# Workshop: Petstagram

This document comprises the fourth segment of the **Petstagram** **Workshop**, focusing on integrating a user into our project. Our agenda for today includes expanding the **user** **model** initially. Subsequently, we'll proceed to incorporate functionalities such as registration, login, and logout. Additionally, we'll address the profile templates and undertake code refactoring across all applications to seamlessly integrate the **user** object where necessary.

The full project description can be found in the [**Workshop Description Document**](https://softuni.bg/downloads/svn/python-web/Jan-2024/Django-Basics/06-Workshop-Part-I/06-Workshop-Description.docx).

## Workshop - Part 4.1

### Extending the Abstract Base User Model

There are several ways that we can choose from to implement our user model. The way Django handles authorization is a perfect fit for our project but still. So, we are going to **create a custom user model inheriting from the** **AbstractBaseUser** Django class. The **model** consists of the following **fields**:

* **email** - **email** field, **required** and **unique**.
* **is\_active** - **boolean** field with **default** value **True**.
* **is\_staff** - **boolean** field with **default** value **False**.

**USERNAME\_FIELD** - specifies the **field** to be used as the **unique** identifier for **authentication**, which is **email**.

**REQUIRED\_FIELDS** - an empty **list**, implying that no additional fields are **required** when creating a **user**.

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

Then, we will need to update the **settings.py** defining the **AUTH\_USER\_MODEL** property to our custom user model:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Now we are going to **link** our **AppUser** with a custom **Petstagram** profile. Create a **new** **model** called **Profile**:

* **user** - **one-to-one** field, **primary key**, with **CASCADE** **delete** option.
* **first\_name** - **character** field, with **maximum** of **30** characters, **optional**.
* **last\_name** - **character** field, with **maximum** of **30** characters, **optional**.
* **date\_of\_birth** - **date** field, **optional**.
* **profile\_picture** - **URL** field, **optional**.

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

### User Manager

We are now creating a custom **manager** that **creates** **users** and **superusers** for our Django application:

* **create\_user(self, email, password=None, \*\*extra\_fields)** - this **method** is responsible for creating a standard **user instance**.
* **create\_superuser(self, email, password=None, \*\*extra\_fields)** - This method is responsible for creating a **superuser** **instance** with **staff** and **superuser** privileges.

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

We need to **import** our manager in the **AppUser** model:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

**Note: if you are building an app with user authentication, you should consider creating a custom user model at the beginning of the work process.**

As this is a workshop project, for the needs of this course, we will need to **drop the database** and create it all over again. Then, we can **delete all migration files**, and **make the migration** for all apps once again.

### User Admin

To adjust a better-looking **admin** site, we can implement some of the **Django** **Admin** **Site** techniques that we have already mastered in the previous **Python ORM** course. First, we need to create a user change form that will inherit the **UserChangeFrom** in our **forms.py** file:

Картина, която съдържа текст, Шрифт, линия, номер

Описанието е генерирано автоматично

There is the **full** **code** structure in the **admin.py** file:

from django.contrib import admin  
from django.contrib.auth.admin import UserAdmin  
from django.contrib.auth import get\_user\_model  
  
from petstagram.accounts.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',)  
  
 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"),  
 },  
 ),  
 )

### User Registration

To create a user registration form, we can **extend the Django built-in** **UserCreationForm**. Let us add a **forms.py** file in our **accounts** **app** and inherit from the built-in **Django** form:

Картина, която съдържа текст, Шрифт, екранна снимка, номер

Описанието е генерирано автоматично

Let us now **add the form functionality in the register view** (when the user is successfully registered, they should be **redirected to the login page**). As we know more about the CBVs in Django we will try to implement the register functionality using **CreateView**. There we should **add the model**, the **form**, the **template**, and the **success URL**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

We should **change the view** in the path function on **accounts/urls.py** file:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

Next, we will add the form to the **register-page.html** template. We can escape showing the help texts on the web page, by simply adding the concrete fields we want our users to fill. **Note: Do not forget to add a "href" on the Login link**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

However, it is **not enough** - it is missing the placeholders:

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

To add them, we can easily **create a** **template** **filter**. Add a **templatetags** package in the **accounts** app and create a **custom\_filters.py** file. Then, add the implementation of the placeholder and register it:

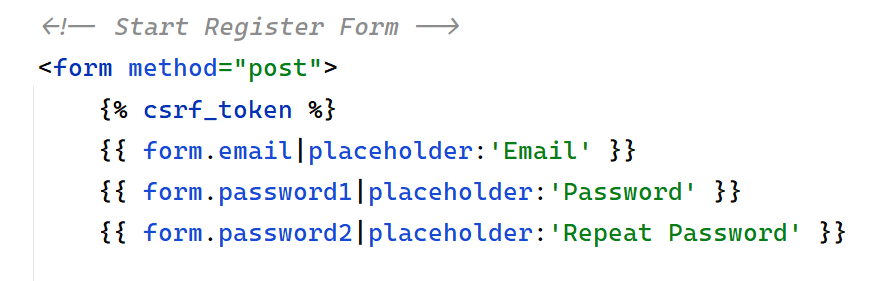
Картина, която съдържа текст, Шрифт, номер, софтуер

Описанието е генерирано автоматично

Now, we can **add the template filter** with the value we want to show on the page:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

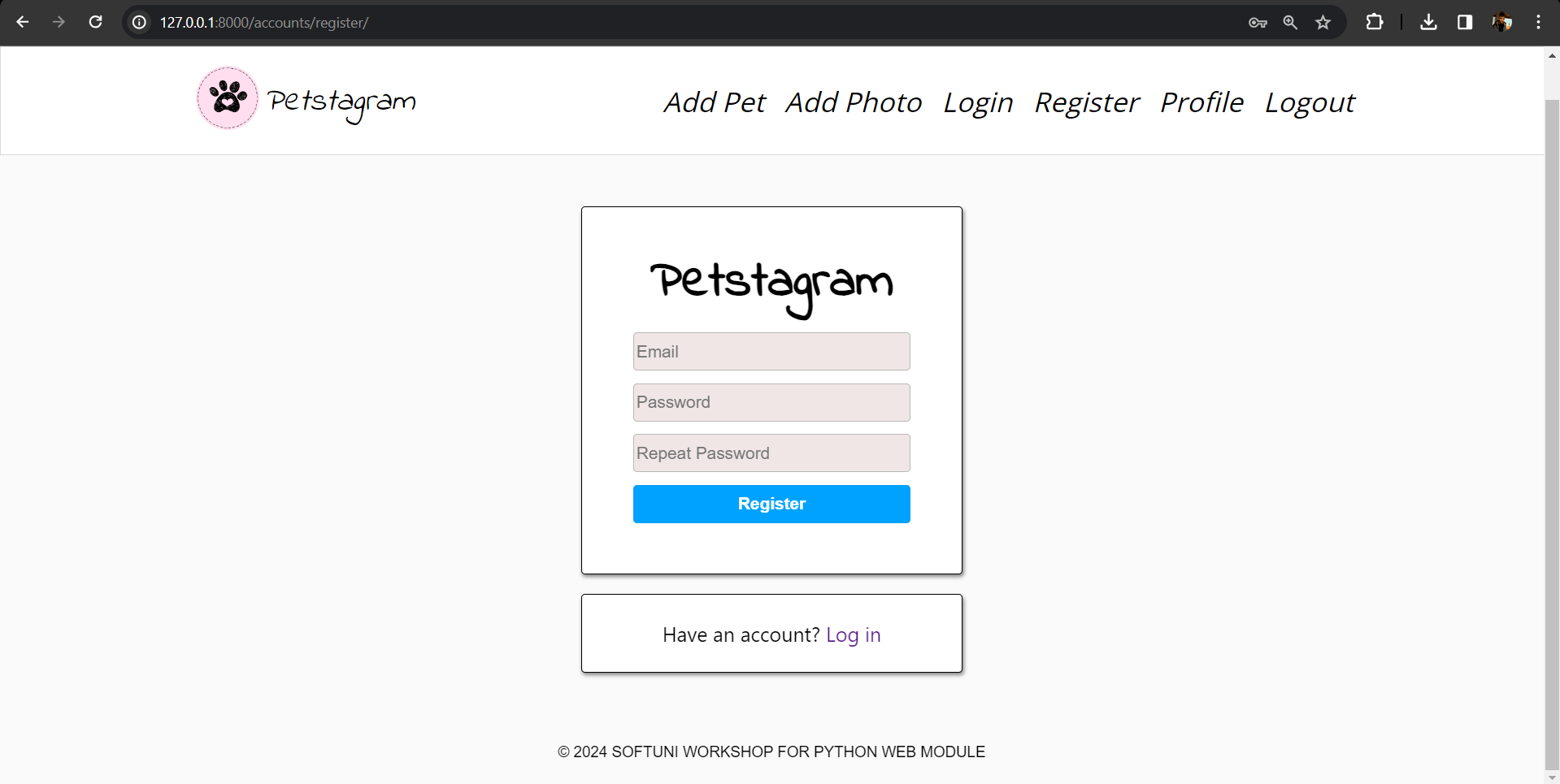
Описанието е генерирано автоматично



Also, we want to **show errors** when they occur. We can decide if they are going to stay above or below the field:



**Restart the development server** and **check** if the implementation **works correctly**:



### User Login

Now, it is time to add a user **login** functionality. First, we will override the **AuthenticationForm** so that we can add placeholders in the fields - "**email**" and "**password**":

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

We can use the built-in view **LoginView** from the **contrib.auth.views** module. It is important to **create** a **default** **profile**. Next, we will **add** the overridden **form**, the **template** we want to use, and the **URL** where requests are **redirected** **after login**:

Картина, която съдържа текст, Шрифт, софтуер, номер

Описанието е генерирано автоматично

Also, in the **settings.py** we need to setup our **login** **redirect** **url**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

The form that we need to override is the **AuthenticationForm**. Look that we are overriding also both **authentication** fields and modifying the **username** field as an **email**:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

Let us refactor the **login-page.html** template by adding the "**form**" attribute. **Note: Do not forget to add the "href" on the Register link.** Then, **start** the development server and **check** if the form is created **correctly**.

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

### User Logout

Let us create the user **logout** functionality. First, we will **add the path** in the **account/urls.py** file:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Next, we will use the Django built-in **LogoutView** to reuse the logout functionality and will **leave it** for later custom implementation if needed:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

In the **settings.py** this time we setup our **logout** **redirect** **url**:

**Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично**

Last, we will **add a button** that the users will use to log out from the app. Let us open the **base.html** file and use a HTML code to **implement a new navigation bar hyperlink**:

Картина, която съдържа текст, софтуер, Компютърна икона, Операционна система

Описанието е генерирано автоматично

**Note**: Since **Django** 5.0, we need to do this through a **POST request**, since it has side-effects. The fact that it worked with a **GET** request was (likely) a violation of the **HTTP** protocol: it made it possible for certain scripts to **log** **out** **users**, without the **user** wanting to.

Add the following **CSS** styles in the **css/home.css** file:

.button {  
 background-color: white;  
 border: none;  
 font-size: 22px;  
 margin: 2.5px 0px 0px 8px;  
 padding: 0;  
 cursor: pointer;  
}

## Workshop - Part 4.2

### Refactor Navigation Bar

Now, as we have a user, we need to make some changes to the app. Let us start by refactoring the navigation bar. We want to **show the links "Profile" and "Logout" to authenticated users only**, and the links **"Register" and "Login" to unauthenticated users only**. Open the **base.html** template and add the condition:

...

{% if not request.user.is\_authenticated %}  
 <li class="nav-item">  
 *<!-- Link to Login Page -->* <a href="{% url 'login' %}">  
 <i>Login</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Register Page -->* <a href="{% url 'register' %}">  
 <i>Register</i>  
 </a>  
 </li>  
{% else %}  
  
 <li class="nav-item">  
 *<!-- Link to Profile Page -->* <a href="#">  
 <i>Profile</i>  
 </a>  
 </li>  
  
 <form method="post" action="{% url 'logout' %}">  
 {% csrf\_token %}  
 <button class="button" type="submit"><i>Logout</i></button>  
 </form>  
  
{% endif %}

...

Next, we can **implement the Profile Page URL**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

### User Editing

We will continue to implement the user functionality by adding a user edit form. We want the form to **contain** the **first name**, **last name**, **date of birth** and **profile picture**:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

Then, we will **implement the profile edit view** that will inherit from the **UpdateView** class:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

Next, we will **refactor the code** in the **profile-edit-page.html** template to implement the user form using the Django template language:

{% extends 'base.html' %}  
{% block content %}

*<!-- Start Edit Profile Section -->* <div class="edit-delete">  
 <h2>Edit Profile</h2>

*<!-- Start Edit Profile From -->* <form method="post">  
 {% csrf\_token %}  
  
 {% for field in form %}

<p class="error"> {{ field.errors }} </p>  
 <div class="label-input">  
 <label>{{ field.label }}</label>  
 {{ field }}  
 </div>  
 <br>

{% endfor %}  
  
 *<!-- Edit Profile Button -->* <button class="edit-btn" type="submit">Edit</button>

</form>  
 *<!-- End Edit Profile Form -->*

</div>  
 *<!-- End Edit Profile Section -->*

{% endblock %}

The **edit** **profile** **form** should be like this:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

## Workshop - Part 4.3

### Refactor Models

We should **connect the project models to the user**. Let us start by **adding the user to the pet's model**:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

Next, we can connect the user to the **photos, the comments, and the likes**. Make the **migration** files and **migrate** the changes to the database.

When a user creates a pet or a photo they should be **automatically added to the model**. Let us implement it:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

Implement the functionality to the **add\_photo** view, too. **Note**: **when our form includes many-to-many fields, we should also call form.save\_m2m() after saving the model instance**. Make changes to the forms if needed.

We should also **add the user to the like and comment views**. Let us start by refactoring the code in the **like\_functionality** view:

**Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично**

Next, **add the user to the comment** they post:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

### Refactor Pet Photo Posts

Let us refactor the **pet-posts.html** template. We want for **each picture to show to the user who uploaded it**. Let us open the file and start implementing the user information:

Картина, която съдържа текст, екранна снимка, номер, Шрифт

Описанието е генерирано автоматично

On the next rows we can see that we should **show the** **first and/or the last name of the user if they exist**. Otherwise, we should **show Anonymous User**. To do that, we can escape writing code in the template by **adding an additional method to the user model**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Then, we can **use the method in the template**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

We should **refactor the pet details URL**, too:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

Start the development server and check the visualization.

### Refactor Photo Details Page

Add the **photo owner's data**, and **the comment user data** and **implement the like button visualization**, so the current user sees if they liked the photo or not. The **buttons edit** and **delete** **should be only visible to the owner** of the photo.

**Hint: you can do that by checking if the photo and its creator id is related to the id of the logged user.**

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

### Refactor Profile Details Page

It is time to **add the user data to the profile**. We will use a CBV to incorporate both the **user** **model** and the **template**, and we will additionally **add the total count of likes in the context**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

In the **profile-details-page.html** template we will **add the user data**, the **count of all user photos**, the **count of all user pets**, and **the count of all likes** that the user has collected, **all user pets**, and **all user photos**.

### Check for Additional Changes

Do not forget to check all functionality on the app to see if everything works correctly:

* Check if the **user object** is needed elsewhere.
* Check all **links** and **buttons**.
* **Login is required** for all pages and buttons, **except** for the **home page**, and **share post button**.

Example: in the **pet** **app**, the **show\_pet\_details** view should **find the owner of the pet** and use it to **visualize the edit/delete button** **to the owner** of the pet **only**:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

## Workshop - Part 4.4

### Add user delete functionality

We will add the final user functionality for this workshop. When a user is deleted, all their **photos**, **pets**, **likes**, and **comments** **should be deleted** too. Then, the app **redirects to the home page** with no authentication:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

We should refactor the **profile-delete-page.html** template:

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

### Django Signals

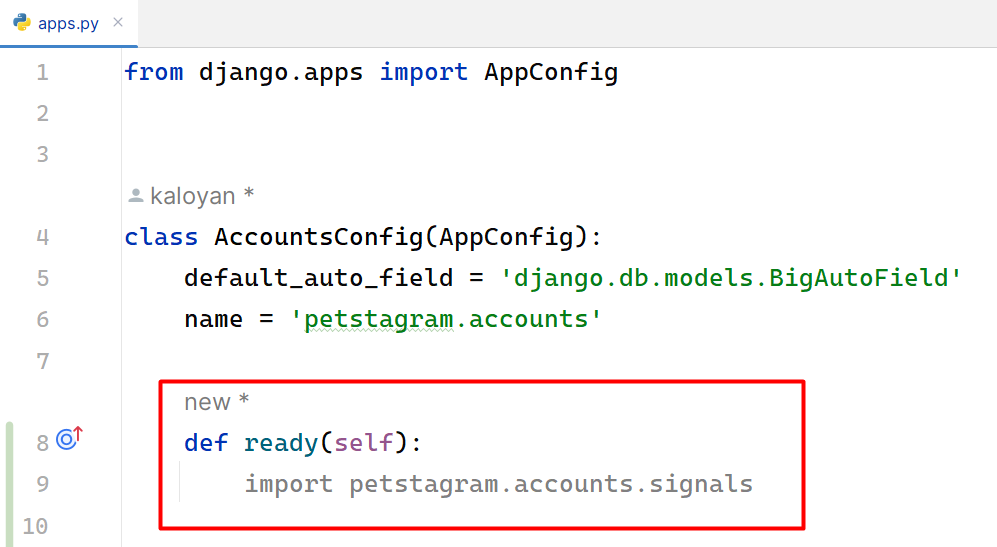
Creating relationship between **AppUser** and **Profile** through the **login** view is not the most accurate way. We can do this with **Django** **signals**. Feel free to **delete** the **form\_valid** method from the **login** view.

In the **accounts** create a new file called **signals.py** and create a **create\_profile** **signal**:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

**Note: it is important to import our signals from the accounts/apps.py file:**



## Workshop - Part 4.5 - Homework

### Extend Profile Details Page

We can finalize our application functionality by extending the **profile** **page**. We must do two things:

* **Link Pet Details Page.**
* **Link Uploaded Pet Photos (descending).**

Inthe **profile-details-page.html** we are going to link our **pets** by defining the **URL** of the **pet**, its **photo** and **name**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Next, we are going to link our **photos**, starting from the **last** **uploaded** one.

Картина, която съдържа текст, екранна снимка, номер, софтуер

Описанието е генерирано автоматично

Update the **profile-details-page.html:**

Картина, която съдържа текст, екранна снимка, номер, Шрифт

Описанието е генерирано автоматично