User and Password Management

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org

Table of Contents

Have a Question?

Software University

sli.do

#python-web

Registration

Registration

- Django comes with a **built-in** user registration form
  - **UserCreationForm**
- It connects to the pre-built model **User**
- It comes with **three** fields
  - **username**
  - **password1**
  - **password2**

Using UserCreationForm (1)

SoftUni Foundation

- Import the **form** and create a **view** as usual

```
from django.contrib.auth.forms import UserCreationForm

def create_profile_view(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        ...
    elif request.method == 'GET':
        form = UserCreationForm()
        ...
```

Using UserCreationForm (2)

- Create a **path** and a **template** as usual
- Start the development server

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Create

Custom Registration Form (1)

SoftUni Foundation

- All User model fields could be used in a registration form

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

class RegistrationForm(UserCreationForm):
    email = models.EmailField(required=True)

    class Meta:
        model = User
        fields = ('username', 'email', 'first_name', 'last_name',)

    def save(self, commit=True):
        # clean the data and save the user
```

Custom Registration Form (2)

- The fields are added to the form

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address:

First name:

Last name:

Password:

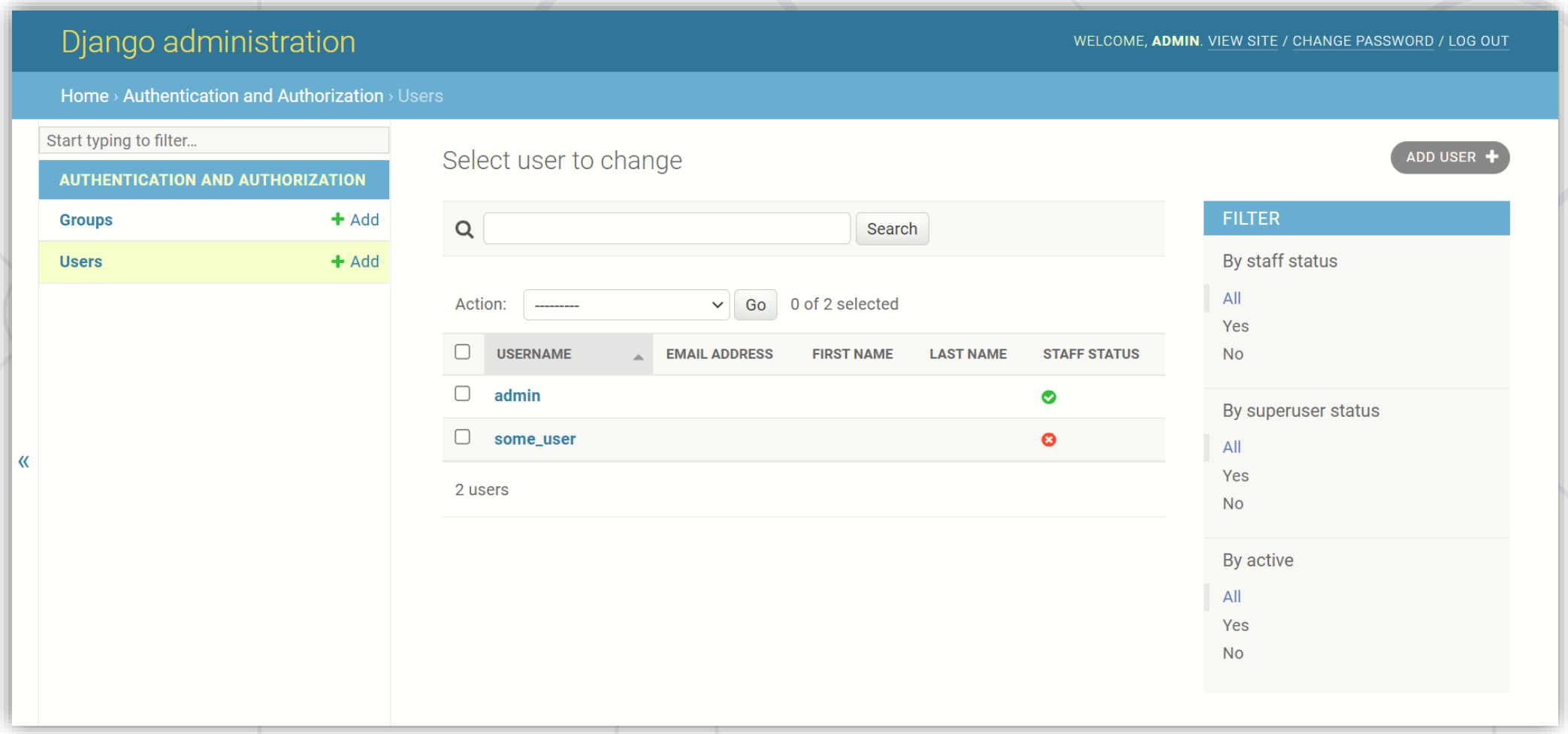
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Create

Users in the Admin Page

- View the registered users in the **admin page**



Login/Logout

Built-in Views

- Once a user is registered, we want to make sure that **they can log in and out** of the site
- Django provides class-based views that you can use:
  - **`LoginView`**
  - **`LogoutView`**
- They use the built-in **authentication forms**, but you can pass in **your own forms**
- Django provides **no default template** for the authentication views

Creating a Login System (1)

SoftUni Foundation

- To use most of the provided Django authentication system, **include the provided URLconf** in your own URLconf

```
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
]
```

- Or you can **directly use the view** in your URLconf

```
from django.contrib.auth import views

urlpatterns = [
    path('sign-in/', views.LoginView.as_view()),
]
```

Creating a Login System (2)

- If needed, you can easily **extend** or **customize the views** by subclassing them

```
from django.contrib.auth.views import LoginView

class CustomLoginView(LoginView):
    # extending or customizing the view
```

- Requests are redirected after login to **'/accounts/profile/'**
  - Change it by adding **LOGIN_REDIRECT_URL** to the settings.py

Creating a Login System (3)

- Using the LoginView the created **template** gets passed four template **context variables**
  - **form** - a Form object representing the AuthenticationForm
  - **next** - the URL to redirect to after successful login
  - **site** - the current site, according to the SITE_ID setting
  - **site_name** - an alias for site.name

Creating a Logout System

- It is like when using **`LoginView`**
- Differences:
  - Add **LOGOUT_REDIRECT_URL** to the settings.py
  - Template **context variables**
    - **title** - the string "Logged out"
    - **site** - the current Site, according to the SITE_ID setting
    - **site_name** - an alias for site.name

logout_then_login

- Logs a user out, then **redirects** to the **login page**
- **login_url** - optional argument, defaults to settings.LOGIN_URL

```
<body>
<form method="post"
      action="{% url django.contrib.auth.views.logout_then_login %}">

{% csrf_token %}
<input type="submit" value="Logout" />

</form>
</body>
```

redirect_to_login

SoftUni Foundation

- Redirects to the **login page**, and then **back to another URL** after a successful login
- Arguments
  - **next** - required; the URL to redirect to after a successful login
  - **login_url** - optional; defaults to LOGIN_URL if not supplied
  - **redirect_field_name** - optional; the name of a GET field containing the URL to redirect to after log out
    - Overrides **next** if the given GET parameter is passed

Extending the Django User

Extending the User Model

- We **often need to extend** the Django User in our applications
- Ways to extend the User model
  - Model inheritance **without** creating a **new table**
  - Having its **own table** with a **One-To-One relationship** with the existing User Model
  - Creating custom user extending the **AbstractBaseUser**
  - Creating a **new user** model that inherits from **AbstractUser**

Model Inheritance (Using Proxy Model)

SoftUni Foundation

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

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

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

    def some_behavior(self):...
```

Using One-to-One Relationship

SoftUni Foundation

- It is used to **store extra information** about the existing User Model that is **not** related to the authentication process

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

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # add new fields
```

Extending the AbstractUser

SoftUni Foundation

- Like the one-to-one relationship it is used to **add some extra information**
- However, it **directly** makes the changes in the **User model**, **without** creating an extra class
- Keep in mind that it **dramatically impacts** the **database** schema

```
class CustomUser(AbstractUser):
    # add extra fields
    # update settings.py AUTH_USER_MODEL property
```

Extending the AbstractBaseUser

SoftUni Foundation

- It is used when the app has **specific requirements in relation** to the authentication process
- Keep in mind that it **dramatically impacts** the **database** schema

```
from django.contrib.auth.models import PermissionsMixin
from django.contrib.auth.base_user import AbstractBaseUser

class CustomUser(AbstractBaseUser, PermissionsMixin):
    # create the class similar to the build-in User Model
    # pay attention to all objects related like Manager, etc.
```

Password Management

Password Management

- Django provides a **secure** and **flexible** set of tools for managing user passwords
- It should **not** be reinvented **unnecessarily**
- It uses the **PBKDF2** algorithm with a **SHA256** hash
  - Quite **secure**, requiring massive amounts of computing time to break

Hashing

- For security reasons, the passwords should be stored in a hashed form
  - This guards against unauthorized access to the database
- Hashing performs a **one-way** transformation on a password, turning it into another string, called **hashed password**
  - It is practically impossible to turn the hashed password back into the original password

Default PASSWORD_HASHERS

```
PASSWORD_HASHERS = [
 'django.contrib.auth.hashers.PBKDF2PasswordHasher',
 'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
 'django.contrib.auth.hashers.Argon2PasswordHasher',
 'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
 'django.contrib.auth.hashers.ScryptPasswordHasher',
]
```

Hashing While Testing

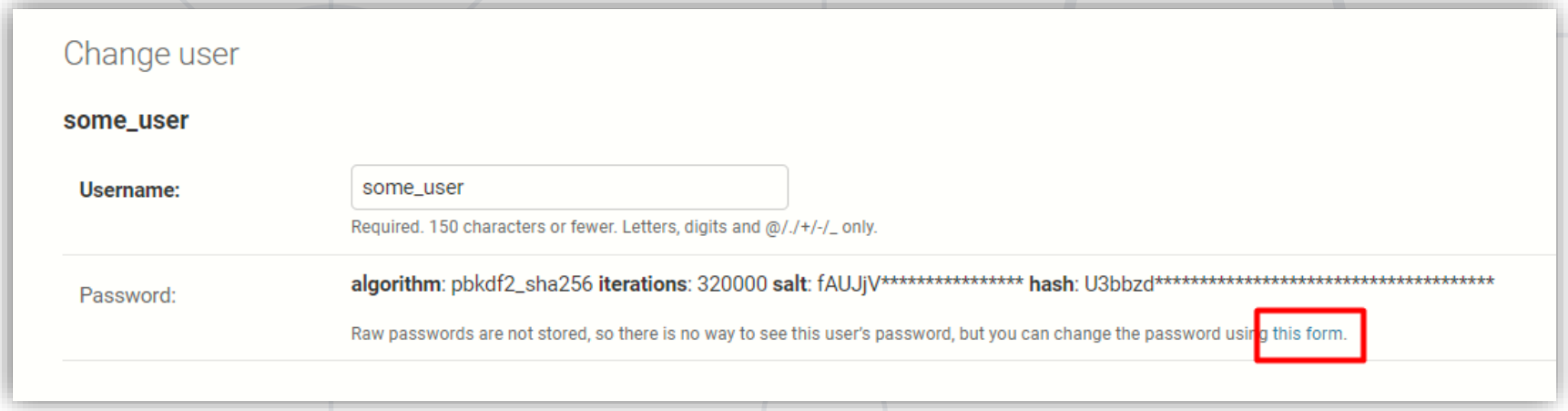
SoftUni Foundation

- The default password hasher is rather slow by design
- When authenticating **many users in tests**, you may want to use a custom settings file and set the **PASSWORD_HASHERS** setting to a faster hashing algorithm

```
PASSWORD_HASHERS = [
'django.contrib.auth.hashers.MD5PasswordHasher',
]
```

Set Password

- **set_password(raw_password)**
  - User model method
  - Sets the user's password to the given raw string, taking care of the password hashing (doesn't save the User object)
- Set password using the **admin page**



Other User Model Password Methods

- **check_password(raw_password)**
  - Returns `True` if the given raw string is the correct password for the user (takes care of the password hashing)
- **set_unusable_password()**
  - Marks the user as having **no password** set (not blank string)
- **has_usable_password()**
  - returns `False` if **set_unusable_password()** has been called

Password Views

- Django auth system comes with a list of views you can use
  - PasswordChangeView
  - PasswordChangeDoneView
  - PasswordResetView
  - PasswordResetDoneView
  - PasswordResetConfirmView
  - PasswordResetCompleteView

Documentation: https://docs.djangoproject.com/en/4.1/topics/auth/default/#django.contrib.auth.views.PasswordChangeView

Password Forms

- Django auth system comes with a list of forms you can use
  - PasswordChangeForm
  - PasswordResetForm
  - SetPasswordForm

Documentation: https://docs.djangoproject.com/en/4.1/topics/auth/default/#module-django.contrib.auth.forms

Password Validation

- Django offers pluggable password validation
- A few validators are included in Django, but you can write your own

```
AUTH_PASSWORD_VALIDATORS = [
    {'NAME': '...UserAttributeSimilarityValidator',},
    {'NAME': '...MinimumLengthValidator',},
    {'NAME': '...CommonPasswordValidator',},
    {'NAME':'...NumericPasswordValidator',},
]
```

Custom Password Validators

- Must implement two methods
  - `validate(self, password, user=None)`
    - Validate a password
    - Return `None` if the password is valid
    - Raise a `ValidationError` with an error message if the password is not valid
  - `get_help_text()`
    - Provide a **help text** to explain the **requirements** to the user

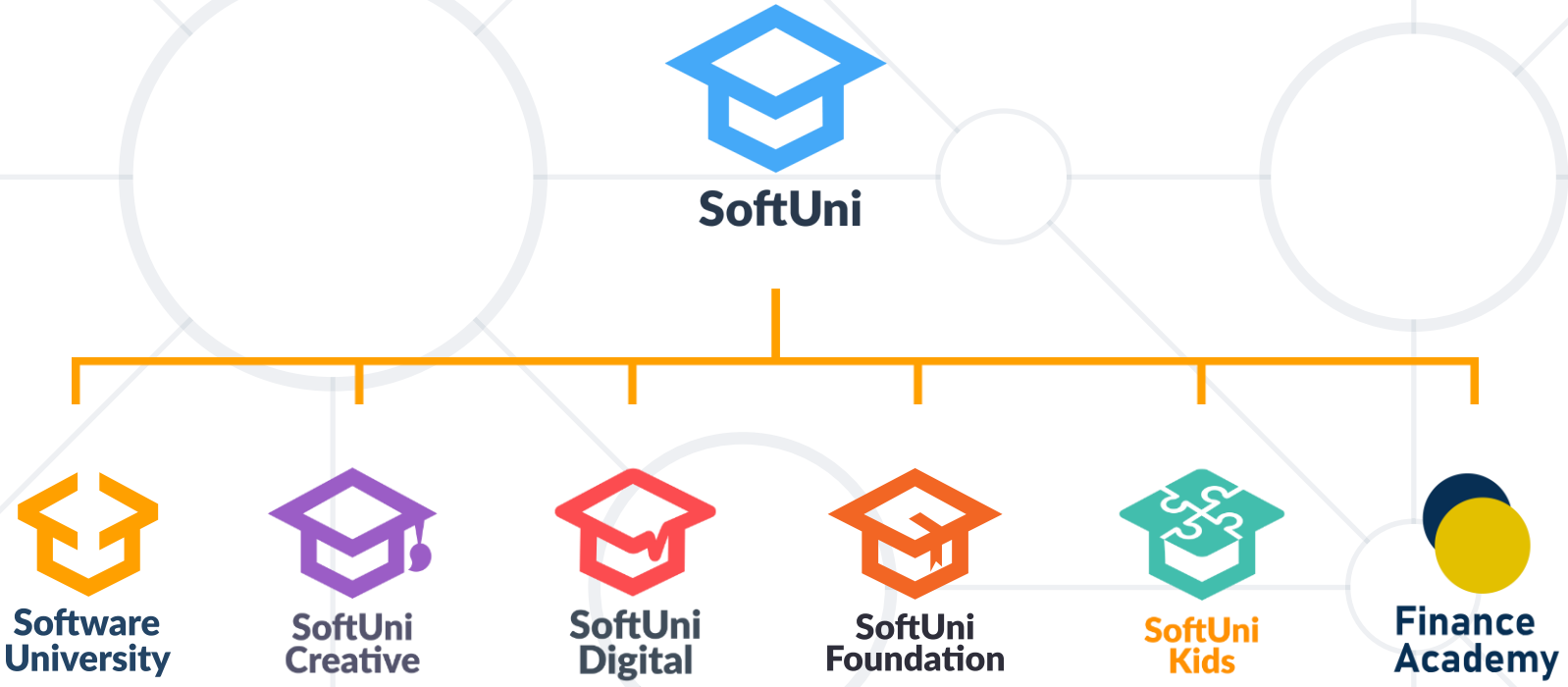
Exercise

Practicing Register, Login, Logout

Summary

- The authentication that comes with Django is good enough for **most common cases**, but you may have needs not met by the out-of-the-box defaults
- Customizing authentication in your projects **requires understanding** what points of the provided system are **extensible** or **replaceable**

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

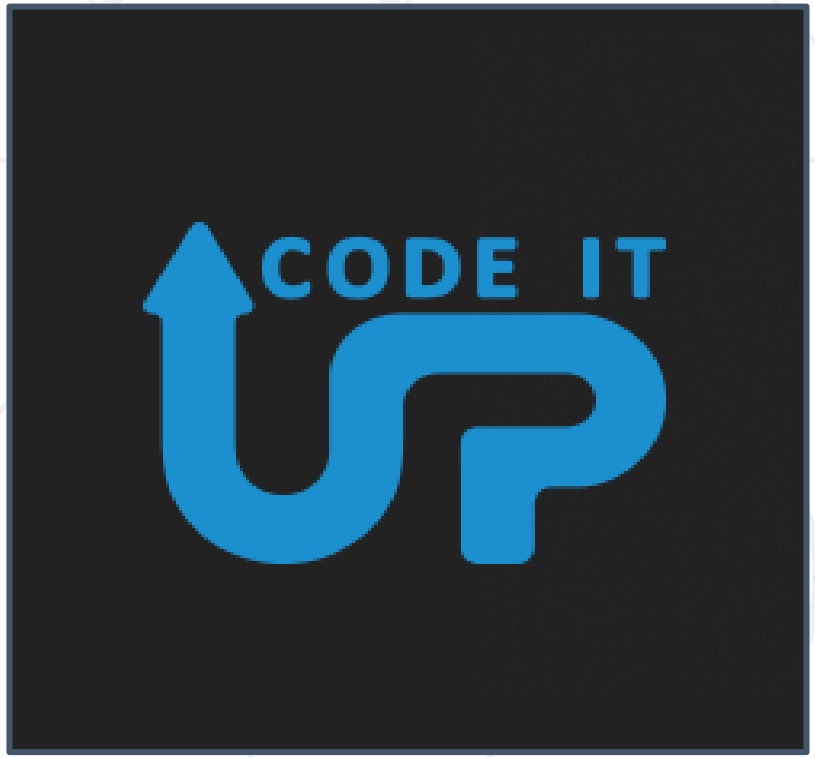
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



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
- Software University Forums
  - forum.softuni.bg

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