Extending the User Model

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org

Have a Question?

Software University

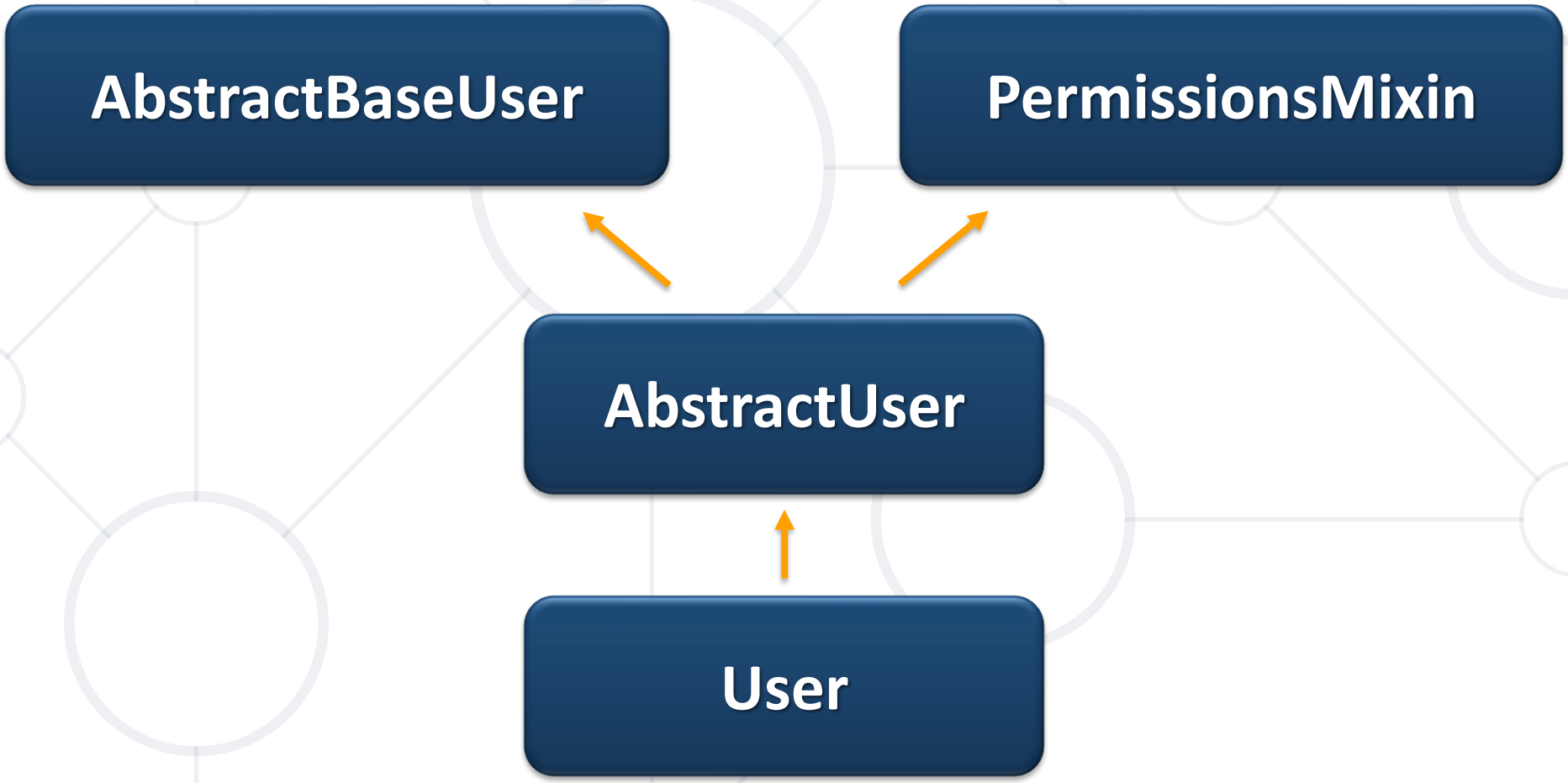
sli.do

#python-web

Table of Contents

User Model Inheritance Chain

Built-in Class User Inheritance Chain



Class User

```python
class User(AbstractUser):
    """
    Users within the Django authentication system are represented by this
    model.

    Username and password are required. Other fields are optional.
    """

    class Meta(AbstractUser.Meta):
        swappable = "AUTH_USER_MODEL"
```

Inherits from AbstractUser

Does not define any fields or methods

Class AbstractUser

```python
class AbstractUser(AbstractBaseUser, PermissionsMixin):
    """
    An abstract base class implementing a fully featured User model with
    admin-compliant permissions.

    Username and password are required. Other fields are optional.
    """

    username_validator = UnicodeUsernameValidator()

    username = models.CharField(
        _("username"),
        max_length=150,
        unique=True,
        help_text=_(
            "Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only."
        ),
        validators=[username_validator],
        error_messages={
            "unique": _("A user with that username already exists."),
        },
    )
    first_name = models.CharField(_("first name"), max_length=150, blank=True)
    last_name = models.CharField(_("last name"), max_length=150, blank=True)
    email = models.EmailField(_("email address"), blank=True)
```

Defines the username field and its validators

Defines also the first_name, last_name, and email fields

Class AbstractBaseUser

```python
class AbstractBaseUser(models.Model):
    password = models.CharField(_("password"), max_length=128)
    last_login = models.DateTimeField(_("last login"), blank=True, null=True)

    is_active = True

    REQUIRED_FIELDS = []

    # Stores the raw password if set_password() is called so that it can
    # be passed to password_changed() after the model is saved.
    _password = None

    class Meta:...

    def __str__(self):...

    def save(self, *args, **kwargs):...

    def get_username(self):...

    def clean(self):...
```

Defines the password and last_login fields

Sets the REQUIRED_FIELDS to an empty list

Defines important methods

PermissionsMixin

```python
class PermissionsMixin(models.Model):
    """
    Add the fields and methods necessary to support the Group and Permission
    models using the ModelBackend.
    """

    is_superuser = models.BooleanField(
        _("superuser status"),
        default=False,
        help_text=_(
            "Designates that this user has all permissions without "
            "explicitly assigning them."
        ),
    )
    groups = models.ManyToManyField(
        Group,
        verbose_name=_("groups"),
        blank=True,
        help_text=_(
            "The groups this user belongs to. A user will get all permissions "
            "granted to each of their groups."
        ),
        related_name="user_set",
        related_query_name="user",
    )
    user_permissions = models.ManyToManyField(
        Permission,
```

Provides a set of methods and fields that are useful for handling permissions and user roles

Extending the Django User

Extending the User Model

- **Extending** the Django User model is a **common practice** in many Django projects
- There are **several reasons** why you might want to do so
  - **Custom** User **Fields**
  - **Custom Methods** and **Properties**
  - **Consistency** Across the Project
  - **Scalability** and Future **Changes**

Extending the User Model

- Use **AUTH_USER_MODEL** in your project **settings** to **specify** your **custom User model**
- This should be done **before** running **`makemigrations`** for the first time
- This approach **ensures** that **all references** to the **User model** within Django and third-party apps **point** to your **extended model**

Ways to Extend the User Model

- Model **inheritance** **without** creating a **new table** (**Proxy Model**)
- A new model that has its **own table** and a **One-To-One relationship** with the **existing** User Model
- Creating a **new** **user** model that **inherits** from the **AbstractUser**
- Creating a **custom** **user** extending the **AbstractBaseUser**

Using a Proxy Model

- The **Proxy Model** is used to **change** the **behavior** of an **existing** model **without affecting** the **existing database** schema
  - e.g., add extra methods, default ordering, etc.

```
from django.contrib.auth.models import User

class AppUserProxy(User):
    class Meta:
        proxy = True
        ordering = ('first_name', )

    def some_custom_behavior(self):...
```

Using a Proxy Model

- The **AppUserProxy** model will **share** the **same** **database table** (**`auth_user`**) with the **default Django User** model
- It can be **useful** if you **only** **need** to **extend** the user model with **additional methods** or **properties** **without** **adding new fields** to the database table
- You can use this **proxy model** in your code, and it **won't** **affect** the **database structure**

Using One-to-One Relationship

- Used to **store extra information** about the **existing** User Model that is **not** **related** to the **authentication** process
- Allows you to **keep** the **default** **`auth_user`** **table** while **extending** it with **additional fields** in a **separate table**
- After creating this model, you would **need** to **run** **`makemigrations`** and **`migrate`** to **apply** the **changes** to the database

One-to-One Relationship Example

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

UserModel = get_user_model()

class Profile(models.Model):
    user = models.OneToOneField(UserModel, on_delete=models.CASCADE,
primary_key=True)
    date_of_birth = models.DateField(null=True, blank=True)
    profile_picture = models.ImageField(upload_to='profile_pics/',
null=True, blank=True)
    # Add any additional fields related to the user profile

    def __str__(self):
        return self.user.username
```

One-to-One Relationship Example

- You can **access** the **profile information** for a **user** using the **one-to-one relationship**

```
# Example usage in views

def example_view(request):
    current_user = request.user

    # Access the related profile
    current_user_profile = current_user.profile
    ...
```

Live Demo

Profile Model with One-to-One Relation

Inheriting from AbstractUser

- Inherit from the `AbstractUser` model to **add extra information** directly to the **default `User`** model
- This approach allows you to **add extra fields** to the user model **without creating** a **separate table** in the database
- You need to **update** the `AUTH_USER_MODEL` property in your project's **`settings.py`** file to **point** to **your new model**
- It **affects** the **entire database schema**, and you need to **be careful** when making such changes, especially in **existing** projects

Inheriting from AbstractUser

```
from django.contrib.auth import models as auth_models

class CustomUser(auth_models.AbstractUser):
    # Add extra fields
    date_of_birth = models.DateField(null=True, blank=True)
    profile_picture = models.ImageField(upload_to='profile_pics/',
null=True, blank=True)
    ...
```

```
# settings.py
AUTH_USER_MODEL = 'your_app_name.CustomUser'
```

Extending the AbstractBaseUser

- Extending `AbstractBaseUser` is **suitable** when:
  - You have **specific requirements** for the **authentication process**
  - You need **more control over** the **user model** compared to the built-in User model
- Remember to **update** the `AUTH_USER_MODEL` **setting** in your project's `settings.py` file **before** running **migrations**
- Consider the **impact** on the **database schema**, especially if your project **already has data**
  - It's **recommended** to **plan** such **changes carefully**

Changing the Authentication Process

- Initial steps to **change** the **authentication process**
  - Specify that the **email** shall be used **as** the **unique identifier instead** of a **username**

```
class AppUser(auth_models.AbstractBaseUser
    USERNAME_FIELD = 'email'

    email = models.EmailField(
        null=False,
        blank=False,
        unique=True,
    )
```

Changing the Authentication Process

- In Django's **authentication system**, the **USERNAME_FIELD** is a **setting** used to **define** the **field** that is a **unique identifier** for **authentication**
- By **default**, **USERNAME_FIELD** is set to '**`username`**'
- When you set it to **another field**, such as '**`email`**', it means that **users** will be **authenticated** based on **their email addresses**

Extending the BaseUserManager

```
from django.contrib.auth import models as auth_models
from django.db import models

class AppUserManager(auth_models.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)

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

A manager class for managing user creation

Includes methods for creating regular users and superusers

Extending the AbstractBaseUser

```
class AppUser(auth_models.AbstractBaseUser, auth_models.PermissionsMixin):
    email = models.EmailField(null=False, blank=False, unique=True)
    # You can add additional fields, related to user authentication
    ...

    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = AppUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

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

Includes necessary fields like email, is_active, is_staff

AppUserManager is assigned to the objects attribute

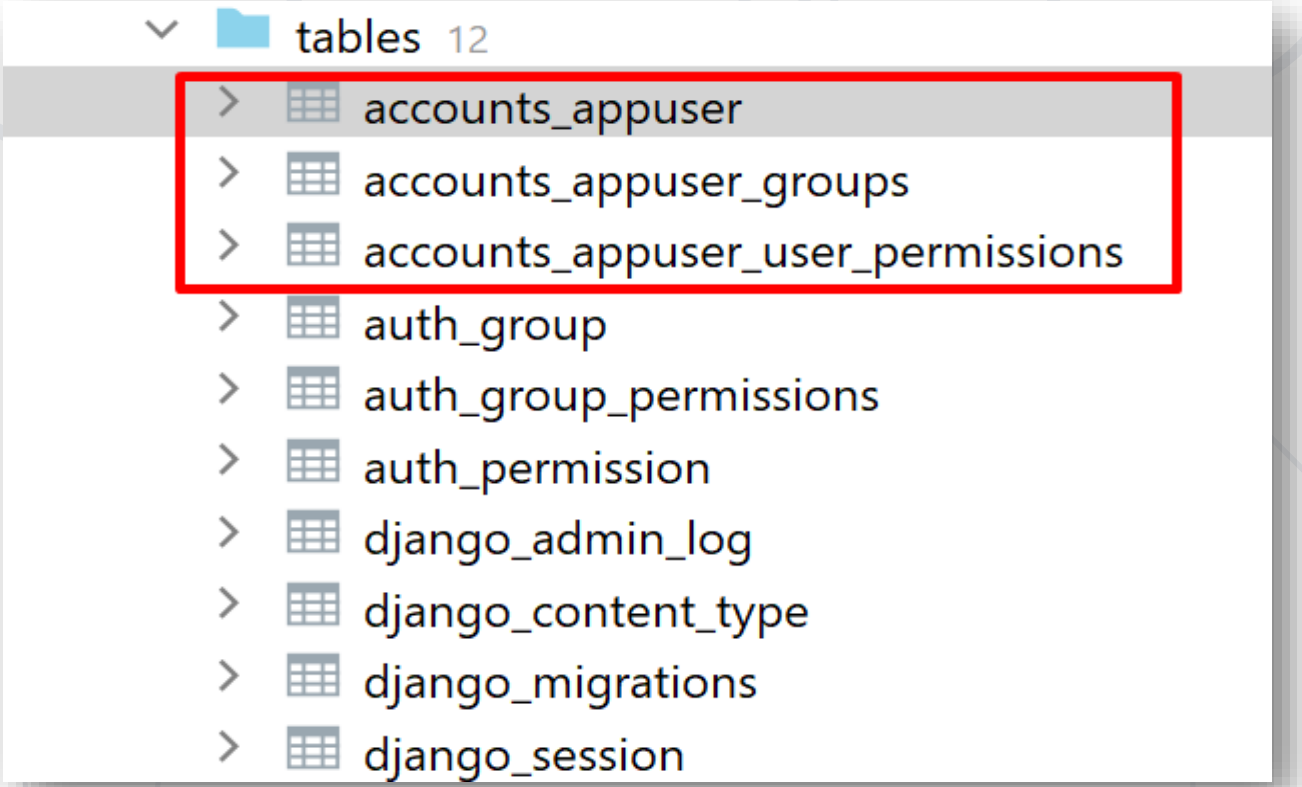
Specifies the unique identifier for the user model and required fields

Applying Changes to the Database

- Set the **AUTH_USER_MODEL** setting
- Run **makemigrations** and **migrate**
- Check the DB tables

App name prefix followed by the custom user class name

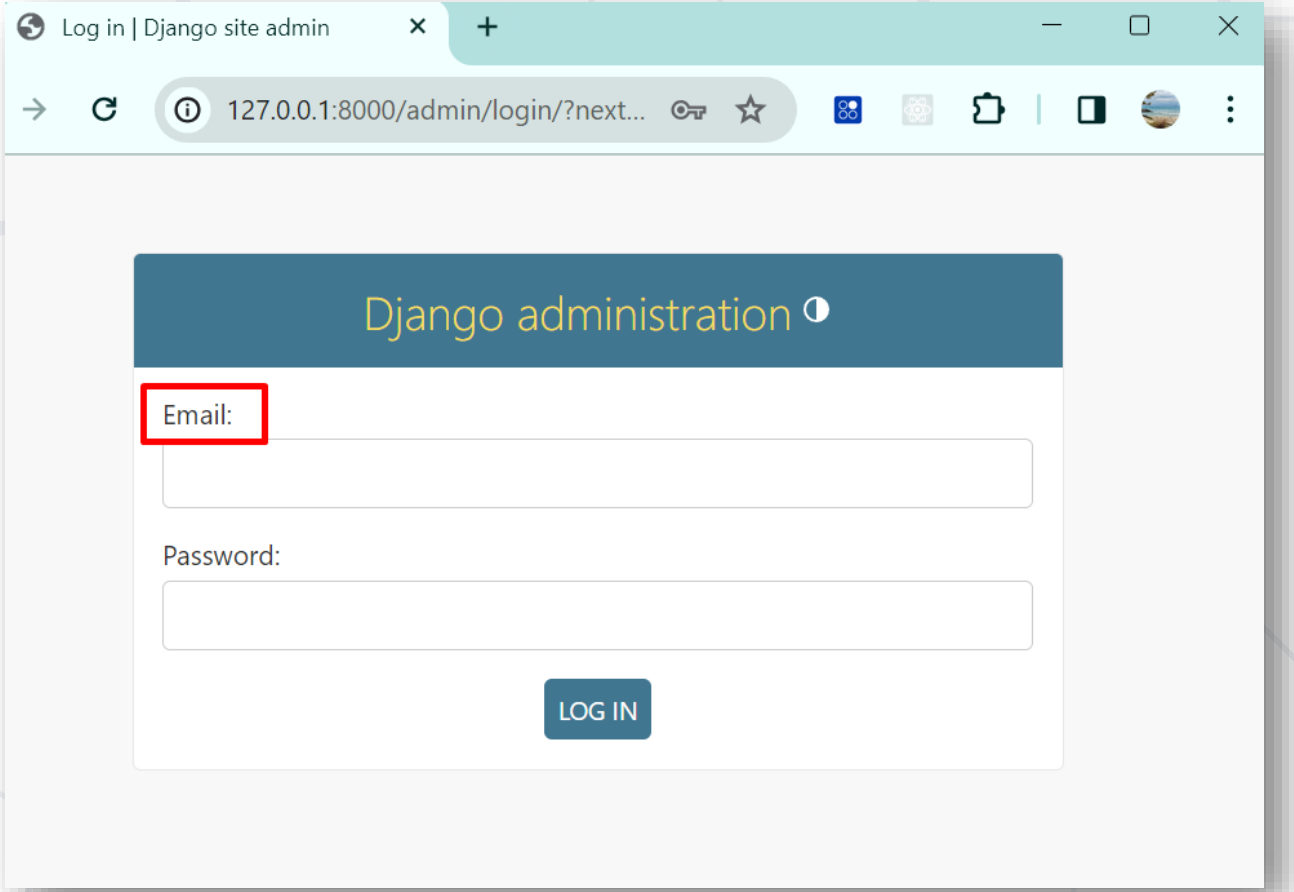
The new tables replace the built-in ones



Admin Site

- Create a **superuser**
- Start the development server
- Go to the **admin** URL

The email field now replaces the old one (username)



AppUser Forms

```
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django.contrib.auth import get_user_model

UserModel = get_user_model()

class AppUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = UserModel
        fields = ('email', )


class AppUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = UserModel
        fields = '__all__'
```

Used for creating new users

Used for updating users

Registering AppUser to the Admin Site

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth import get_user_model
from .forms import AppUserCreationForm, AppUserChangeForm

UserModel = get_user_model()

@admin.register(UserModel)
class AppUserAdmin(UserAdmin):
    model = UserModel
    add_form = AppUserCreationForm
    form = AppUserChangeForm

    list_display = ('pk', 'email', 'is_staff', 'is_superuser')
    search_fields = ('email', )
    ordering = ('pk',)

    ... # Continued on next page
```

Set the model and forms

Customize the display and behavior in the admin panel

Registering AppUser to the Admin Site

Software University

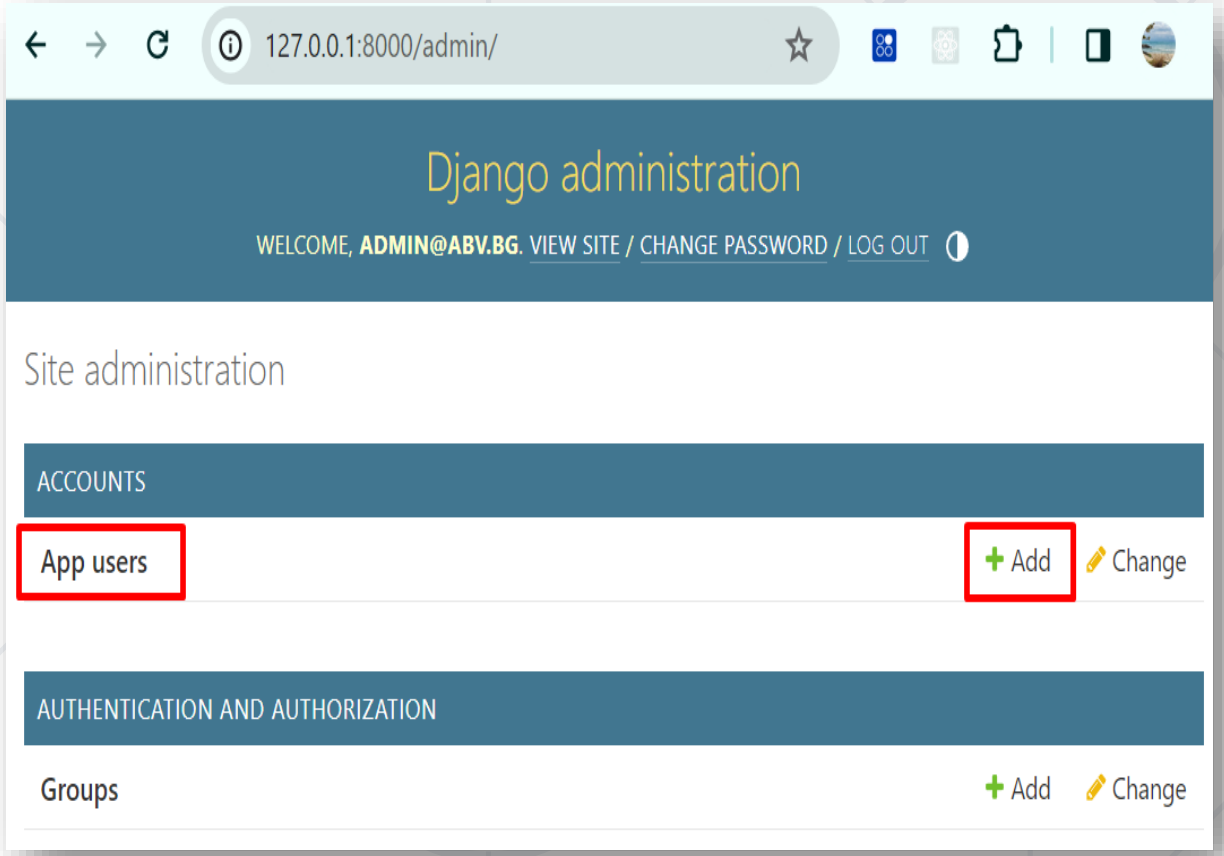
```
    ...
    fieldsets = (
        (None, {'fields': ('email', 'password')}),
        ('Personal info', {'fields': ()}),
        ('Permissions', {'fields': ('is_active', 'is_staff', 'groups',
'user_permissions')}),
        ('Important dates', {'fields': ('last_login',)}),
    )

    add_fieldsets = (
        (
            None,
            {
                "classes": ("wide",),
                "fields": ("email", "password1", "password2"),
            },
        ),
    )
```

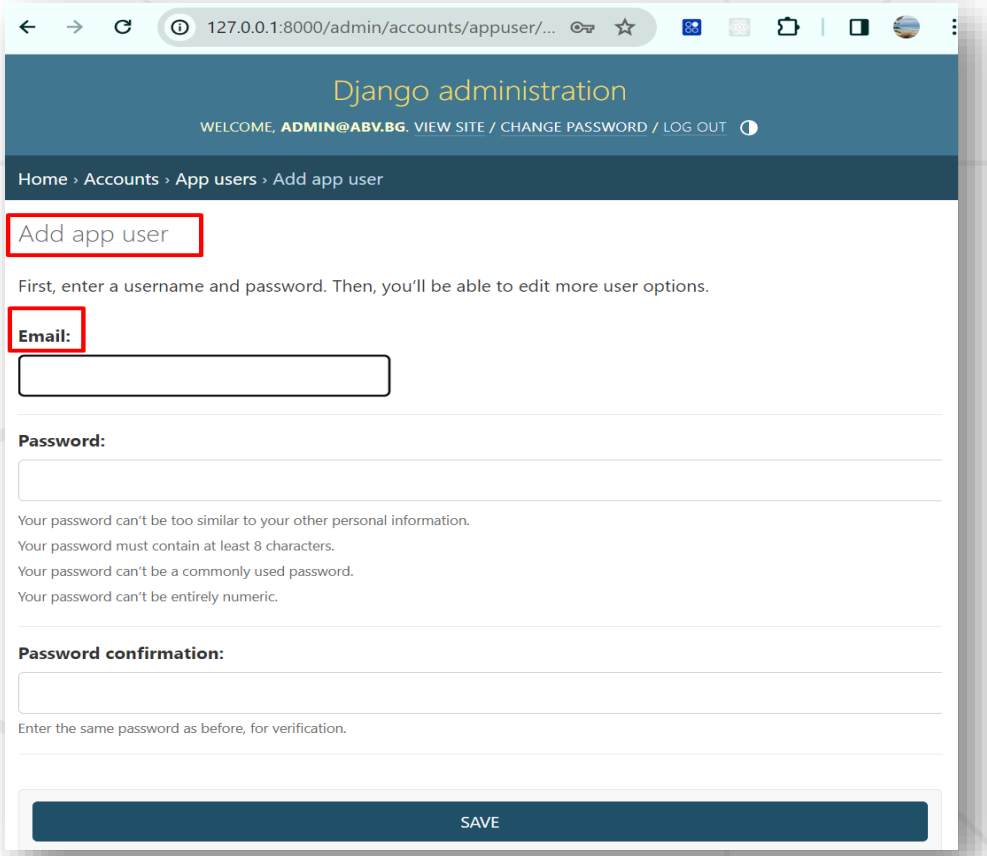
Explicitly define the fieldsets

Add personal info fields if you added some to your model

Explicitly define the add_fieldsets

Admin Site – Add New AppUsers





Live Demo

Extending the AbstractBaseUser

Django Signals

Django Signals

- Django **Signals provide** a way to **allow decoupled applications** to **get notified** when **certain events occur** in a Django application
- One **common use case** is when you extend the custom **Django User** through a **one-to-one** relationship with a **Profile** model
  - In this scenario, a **signal dispatcher** can be used to **listen** for the **custom User's `post_save` event**, **triggering actions** such as **creating** or **updating** the **associated** Profile **instance**

When to Use Signals

- Django **Signals** are particularly **useful** in scenarios where **multiple pieces** of **code** may be **interested** in the **same events**
  - This allows for a **decoupled** and **modular** design, as **different components** can **respond** to **events** without being tightly coupled
- Additionally, **signals** are **beneficial** when you **need** to **interact** with decoupled applications, such as
  - **Django core models** or models **defined** by **third-party** apps

pre_save/post_save

- Used when the business **requirements** of an application **demand** some **processing just before** or **after saving data** to the database
  - One **approach** would be to **override** the `save()` method on each model
  - A **more efficient** and **modular** way is to use **Django signals**
- The use of **senders** (usually the **model**) and **receivers** (usually the **processing function**) allows **various components** to **react** to these **signals without direct dependencies**

Django Signals - Example

```
# your_app/signals.py

from django.contrib.auth import get_user_model
from django.dispatch import receiver
from django.db.models.signals import post_save

UserModel = get_user_model()

# Define a signal receiver function

@receiver(post_save, sender=UserModel)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)
```

A signal receiver function to create a new profile when a new user is created

Import Django Signals

```
# your_app/apps.py

from django.apps import AppConfig

class AccountsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'your_project.your_app'

    def ready(self):
        import your_project.your_app.signals
```

Alternatively, you can import your signals into your_app/urls.py file

Import your signals into the ready() method

More about Django Signals: https://docs.djangoproject.com/en/5.0/topics/signals/

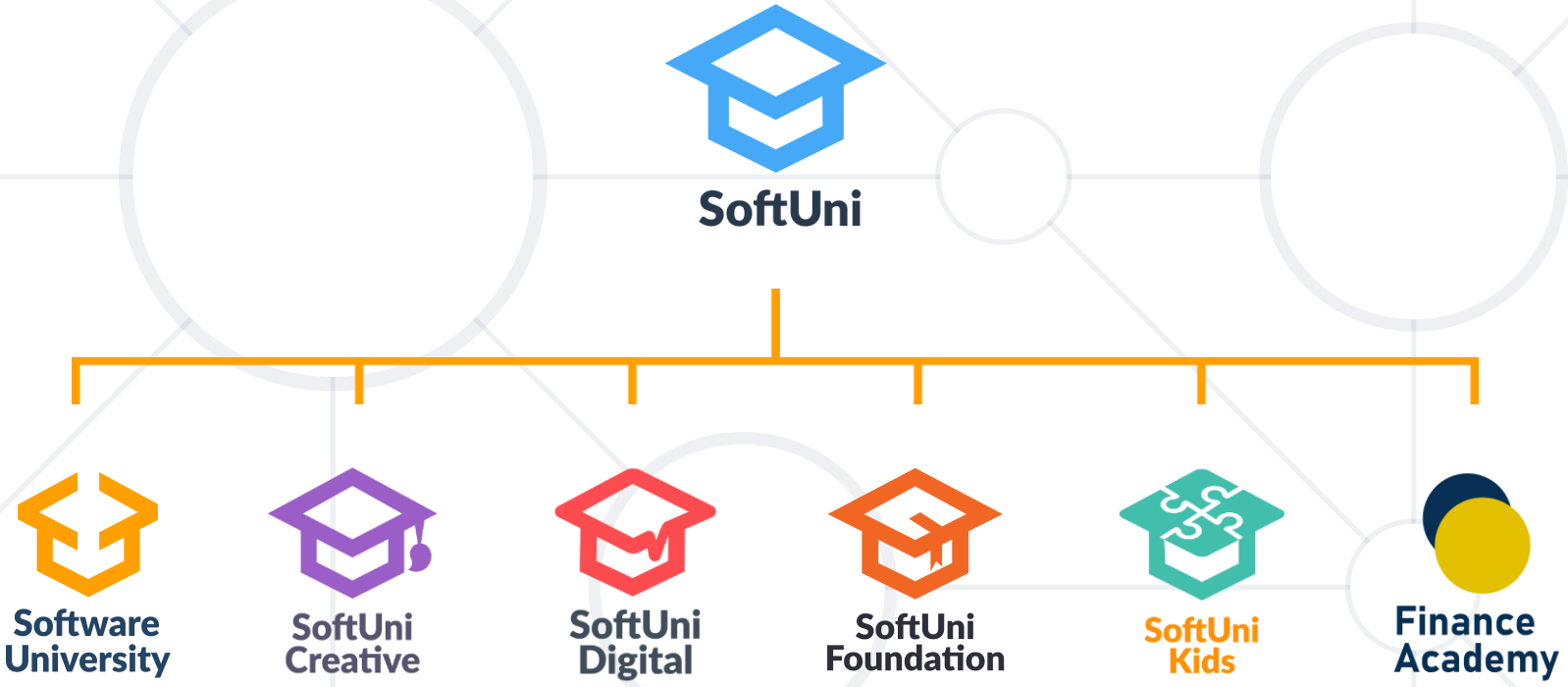
Warnings about Signals

- Django signals give the **appearance** of **loose coupling** between **components** in an application, **but**:
  - They can **lead** to **code** that is **harder** to **understand**, **adjust**, and **debug** in certain situations
    - **Debugging** can become more **complex**, especially when **multiple** **receivers** **handle** the **same** **signal**
    - **Code** **readability** can **suffer** when using signals **extensively** (**not** immediately **obvious** how different parts of the system are **interconnected**)

Summary

- User Model **Inheritance Chain**
- **Extending** the User Model
  - **Proxy** Model
  - **One-to-One** Relation
  - **AbstractUser**
  - **AbstractBaseUser**
    - Changing the authentication process

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank
Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR
Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg