skypro

Основы аутентификации. Шпаргалка

Аутентификация — это процесс проверки подлинности пользователя. В контексте веб-приложений аутентификация означает проверку данных, предоставленных пользователем (например, имени пользователя и пароля), чтобы подтвердить его личность. Этот процесс позволяет обеспечить системе взаимодействие с доверенным пользователем.

Модель пользователя

Модель пользователя в Django — представляет собой структуру данных, которая определяет, какие данные будут храниться о пользователе в базе данных.

Обычно логика работы с кастомными пользователями выносится в отдельное приложение. Это помогает структурировать проект и облегчает управление кодом. Примеры названий таких приложений: `accounts`, `users`, `authentication`.

Для создания кастомной модели пользователя необходимо:

1. Создать новую модель, наследующую от `AbstractUser` или `AbstractBaseUser`.
2. Определить дополнительные поля.
3. Настроить Django для использования этой модели.

AbstractUser

Класс `AbstractUser` — предоставляет готовую модель пользователя с полями и методами, которые включают большинство функциональностей, необходимых для управления пользователями. Это удобный выбор, если вам нужно добавить только несколько дополнительных полей или изменить поведение стандартной модели пользователя.

Преимущества:

- Включает все основные поля и методы, такие как `username`, `email`, `first_name`, `last_name`, `is_staff`, `is_active` и другие.
- Удобен для быстрого создания кастомной модели пользователя с минимальными изменениями.

Пример использования:

```
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    phone_number = models.CharField(max_length=15, blank=True, null=True)

    def __str__(self):
        return self.email
```

AbstractBaseUser

> **Класс** `AbstractBaseUser` — предоставляет минимальную базовую модель пользователя только с основными полями и методами, такими как `password` и `last_login`.
> Этот класс предназначен для более гибкой кастомизации, когда стандартная модель пользователя слишком ограничена для ваших нужд.

Преимущества:

- Полная свобода в определении полей и методов модели пользователя.
- Полезен для создания полностью кастомной модели пользователя с уникальными требованиями.

Недостатки:

- Требует больше работы по реализации полей и методов, которые предоставляет `AbstractUser`.
- Необходимо вручную определить менеджер модели, например `UserManager`, чтобы правильно управлять созданием и управлением пользователями.

> **Менеджер модели** — это интерфейс для взаимодействия с базой данных. Менеджеры позволяют определять методы для создания пользователей и суперпользователей, а также для управления данными модели.

Пример использования:

```
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
from django.db import models

class CustomUserManager(BaseUserManager):
    def create_user(self, email, password=None, **extra_fields):
        if not email:
            raise ValueError('The Email field must be set')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')

        return self.create_user(email, password, **extra_fields)

class CustomUser(AbstractBaseUser):
    email = models.EmailField(unique=True)
    first_name = models.CharField(max_length=30, blank=True)
    last_name = models.CharField(max_length=30, blank=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    def __str__(self):
        return self.email
```

Какую модель для наследования выбирать

- Используйте `AbstractUser`, если вам нужно добавить только несколько дополнительных полей или изменить поведение стандартной модели пользователя. Это позволяет быстро создать кастомную модель с минимальными изменениями.
- Используйте `AbstractBaseUser`, если вам нужно создать полностью кастомную модель пользователя с уникальными требованиями. Это требует больше работы, но предоставляет полную свободу в определении полей и методов модели.

Основные поля для регистрации и авторизации

Основные поля модели пользователя и их назначение

1. `username` — уникальный идентификатор для каждого пользователя.
2. `password` — пароль, который хранится в зашифрованном виде.
3. `email` — электронная почта, которая может использоваться как альтернативный идентификатор для входа. Это поле также важно для восстановления пароля и уведомлений.
4. `first_name` — имя пользователя.
5. `last_name` — фамилия пользователя.
6. `is_staff` — булевое поле, указывающее, является ли пользователь сотрудником, который может войти в административную часть сайта.
7. `is_active` — булевое поле, указывающее, является ли аккаунт активным. Неактивные пользователи не могут войти в систему.
8. `is_superuser` — булевое поле, указывающее, является ли пользователь суперпользователем, который имеет все права без ограничений.
9. `last_login` — дата и время последнего входа пользователя в систему. Используется для отслеживания активности.
10. `date_joined` — дата и время регистрации пользователя. Может использоваться для аналитики и статистики.

Настройки USERNAME_FIELD и REQUIRED_FIELDS

1. USERNAME_FIELD

> **Настройка** `USERNAME_FIELD` — определяет, какое поле будет использоваться в качестве уникального идентификатора для аутентификации пользователя. Используется, когда вы хотите изменить стандартное поле для входа, например с `username` на `email`.

Пример использования:

```
USERNAME_FIELD = 'email'
```

2. REQUIRED_FIELDS

> **Настройка** `REQUIRED_FIELDS` — определяет дополнительные обязательные поля, которые должны быть указаны при создании суперпользователя через команду `createsuperuser`. Используется для указания полей, которые должны быть заполнены при создании суперпользователя, помимо поля, указанного в `USERNAME_FIELD`.

Примеры использования:

- В данном случае при создании суперпользователя через команду `createsuperuser`, кроме `USERNAME_FIELD`, не потребуется заполнять никакие дополнительные поля. Только `username`, `email` и пароль:

  ```
  REQUIRED_FIELDS = []
  ```

- В этом случае при создании суперпользователя через команду `createsuperuser`, кроме обязательных полей, нужно будет обязательно указать значения для полей `first_name` и `last_name`:

  ```
  REQUIRED_FIELDS = ['first_name', 'last_name']
  ```

Пример реализации кастомной модели пользователя

```
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    email = models.EmailField(unique=True)
    phone_number = models.CharField(max_length=15, blank=True, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username', ]

    def __str__(self):
        return self.email
```

Настройка проекта для работы с новой моделью пользователя

Важно!

> Если вы уже применяли миграции в вашем проекте (выполняли команду `python manage.py migrate`), то до создания кастомной модели пользователя необходимо обязательно откатить миграции стандартного приложения `auth`, отвечающего за аутентификацию. Когда вы применяете миграции, Django автоматически применяет миграции стандартного приложения `auth`.

Для отката миграций приложения `auth` воспользуйтесь следующей командой:

```
python manage.py migrate auth zero
```

> Если миграции не откатить, ваше приложение не сможет подключиться к базе данных из-за конфликта в структуре таблиц!

Изменение настроек проекта

1. Добавьте кастомную модель пользователя в приложение, например `users`.
2. Зарегистрируйте приложение `users` в `INSTALLED_APPS` в файле `settings.py`.
3. В файле `settings.py` укажите вашу модель пользователя, добавив следующую строку:

```
AUTH_USER_MODEL = 'users.CustomUser'
```

Где:

- `users` — имя приложения, где хранится ваша кастомная модель пользователя.
- `CustomUser` — имя вашей кастомной модели пользователя.

Применение миграций

1. Сделайте миграции приложения `users`.
2. Примените миграции базы данных, для этого выполните команду:

```
python manage.py migrate
```

Пароли

Пароли — это ключевой элемент аутентификации, обеспечивающий безопасность аккаунтов пользователей.

Генерация паролей

Для создания нового пользователя или изменения пароля используется метод `set_password`, который автоматически хеширует пароль:

```
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    username = None
    email = models.EmailField(unique=True)
    phone_number = models.CharField(max_length=15, blank=True, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username', ]

    def __str__(self):
        return self.email

# Пример использования метода set_password
user = CustomUser(email='user@example.com')
user.set_password('plain_text_password')
user.save()
```

Обработка паролей в Django

Django использует модуль `django.contrib.auth.hashers` для хеширования и проверки паролей. По умолчанию Django использует алгоритм PBKDF2, который считается безопасным и устойчивым к атакам.

PBKDF2 (Password-Based Key Derivation Function 2) — это алгоритм хеширования паролей, который преобразует пароль в криптографический ключ. Он был разработан для защиты паролей, хранения ключей и других секретных данных.