**Глава 4(Создание представления входа в систему)**

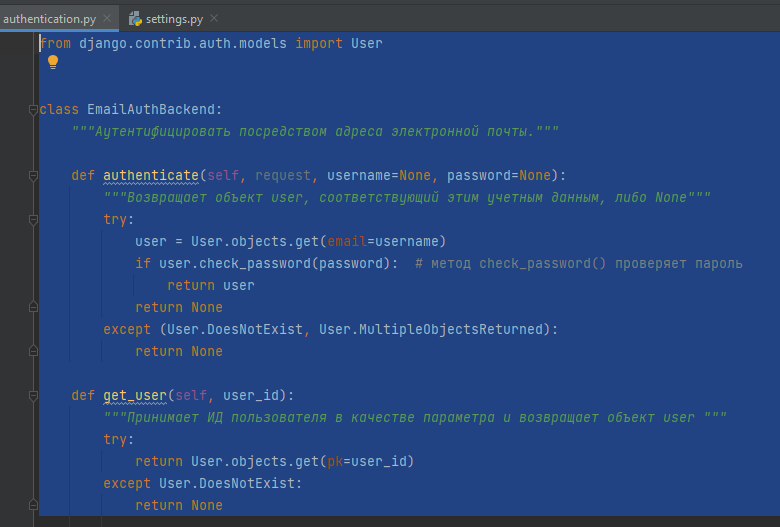
**authenticate()** = **Аутентифицирует** пользователя по базе данных. Указанный метод принимает объект request, параметры username и password и возвращает объект User, если пользователь был успешно аутентифицирован, либо **None** в противном случае.

**is\_active =** Проверяет активен ли пользователь

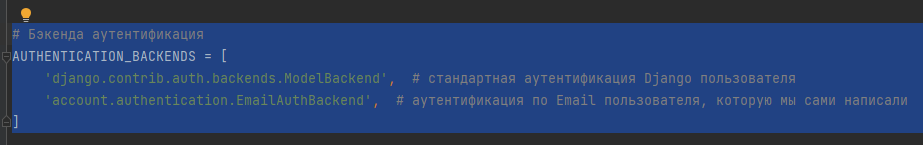
**login()** = Задает пользователя в текущем сеансе для входа пользователя в сайт. ВХОД выполняется вызовом функции **login**. Запрос должен быть представлен экземпляром класса HttpRequest, а пользователь, от имени которого выполняется вход, — записью модели User.

**Глава 4(Разработка конкретно-прикладного бэкенда аутентификации)**

Мы создали **дополнительный** бэкенд **аутентификацию**, для этого мы создали файл **authentication.py** и в нем прописали класс аутентификации



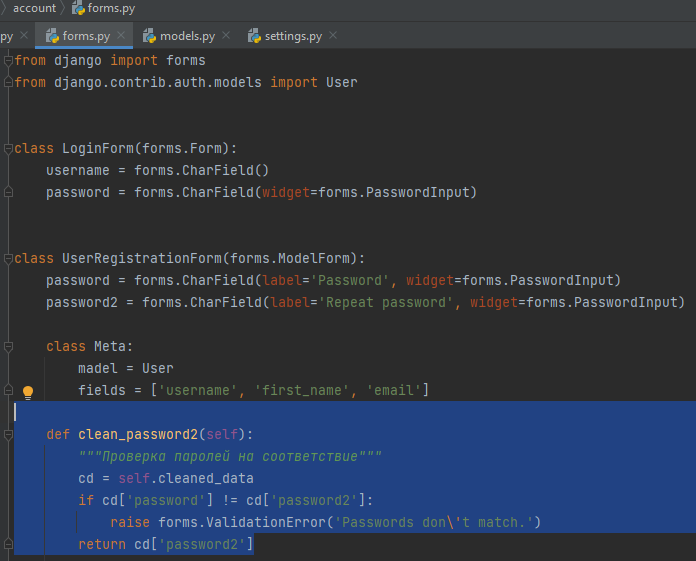
После в настройках проекта добавили собственную аутентификацию после встроенного, и теперь пользователь может аутентифицироваться как по **имени**, так и по **email**



**Глава 4(**Регистрация пользователя**)**

В форме если нужно проверить валидацию отдельного поля можно использовать метод с началом **clean\_<название поля>**. Такие методы исполняются, когда форма проходит валидацию при вызове метода is\_valid(). А если хотите проверить/валидировать всю форму можно использовать метод **clean()**

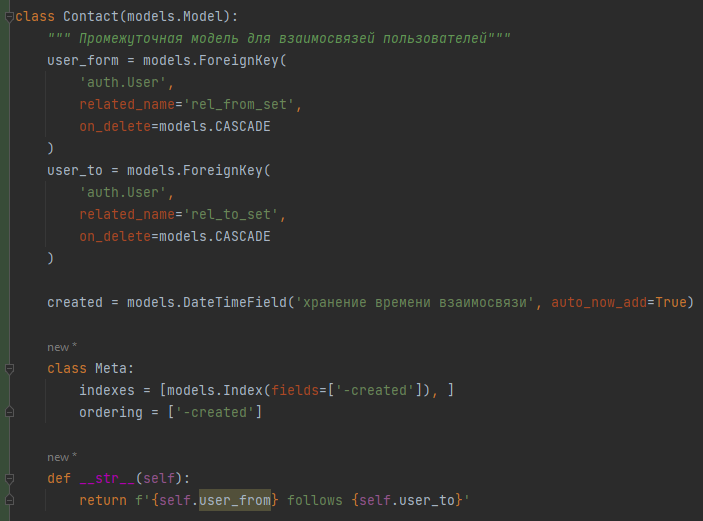
Пример:



**Глава 7(Формирование взаимосвязей многие-ко-многим с промежуточной моделью)**

Создание промежуточной модели необходимо, когда требуется хранить дополнительную информацию о взаимосвязи, например дату создания взаимосвязи или поле, описывающее природу взаимосвязи.

Фото промежуточной модели:



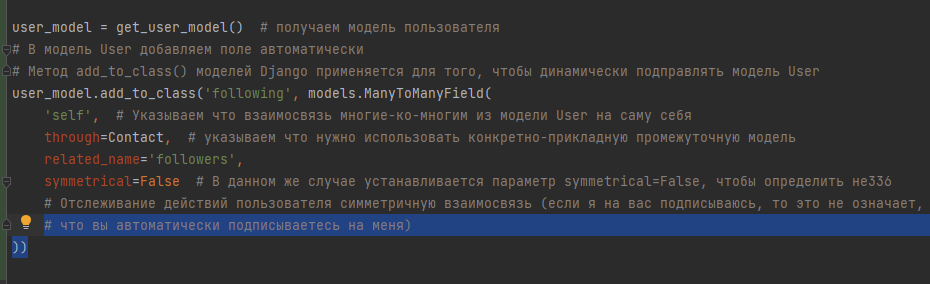
• user\_from: внешний ключ (ForeignKey) для пользователя, который создает взаимосвязь;

• user\_to: внешний ключ (ForeignKey) для пользователя, на которого есть подписка;

• created: поле DateTimeField с параметром auto\_now\_add=True для хранения времени создания взаимосвязи.

Мы в модель пользователей (User) добавили поле **динамическим** способом.

Фото примера:



**get\_user\_model()** = Извлекаем модель пользователя

**add\_to\_class()** = моделей Django применяется для того, чтобы динамически подправлять модель User

**'self'** = Указываем что взаимосвязь многие-ко-многим из модели User на саму себя

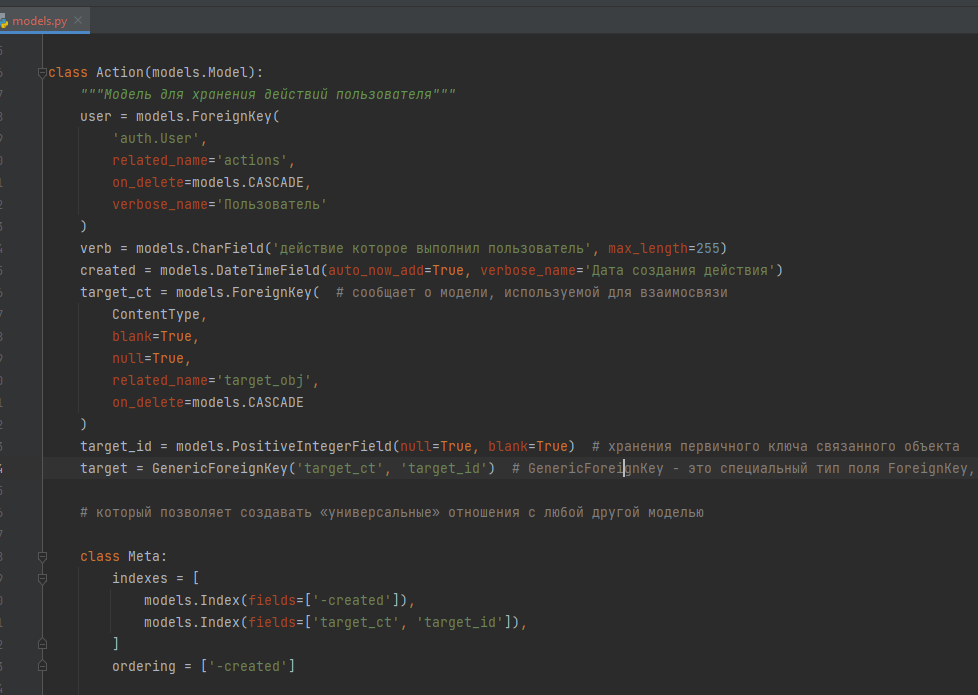
**through=Contact** = указываем что нужно использовать конкретно-прикладную промежуточную модель

**symmetrical=False** = При определении поля ManyToManyField в модели, создавая взаимосвязь с самой моделью, Django навязывает взаимосвязи симметричность. В данном же случае устанавливается параметр **symmetrical=False**, чтобы определить **несимметричную взаимосвязь** (если я на вас подписываюсь, то это не означает, что вы автоматически подписываетесь на меня).

**Глава 7 (Применение фреймворка contenttypes)**

**Contenttypes =** отслеживать все установленные в проекте модели и предоставляет типовой интерфейс взаимодействия с этими моделями. Если нужно возможность подключатся к разным моделям, используй contenttypes

Фото примера:



**target\_ct** = поле ForeignKey, указывающее на модель **ContentType**;

**target\_id** = **PositiveIntegerField** для хранения первичного ключа связанного объекта;

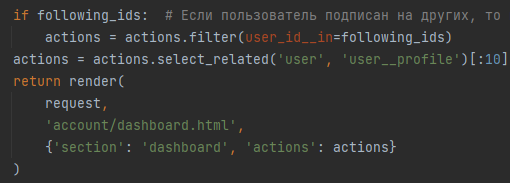
**target** = поле **GenericForeignKey** для связанного объекта на основе комбинации двух предыдущих полей.

**Глава 7 (Оптимизация наборов запросов, предусматривающих связанные объекты)**

**Применение метода select\_related.**

Используется для оптимизации работы с базой данных при работе с внешними ключами **один-ко-многим**. Он возвращает QuerySet, который будет «следовать» за отношениями внешнего ключа, выбирая дополнительные данные связанных объектов при выполнении своего запроса. Метод select\_related предназначен для полей **ForeignKey** и **OneToOne**.

Фото:



В этом примере **select\_related** используется для оптимизации запроса к базе данных. Он указывает Django выбрать связанные данные для полей **user** и **user\_\_profile** при выполнении запроса к базе данных. Это означает, что при итерации по объектам actions, доступ к связанным данным пользователя и профиля пользователя не потребует дополнительных запросов к базе данных.

Кроме того, в этом примере используется срез [:10], чтобы ограничить количество возвращаемых объектов до 10. Это означает, что запрос вернет только первые 10 объектов из результирующего набора.

**user\_\_profile** - это обозначение связи между двумя моделями в Django. Двойное подчеркивание \_\_ используется для указания связи между моделями. В данном случае user\_\_profile означает, что мы хотим получить доступ к связанному объекту profile, который связан с объектом user. Эта связь определяется в модели User, где указывается, что у каждого пользователя есть профиль

**Применение метода prefetch\_related()**

**prefetch\_related** - это метод, который используется для оптимизации запросов к базе данных в Django. Он работает аналогично методу **select\_related**, но используется для связей “**многие-ко-многим**” и “ **многие-к-одному** ”. **prefetch\_related** выполняет отдельный запрос для получения связанных объектов и затем “склеивает” их с основными объектами в Python. Это позволяет уменьшить количество запросов к базе данных при обращении к связанным объектам.

Фото:

