# Lab 3 - Django: Building a News Application with CRUD, File Upload, and User Authentication

This guide provides a step-by-step breakdown of the lecture content, focusing on building a Django application that includes CRUD operations for news articles, file uploads for profile pictures, and user login and registration forms.

## 1. Setting up the Project

1. **Create a New Database Schema:**
    - Open your workbench and create a new schema named `lab3`.
2. **Configure Database Settings:**
    - In your Django project's `settings.py` file, modify the `DATABASES` dictionary to connect to the MySQL database:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'lab3',
        'HOST': 'localhost',
        'PORT': '3306',
        'USER': 'root',
        'PASSWORD': 'root'
    }
}
```

3. **Create a Django App:**
    - Open your terminal and navigate to your project's directory.
    - Run the following command to create a new app called `vesti`:

```
python.exe .\manage.py startapp vesti
```

4. **Add the App to Settings:**
    - In your project's `settings.py`, add the new app to the `INSTALLED_APPS` list:

```
INSTALLED_APPS = [
    # ... other apps ...
    'vesti.apps.VestiConfig',
]
```

## 2. Defining Models

1. **Create User Model:**
   - In the `models.py` file of your `vesti` app, create a custom user model `Korisnik` that inherits from Django's `AbstractUser`:

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

class Korisnik(AbstractUser):
    br_objavljenih_vesti = models.IntegerField(default=0)
```

2. **Customize User Model:**
   - Add an attribute `br_objavljenih_vesti` to track the number of news articles published by each user.

3. **Set Custom User Model:**
   - In your project's `settings.py`, update the `AUTH_USER_MODEL` setting to use your custom user model:

```
AUTH_USER_MODEL = 'vesti.Korisnik'
```

4. **Create News Article Model:**
   - Define a new model `Vest` to represent news articles:

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

class Vest(models.Model):
    autor = models.ForeignKey(Korisnik, on_delete=models.CASCADE)
    naslov = models.CharField(max_length=50)
    sadrzaj = models.CharField(max_length=300)
    timestamp = models.DateTimeField(default=datetime.datetime.now())

    class Meta:
        db_table = 'Vest'
```

5. **Create Comment Model:**
   - Create a model `Komentar` to handle comments associated with news articles:

```
class Komentar(models.Model):
    vest = models.ForeignKey(Vest, on_delete=models.CASCADE)
    autor = models.ForeignKey(Korisnik, on_delete=models.CASCADE)
    sadrzaj = models.CharField(max_length=300)
    timestamp = models.DateTimeField(default=datetime.datetime.now())

    class Meta:
        db_table = 'Komentar'
```

## 3. Creating Migrations and Applying to Database

1. **Check for Existing Migrations:**
   - Run the following command to list any existing migrations for the `vesti` app:

```
python.exe manage.py showmigrations
```

2. **Generate Migrations:**
   - Create migrations for the changes you made to your models:

```
python.exe manage.py makemigrations vesti
```

3. **Preview SQL Statements:**
   - View the SQL statements that will be executed to create the database tables:

```
python.exe manage.py sqlmigrate vesti 0001
```

4. **Apply Migrations to Database:**
   - Apply the migrations to your database:

```
python.exe manage.py migrate
```

## 4. Creating Views and Templates

1. **Create Views File:**
   - In your `vesti` app, create a new file called `views.py`.
2. **Create Forms File:**

- Create a file named `forms.py` in the `vesti` app to define forms.

3. **Create Search Form:**
   - In `forms.py`, define a simple search form:

```
from django.forms import ModelForm, Form
from django import forms

class SearchForm(Form):
    term = forms.CharField(max_length=50)
```

4. **Create Index View:**
   - In `views.py`, create an index view to display news articles:

```
from django.shortcuts import render
from django.http import HttpRequest
from .forms import *
from .models import *

def index(request: HttpRequest):
    searchform = SearchForm()
    vesti = Vest.objects.order_by('-timestamp')
    context = {
        'searchform': searchform,
        'vesti': vesti
    }
    return render(request, 'vesti/index.html', context)
```

5. **Create Templates:**
   - In your project's `templates` directory:
     - Create a file named `base.html` for the base template.
     - Create a subdirectory `vesti` and inside it create a file named `index.html` for the news list view.

6. **Base Template (`base.html`):**
   - Define the basic structure of your HTML pages, including blocks for the title and content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
            Base title
```

```
        {% endblock %}
    </title>
</head>
<header style="background-color: aquamarine; text-align: right">
{% block header %}

    Base Header

{% endblock %}
</header>

<body>

{% block content %}
    Base content
{% endblock %}

</body>
</html>
```

7. **Index Template (`templates/vesti/index.html`):**
    - Extend the base template and define content for the news list:

```
{% extends 'base.html' %}

{% block title %}
    Vesti
{% endblock %}


{% block content %}
<ul>
{% for vest in vesti %}
    <li>{{vest}}</li>
{% endfor %}
</ul>

<form method="post">
{% csrf_token %}
{{ searchform }}
<input type="submit" value="Search">
</form>

{% endblock %}
```

8. **Update Index View (`views.py`):**
    - Implement search functionality in your index view:

```
def index(request: HttpRequest):
    searchform = SearchForm(data=request.POST or None)
    vesti = []
    if searchform.is_valid():
        term = searchform.cleaned_data.get('term')
        vesti = Vest.objects.filter(sadrzaj__contains=term)
    else:
        vesti = Vest.objects.order_by('-timestamp')
    context = {
        'searchform': searchform,
        'vesti': vesti
    }
    return render(request, 'vesti/index.html', context)
```

9. **Create `urls.py`:**
    - In the `vesti` app, create a file named `urls.py` to define URL patterns for your app:

```
from django.contrib import admin
from django.urls import path
from .views import *
urlpatterns = [
    path('', index, name='home'),
]
```

10. **Include App URLs in Main `urls.py`:**

- In your project's `urls.py`, include the URLs for the `vesti` app:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('vesti.urls'))
]
```

11. **Run the Application:**

- Run the development server:

```
python.exe manage.py runserver
```

- Access your application at `http://127.0.0.1:8000/`.

## 5. User Authentication

1. **Create a Superuser:**
   - Create an administrative user for managing the application:

     ```
     python.exe manage.py createsuperuser
     ```

   - Provide a username, email address, and password.
2. **Register Models in Admin:**
   - In the `admin.py` file of your `vesti` app, register your models:

     ```
     from django.contrib import admin

     # Register your models here.
     from .models import *
     admin.site.register(Vest)
     admin.site.register(Korisnik)
     admin.site.register(Komentar)
     ```

3. **Access Admin Panel:**
   - Go to `http://127.0.0.1:8000/admin/` and log in using the credentials of the superuser you created.
4. **Add Users Manually:**
   - In the admin panel, navigate to the "Users" section and add a few users manually.
5. **Create Login Template:**
   - In the `templates` directory, create a new subdirectory named `registration` and create a file named `login.html` inside it.
6. **Create Login View:**
   - In `views.py`, create a view for handling user login:

     ```
     from django.shortcuts import render, redirect
     from django.http import HttpRequest
     from django.contrib.auth import login, authenticate

     from .forms import *
     from .models import *

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

```
def login_req(request: HttpRequest):
    form = AuthenticationForm(request=request, data=request.POST or
None)
    if form.is_valid():
        username = form.cleaned_data['username']
        password = form.cleaned_data['password']
        user = authenticate(username=username,password=password) # its
important to name the credentials in authenticate.
        if user:
            login(request, user) # takes the user and the request and
adds that current user in the session storage.
            return redirect('home') # if we are autheticated and
loggedin we redirect

    context = {
        'form': form
    }
    return render(request,'registration/login.html',context) # the
render takes in the request, the template and the context.
```

7. **Create Logout View:**

   - Create a view to log out users:

```
from django.shortcuts import render, redirect

# Create your views here.
from django.http import HttpRequest
from django.contrib.auth import login, authenticate, logout

def logout_req(request: HttpRequest):
    logout(request) # this from session storage clears the user
    return redirect('home') # we redirect to home.
```

8. **Update `urls.py`:**

   - Add paths for the login and logout views:

```
from django.contrib import admin
from django.urls import path
from .views import *
urlpatterns = [
    path('', index, name='home'),
    path('login/', login_req, name='login'),
    path('logout/', logout_req, name='logout'),
]
```

9. **Test Login:**
    - Run the application and access the login page at `http://127.0.0.1:8000/login/`.
    - Log in using one of the users you created.
10. **Display User Status in Header:**

- Update the `base.html` template to show the logged-in user's name and provide a logout link:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
            Base title
        {% endblock %}
    </title>
</head>
<header style="background-color: aquamarine; text-align: right">
{% block header %}

    {% if user.is_authenticated %}
        <a href="{% url 'logout' %}">logout</a>
        {{ user }}
        {% else %}
        <a href="{% url 'login' %}">login</a>
    {% endif %}
{% endblock %}
</header>

<body>

{% block content %}
    Base content
{% endblock %}

</body>
</html>
```

11. **Create Registration Template:**

- Create a new file named `registration.html` in the `templates/registration` directory.

12. **Create Registration View:**

- In `views.py`, create a view for handling user registration:

```
from django.shortcuts import render, redirect
```

```
# Create your views here.
from django.http import HttpRequest
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib import messages
from django.contrib.auth.models import Group

from .forms import *
from .models import *

def registration(request: HttpRequest):
    form = UserCreationForm(request.POST, request.FILES)
    if form.is_valid():
        user: Korisnik = form.save()
        group = Group.objects.get(name='default')
        user.groups.add(group)
        login(request, user)
        return redirect('home')

    context = {
        'form': form
    }
    return render(request,'registration/registration.html',context)
```

13. **Create Custom Registration Form:**

- In `forms.py`, create a custom registration form:

```
from django.forms import ModelForm, Form
from django import forms
from django.contrib.auth.forms import  UserCreationForm
from .models import *

from django.core.exceptions import ValidationError

class KorisnikCreationForm(UserCreationForm):

    class Meta:
        model = Korisnik
        fields = ['username', 'password1','password2']

class SearchForm(Form):
    term = forms.CharField(max_length=50)
```

14. **Update Registration View:**

- Use the custom registration form in the view:

```
def registration(request: HttpRequest):
    form = KorisnikCreationForm(request.POST, request.FILES)
    if form.is_valid():
        user: Korisnik = form.save()
        group = Group.objects.get(name='default')
        user.groups.add(group)
        login(request, user)
        return redirect('home')

    context = {
        'form': form
    }
    return render(request,'registration/registration.html',context)
```

15. **Add Registration URL:**

- Add a path for the registration view in `urls.py`:

```
from django.contrib import admin
from django.urls import path
from .views import *
urlpatterns = [
    path('', index, name='home'),
    path('login/', login_req, name='login'),
    path('logout/', logout_req, name='logout'),
    path('register/', registration, name='register')
]
```

16. **Add Error Handling and Messages:**

- Update the login view to display error messages if login fails:

```
from django.shortcuts import render, redirect

# Create your views here.
from django.http import HttpRequest
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib import messages
from django.contrib.auth.models import Group

from .forms import *
from .models import *

def login_req(request: HttpRequest):
    form = AuthenticationForm(request=request, data=request.POST or None)
    if form.is_valid():
        username = form.cleaned_data['username']
```

```
        password = form.cleaned_data['password']
        user = authenticate(username=username,password=password)
        if user:
            login(request, user)
            messages.info(request,'Succesful login')
            return redirect('home')
    else:
        messages.error(request, 'Fail login')
    context = {
        'form': form
    }
    return render(request,'registration/login.html',context)
```

17. **Display Messages in Template:**

- Update the `base.html` template to display messages using the `messages` framework:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
            Base title
        {% endblock %}
    </title>
</head>
<header style="background-color: aquamarine; text-align: right">
{% block header %}

    {% if messages %}
        {% for message in messages %}
            {{ message }}
        {% endfor %}
            =|=
    {% endif %}

    {% if user.is_authenticated %}
        <a href="{% url 'logout' %}">logout</a>
        {{ user }}
        {% else %}
        <a href="{% url 'login' %}">login</a>
    {% endif %}
{% endblock %}
</header>

<body>

{% block content %}
    Base content
{% endblock %}
```

```
</body>
</html>
```

## 6. Adding Permissions and CRUD Operations

1. **Create Create News Article View:**
   - In `views.py`, create a view to handle the creation of news articles:

```
@login_required(login_url='login')
def create_vest(request: HttpRequest):
    form = VestForm(request.POST or None)
    if form.is_valid():
        vest = form.save(commit=False)
        vest.autor =
Korisnik.objects.get(username=request.user.get_username())
        vest.save()
        return redirect('home')
```

2. **Create Delete News Article View:**
   - Create a view to delete news articles:

```
@login_required(login_url='login')
@permission_required('vesti.delete_vest', raise_exception=True)
def delete_vest(request: HttpRequest):
    vest_id = request.POST.get('vest_id')
    if vest_id:
        vest = Vest.objects.get(pk=vest_id)
        if vest.autor == request.user or
request.user.has_perm('vesti.delete_vest'):
            vest.delete()

    return redirect('home')
```

3. **Create VestForm:**
   - In `forms.py`, create a form for creating news articles:

```
class VestForm(ModelForm):
    class Meta:
        model = Vest
        exclude = ['autor']
```

4. **Update Index Template:**

   - Add buttons for deleting news articles and a form for creating new articles:

```
{% extends 'base.html' %}

{% block title %}
    Vesti
{% endblock %}


{% block content %}

<form method="post">
{% csrf_token %}
{{ searchform }}
<input type="submit" value="Search">
</form>

<ul>
    {% for vest in vesti %}
        <li>
        <h4>{{ vest.naslov }}</h4>
        <p>{{ vest.sadrzaj }}</p> -- {{ vest.autor }}


        {% if user == vest.autor or perms.vesti.delete_vest %}
            <form method="post" action="{% url 'delete_vest' %}">
            {% csrf_token %}
            <button type="submit" value="{{ vest.id }}"
name="vest_id">Delete</button>
            </form>
        {% endif %}
        </li>
    {% endfor %}
</ul>

{% if user.is_authenticated %}
<form method="post" action="{% url 'create_vest' %}">
{% csrf_token %}
{{ vestform }}
<input type="submit" value="Create">
</form>
{% endif %}
{% endblock %}
```

5. **Update Index View:**

   - Pass the `vestform` to the template and add logic for searching by author:

```
from django.shortcuts import render, redirect

# Create your views here.
from django.http import HttpRequest
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.forms import AuthenticationForm,
UserCreationForm
from django.contrib import messages
from django.contrib.auth.models import Group
from django.db.models import Q

from .forms import *
from .models import *
from django.contrib.auth.decorators import login_required,
permission_required

def index(request: HttpRequest):
    searchform = SearchForm(data=request.POST or None)
    vesti = []
    if searchform.is_valid():
        term = searchform.cleaned_data.get('term')
        vesti = Vest.objects.filter(Q(sadrzaj__contains=term) |
Q(autor__username__contains=term))
    else:
        vesti = Vest.objects.order_by('-timestamp')
    context = {
        'searchform':searchform,
        'vesti':vesti,
        'vestform': VestForm()
    }
    return render(request,'vesti/index.html',context)
```

6. **Add URLs for New Views:**

   - Add paths for the new views in `urls.py`:

```
from django.contrib import admin
from django.urls import path
from .views import *
urlpatterns = [
    path('', index, name='home'),
    path('login/', login_req, name='login'),
    path('logout/', logout_req, name='logout'),
    path('register/', registration, name='register'),
    path('create_vest/', create_vest, name='create_vest'),
    path('delete_vest/', delete_vest, name='delete_vest'),

]
```

# 7. Adding Profile Pictures

1. **Update User Model:**
   - Add a field for storing profile pictures in the `Korisnik` model:

```
class Korisnik(AbstractUser):
    br_objavljenih_vesti = models.IntegerField(default=0)
    pfp = models.ImageField(upload_to='imgs/',null=True) # need Pillow
library
```

2. **Configure Media Settings:**
   - In your project's `settings.py`, add the following settings to define the media URL and root directory:

```
MEDIA_URL = '/media/'  # folder name

import os
MEDIA_ROOT = os.path.join(BASE_DIR, 'media') # path to folder
```

3. **Add Media URL Pattern:**
   - In your project's `urls.py`, add a pattern for serving media files:

```
from django.conf.urls.static import static
from django.conf import settings


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('vesti.urls'))
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

4. **Run Migrations:**
   - Generate and apply migrations for the changes to your models:

```
python.exe manage.py makemigrations
python.exe manage.py migrate
```

5. **Upload Profile Picture in Admin:**
    - Log in to the admin panel and upload a profile picture for one of the users.
6. **Display Profile Picture in Template:**
    - Update the `index.html` template to display the profile picture:

```
{% extends 'base.html' %}

{% block title %}
    Vesti
{% endblock %}


{% block content %}

<form method="post">
{% csrf_token %}
{{ searchform }}
<input type="submit" value="Search">
</form>

<ul>
    {% for vest in vesti %}
        <li>
        <h4>{{ vest.naslov }}</h4>
        <p>{{ vest.sadrzaj }}</p> -- {{ vest.autor }}
        {% if vest.autor.pfp %}
            <img src="{{ vest.autor.pfp.url }}" alt="img" width="50px"
height="50px">
            {% else %}
            <img src="" alt="img">
        {% endif %}

        {% if user == vest.autor or perms.vesti.delete_vest %}
            <form method="post" action="{% url 'delete_vest' %}">
            {% csrf_token %}
            <button type="submit" value="{{ vest.id }}"
name="vest_id">Delete</button>
            </form>
        {% endif %}
        </li>
    {% endfor %}
</ul>

{% if user.is_authenticated %}
<form method="post" action="{% url 'create_vest' %}">
{% csrf_token %}
{{ vestform }}
<input type="submit" value="Create">
</form>
{% endif %}
```

```
{% endblock %}
```

## 8. File Upload during Registration

1. **Update Registration Template:**
   - Add the `enctype` attribute to the registration form:

```
{% extends 'base.html' %}

{% block title %}
    Register
{% endblock %}

{% block content %}
<form method="post" enctype="multipart/form-data">
{% csrf_token %}
{{form}}
<input type="submit" value="Register">
</form>
    <br>
    Imas profil? Uloguj se<a href="{% url 'login'%}">ovde!</a>

{% endblock %}
```

2. **Update Registration View:**
   - Make sure the view handles `request.FILES`:

```
def registration(request: HttpRequest):
    form = KorisnikCreationForm(request.POST, request.FILES)
    if form.is_valid():
        user: Korisnik = form.save()
        group = Group.objects.get(name='default')
        user.groups.add(group)
        login(request, user)
        return redirect('home')

    context = {
        'form': form
    }
    return render(request,'registration/registration.html',context)
```

3. **Update Registration Form:**
   - Add the `pfp` field to the `KorisnikCreationForm`:

```
class KorisnikCreationForm(UserCreationForm):

    class Meta:
        model = Korisnik
        fields = ['username', 'password1','password2','pfp']
```

Now you have a fully functional Django application with CRUD operations, file uploads, and user authentication. Remember to customize the templates and add more features as needed.