TECHNICAL UNIVERSITY OF KOSICE
Department of Computer and Informatics

OPERATING SYSTEM

GUI-Based File and Text Transfer Tool

2024 **ANUSYA SENTHIL KUMAR**

CONTENT

1. Objective:

Design and implement a Python-based Graphical User text and files between local and remote systems

Backend:

2. views.py Documentation:

This file contains the views for handling the business logic in your Django application. It covers the following main functionalities:

- Creating and displaying a Person form.
- Searching for a Person object by ID.
- Uploading and displaying images.
- Deleting images.

a. person_form(request):

Purpose:

This view handles the creation or updating of a Person object. It renders a form where users can input data, and upon form submission, saves the Person object in the database.

Parameters:

- request: The HTTP request object. It contains the form data when a user submits the form via POST or GET request for displaying an empty form.

Flow:

- If the request method is POST, it checks whether the form is valid. If valid, the form data is saved, and the user is redirected to the success page.
- If the request method is GET, it renders an empty form for the user to fill

out.

Code:

```
def person_form(request):
    if request.method == 'POST':
        form = PersonForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('success')
    else:
        form = PersonForm()
    return render(request, 'ftpapp/person_form.html', {'form': form})
```

Template Rendered:

- 'ftpapp/person_form.html': Displays the PersonForm to the user.

Redirect:

- Redirects to the success page upon successful form submission.

b. success(request):

Purpose:

This view renders a success page indicating that the form was successfully submitted.

Parameters:

- request: The HTTP request object (though not used in the view).

Flow:

- It simply renders a success message or page to the user after they have successfully created or updated a Person.

Code:

```
def success(request):
    return render(request, 'ftpapp/success.html')
```

Template Rendered:

- 'ftpapp/success.html': Displays a success message to the user.

c. search_person(request):

Purpose:

This view allows users to search for a Person object by their ID. If found, it will display the Person; if not, it will display None.

Parameters:

- request: The HTTP request object. It contains the query parameter (id) to search for a specific Person.

Flow:

- If the id query parameter is provided in the request, the view tries to fetch the Person object using get_object_or_404(). If the object is found, it is passed to the template. Otherwise, it displays None.

Code:

```
def search_person(request):
    person = None
    query = request.GET.get('id')
    if query:
        try:
            person = get_object_or_404(Person, id=query)
        except:
            person = None
    return render(request, 'ftpapp/search_person.html', {'person': person})
```

Template Rendered:

- 'ftpapp/search_person.html': Displays the search result (either the found Person object or None).

d. upload_image(request):

Purpose:

This view handles image uploads. It displays a form for users to upload an image, and upon successful submission, it saves the image to the database.

Parameters:

- request: The HTTP request object. It contains both form data (via POST) and the uploaded image file (via FILES).

Flow:

- If the method is POST and an image file is provided, the view attempts to

validate and save the image using the ImageUploadForm.

- If the form is valid, it saves the image and redirects to the show_images page where all uploaded images are displayed.
- If the method is GET, it renders an empty form for the user to upload an image.

Code:

```
def upload_image(request):
    if request.method == 'POST' and request.FILES['image']:
        form = ImageUploadForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('show_images')
    else:
        form = ImageUploadForm()

    return render(request, 'upload_image.html', {'form': form})
```

Template Rendered:

- 'upload_image.html': Displays the form for image upload.

Redirect:

- Redirects to the show_images page after the image is successfully uploaded.

e. show_images(request):

Purpose:

This view displays all uploaded images in the database.

Parameters:

- request: The HTTP request object.

Flow:

- The view queries all UploadedImage objects and passes them to the template for rendering.

Code:

```python
def show_images(request):
    images = UploadedImage.objects.all()
    return render(request, 'show_images.html', {'images': images})
```

Template Rendered:

- 'show_images.html': Displays a list or gallery of all uploaded images.

f. delete_image (request, image_id):

Purpose:

This view handles the deletion of an uploaded image. It receives an image_id, deletes the image, and redirects back to the show_images page.

Parameters:

- request: The HTTP request object (though not used in this view).
- image_id: The ID of the image to be deleted, passed as part of the URL.

Flow:

- The view retrieves the UploadedImage object by its id using get_object_or_404().
- If the image exists, it is deleted, and the user is redirected to the show_images page.

Code:

```python
def delete_image(request, image_id):
    image = get_object_or_404(UploadedImage, id=image_id)

    image.delete()

    return redirect('show_images')
```

Redirect:

- Redirects to the show_images page after successfully deleting the image.

3. urls.py Documentation:

This urls.py file defines the URL routing for your Django application. It maps the URLs users can visit to specific views (functions in views.py) that handle requests and return responses. The file also ensures that media files, such as uploaded images, are served correctly during development.

Frontend :

This code consists of several HTML templates that are part of a web application. The application seems to involve entering, searching for, uploading, and viewing images related to persons. The templates use Django's template language for dynamic content rendering and include Bootstrap 4 for styling.

4. upload.html:

This file defines the HTML structure for a form that allows users to upload an image. It includes:

- **HTML Structure:** Basic HTML tags like <!DOCTYPE html>, <html>, <head>, <body>, etc.

- **Title:** Sets the title of the page to "Upload Image" using the <title> tag.

- **Styles:** Defines styles for the form and its elements using the <style> tag. Here, it includes styles for:
    - Global styles (font family, background color, text color)
    - Headings (text-align, color, margin)
    - Form container (width, max-width, margin, padding, background color, border-radius, box-shadow)
    - Form container heading (text-align, color, margin-bottom)
    - Input fields (width, padding, margin, border, border-radius, font-size)
    - Button (background color, color, font-size, padding, border, border-radius, cursor, width, margin-top, transition)
    - Button hover effect (background color)
    - Form labels (font-size, color, margin-bottom, display)
    - CSRF token styling (margin-bottom)
    - Responsive design for smaller screens (adjustments for headings and form container)

- **Form:** Creates a form element with the method set to "POST" and enctype set to "multipart/form-data" to handle file uploads. It includes:
    - CSRF token (using Django's {% csrf_token %} template tag)
    - File input field with the type set to "file"
    - Submit button with the text "Upload"

5. show_images.html:

This file defines the HTML structure for displaying a grid of uploaded images. It includes:

- **HTML Structure:** Similar to upload.html.
- **Title:** Sets the title of the page to "Uploaded Images" using the <title> tag.

- **Styles:** Defines styles for the image container and individual images using the <style> tag. Here, it includes styles for:
    - Global styles (font family, margin, padding, background color)
    - Headings (text-align, color, margin-top)
    - Image container (display, grid-template-columns, gap, padding, justify-items)
    - Individual image item (background color, border-radius, box-shadow, padding, text-align, transition)
    - Image item hover effect (transform)
    - Images (width, height, max-width, border-radius)
    - Delete link (display, margin-top, color, text-decoration, font-weight, text-align, transition)
    - Delete link hover effect (color)
    - Empty state message (text-align, font-size, color, padding)
    - Upload button (display, margin, padding, background color, color, text-align, text-decoration, font-size, border-radius, width, transition)
    - Upload button hover effect (background color)
    - Responsive design for smaller screens (adjustments for headings and upload button)

- **Looping through Images:** Uses a Django template tag {% for image in images %} to loop through a list of images passed to the template context and display them in a grid. For each image, it displays:
    - The image itself using the {{ image.image.url }} template tag, which gets the URL of the uploaded image.
    - A delete link for each image, allowing users to remove them. It uses the {% url 'delete_image' image.id %} template tag to generate the URL for the delete view function that takes the

image ID as an argument.

6. success.html:

This file defines the HTML structure for a success message after submitting the form in upload.html. It includes:

- **HTML Structure:** Similar to upload.html.
- **Title:** Sets the title of the page to "Success" using the <title> tag.
- **Link to stylesheet:** Includes a link to an external stylesheet (success.css) using the <link> tag. This stylesheet might define styles for the success message and buttons.
- **Styles (inline):** Defines some inline styles using the <style> tag within the <head> section. Here, it imports a font from Google Fonts and sets styles for the body and the "click" class (which might be used for buttons or links).
- **Heading:** Displays a heading "Form

8. Conclusion:

This documentation describes a Django application for managing people and their associated images. It covers the backend views (views.py), URL routing (urls.py), and frontend HTML templates (upload.html, show_images.html, success.html).

References:

- Django : The web framework for perfectionists with deadlines | Django (djangoproject.com)
- Chat GPT : ChatGPT (openai.com)
- Google : www.google.com
- Stack overflow : https://stackoverflow.com
- W3 schools : https://w3schools.com