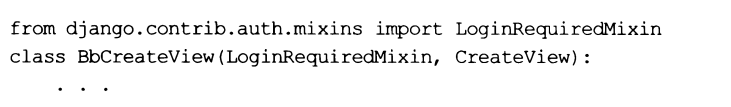
□ has\_no\_permission(self) — метод, вызывается в том случае, если текущий пользователь не выполнил вход или не имеет необходимых прав, и на это нужно как-то отреагировать.

В изначальной реализации, если значение атрибута raise exception равно True, возбуждает исключение permissiondenied с сообщением, возвращенным методом get\_permission\_denied\_message(). Если же Значение атрибута raise\_Exception равно False, то выполняет перенаправление по интернет-адресу, возвращенному методом get login url ().

Рассмотрим классы-примеси, производные ОТ accessmixin:

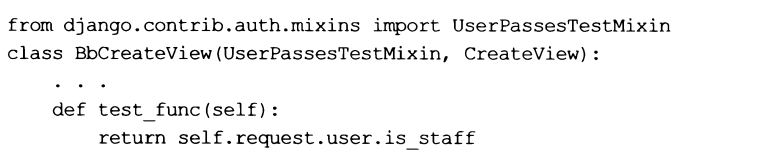
□ loginrequiredmixin— допускает к странице только пользователей, выполнивших вход.

Разрешаем добавлять новые объявления только пользователям, выполнившим вход:



□ userpassestestmixin — допускает к странице только тех пользователей, кто выполнил вход и в чьем отношении переопределенный метод test func(seif) вернет в качестве результата значение True (в изначальной реализации метод Test\_func() возбуждает исключение notimpiementederror, поэтому его обязательно следует переопределить).

Разрешаем создавать новые объявления только пользователям со статусом персонала:

****

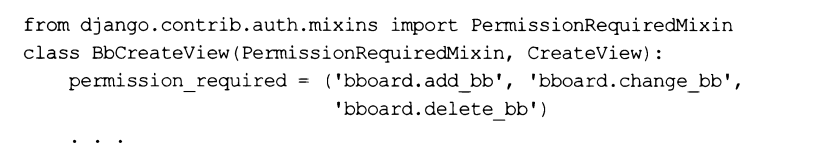
□ permissionrequiredmixin— допускает к странице только пользователей, имеющих заданные права. Класс поддерживает дополнительные атрибут и методы:

• permission required — атрибут, задает требуемые права, которые указываются в том же формате, который применяется в методах has permo и Has permso. Можно задать одно право или последовательность прав;

• get permission required(seif) — метод, должен возвращать требуемые права. В изначальной реализации возвращает значение атрибута permission\_Required;

• has permission(seif) — метод, должен возвращать True, если текущий пользователь имеет заданные права, и False — в противном случае. В изначальной реализации возвращает результат вызова метода has perms () у текущего пользователя.

Разрешаем создавать новые объявления только пользователям с правами на создание, правку и удаление записей модели вь:

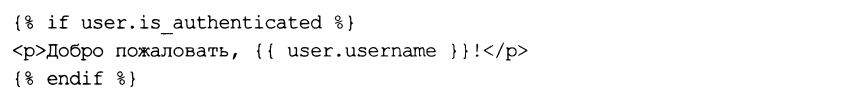


**Авторизация в шаблонах**

Если в числе активных обработчиков контекста, указанных в параметре Context\_processors настроек шаблонизатора, имеется django.contrib.auth.contextprocessors. Auth, то он будет добавлять в контекст каждого шаблона переменные user и perms, хранящие соответственно текущего пользователя и его права.

Переменная user хранит экземпляр класса user, и из него можно извлечь сведения о текущем пользователе.

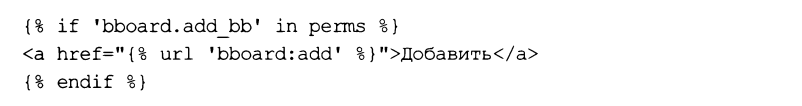
В следующем примере устанавливается, выполнил ли пользователь вход на сайт, и если выполнил, то выводится его имя:

****

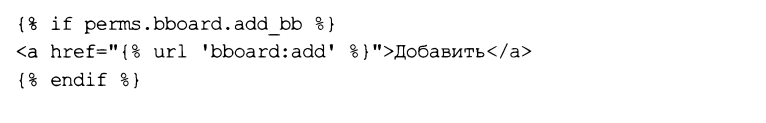
Переменная perms хранит особый объект, который можно использовать для выяснения прав пользователя, причем двумя способами:

□ первый способ — применением оператора in или not in. Права записываются в виде строки в том же формате, который применяется в вызовах методов Has\_perm() Hhas\_perms().

В следующем примере выполняется проверка, имеет ли пользователь право на добавление объявлений, и если имеет, то выводится гиперссылка на страницу добавления объявления:

****

□ второй способ— доступ К атрибуту С именем вида <приложение>. <операция>\_<модель>. Такой атрибут хранит значение True, если пользователь имеет право на выполнение заданной операции В заданной модели указанного приложения, И False — в противном случае. Пример:



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);

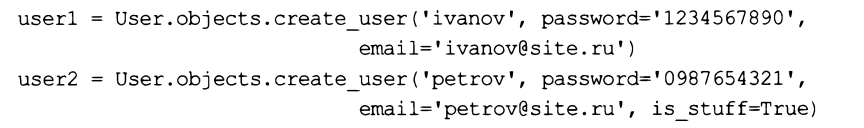
□ auth user model— имя класса модели, хранящей список зарегистрированных пользователей, в виде строки. По умолчанию: "auth.user" (стандартная модель User).

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:

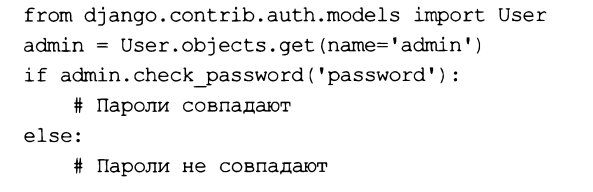
****

□ 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**

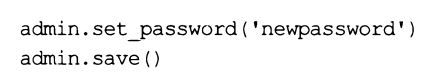
Еще четыре метода, поддерживаемые моделью user, предназначены для работы с паролями:

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>) .

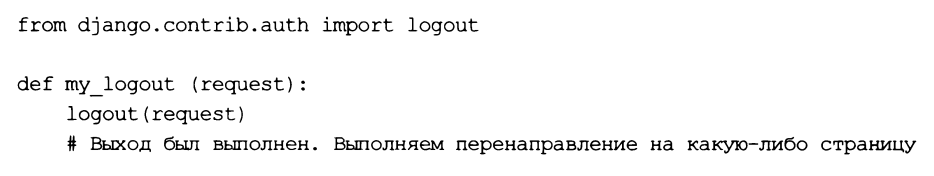
Запрос должен быть представлен экземпляром класса httprequest, а пользователь.

От имени которого выполняется вход, — записью модели user.

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

****

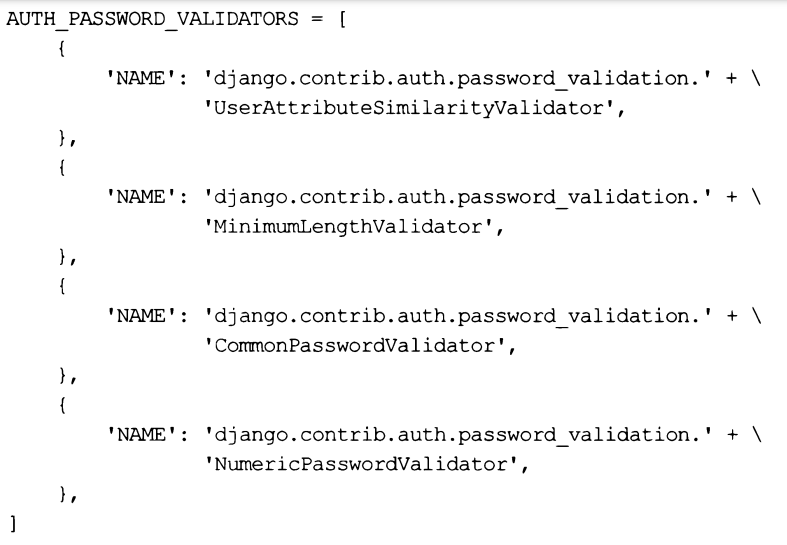
Выход с сайта выполняется вызовом функции logout(<запрос>). Запрос должен быть представлен экземпляром класса httprequest. Пример:



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:

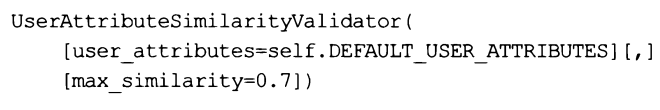


Это список, включающий все (четыре) стандартные валидаторы, поставляемые в составе Django.

**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:

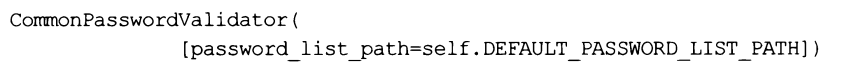
****

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);

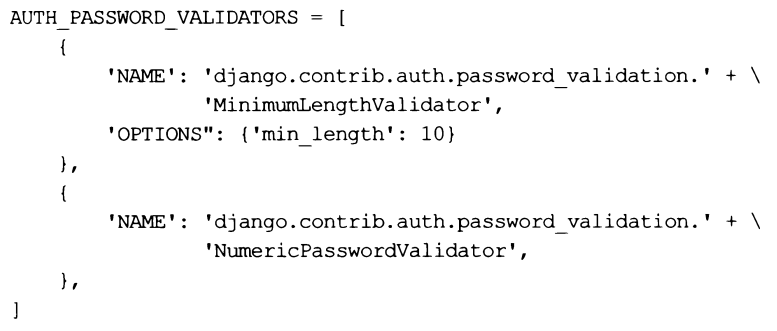
□ commonpasswordvalidator () — проверяет, не входит ли пароль в указанный перечень наиболее часто встречающихся паролей. Формат вызова конструктора:

****

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**

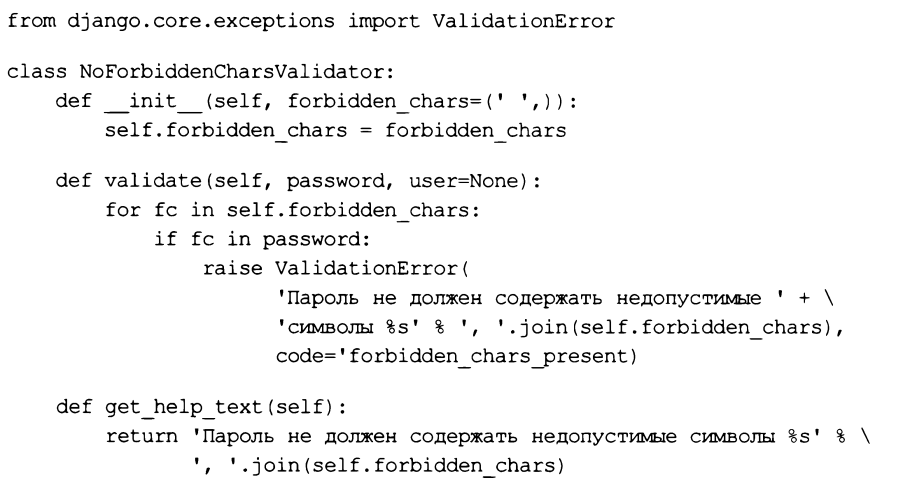
Валидаторы паролей обязательно должны быть реализованы в виде классов, поддерживающих два метода:

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

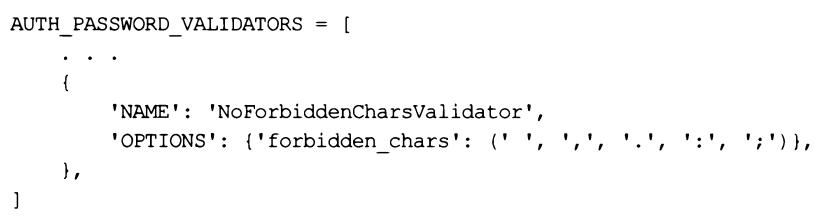
Метод не должен возвращать значения. Если пароль не проходит валидацию, то следует возбудить исключение validationerror ИЗ модуля django. Core.exceptions;

□ 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;

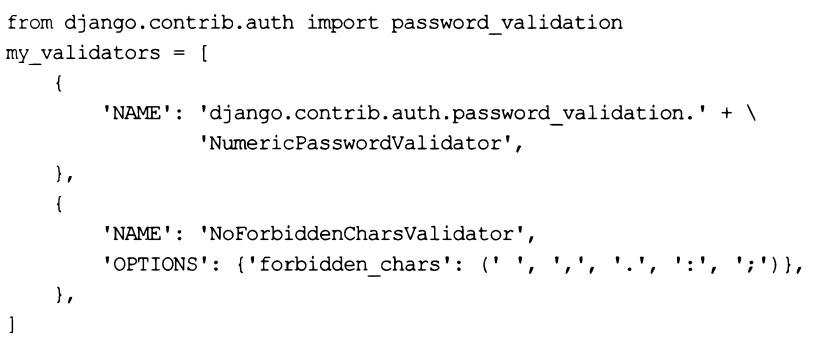
□ password\_changed (<пароль>[, user=None] [, password\_validators=None]) —сообщает всем валидаторам, что пароль пользователя изменился.

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!

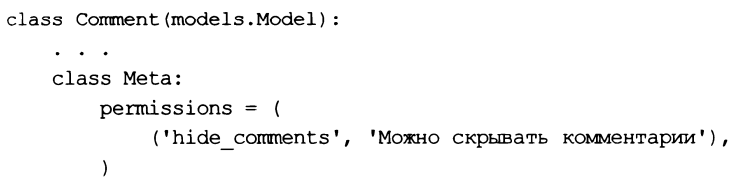
Полное руководство по Python Social Auth располагается здесь:

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

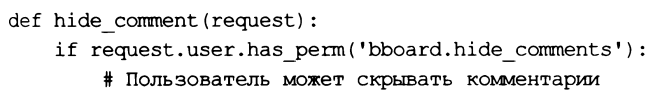
1. Создание своих прав пользователя

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.

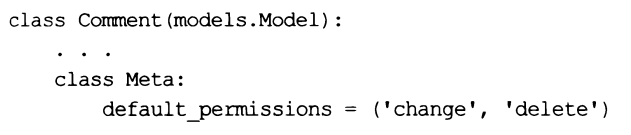
Для любой модели можно создать дополнительный набор произвольных прав. Для этого мы воспользуемся параметром permissions, задаваемым для самой модели — во вложенном классе Meta. В качестве его значения указывается список или кортеж, каждый элемент которого описывает одно право и также представляет собой 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.