RV Educational Institutions
RV College of Engineering
Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi
Go, change the world

COMPUTER NETWORKS ASSIGNMENT 22MCA13TL

On

"File Transfer using File Transfer Protocol"

Submitted by

Ashish Garg
SAP ID: RVCE22MCA001

Leandra Anderson
SAP ID: RVCE22MCA037

Under the Guidance
Of
Dr. Deepika K
Assistant Professor
Department of MCA
RV College of Engineering
Bengaluru – 560059

Introduction:

The purpose of this project was to design a file transfer protocol (FTP) platform that would allow college professors to upload any document to a website. Additionally, the platform would allow students to download files only if they were connected to the college network. The platform was designed to provide a secure and reliable means of sharing documents between professors and students within the college environment.

Design: The FTP platform was designed with the following features:

- **Login System**: The platform requires users to login with their college credentials before they can access any files. This ensures that only authorized users canaccess the platform.

- **User roles**: The platform has two user roles - professors and students. Professors have the ability to upload files while students can only download files.

- **File upload**: Professors can upload any document tothe platform. The platform accepts a wide range of document formats including PDF, DOC, DOCX, PPT, and PPTX.

- **File download**: Students can only download files ifthey are connected to the college network. This ensures that files are only accessible to authorizedusers.

- **File management**: The platform allows professors tomanage their uploaded files. Professors can delete files or update them as needed.

- **Security**: The platform uses industry-standard security measures to ensure that files are protected from unauthorized access or hacking attempts. This includes encryption of user data and files as well as regular security audits

Implementation

The FTP platform was implemented using a combination of technologies. The platform was hosted on a secure web server that was accessible only to authorized users

The login system was implemented using Java Script and Ajax. User credentials were stored in manage.py and encrypted using industry-standard encryption.

The file upload and download features were implemented packages. Files were stored in a secure directory on the web server and accessed only by authorized users

The file management feature was implemented using packages. Professors were able to manage their uploaded files through a web-based interface that allowed them to delete or update files as needed.

The security features of the platform were implemented through a combination of server-side and client-side measures. Server-side measures included encryption of user data and files as well as regular security audits. Client-side measures included the use of SSH encryption to protect data in transit and the use of secure login credentials.

Conclusion:

The FTP platform designed in this project provides a secure and reliable means of sharing documents between professors and students within the college environment. The platform is easy to use and offers a range of features including file upload, download, and management. The security features of the platform ensure that files are protected from unauthorized access or hacking attempts.

Code Explanation:

1. Imports

```
from django.shortcuts import render
from django.http import JsonResponse, HttpResponse, Http404
from ipware import get_client_ip
import ipaddress
from .models import Users
from django.contrib.auth.hashers import check_password
from Crypto.PublicKey import RSA
import os
```

- render: Renders HTML templates with context.
- JsonResponse: Sends JSON responses.
- HttpResponse and Http404: Used to handle HTTP responses and 404 errors.
- get_client_ip: Gets the client's IP address.
- ipaddress: Module for handling IP addresses and networks.
- Users: User model (imported from the local app).
- check_password: Checks hashed passwords.
- RSA: Generates RSA keys.
- os: Handles file system operations.

2. ipCheck Function

```
def ipCheck(request):
    request.META['REMOTE_ADDR'] = '172.16.1.1'
    client_ip, is_routable = get_client_ip(request)
    if client_ip is not None:
        if ipaddress.IPv4Address(client_ip) in ipaddress.IPv4Network('172.16.0.0/12'):
            return render(request, 'login.html', {'client_ip': client_ip})
        else:
            return render(request, 'not_found.html')
    else:
        return render(request, 'not_found.html')
```

- Sets the REMOTE_ADDR to a static IP address for testing.
- Gets the client's IP address and checks if it's in the specified range.
- Renders login.html if the IP is in the range, otherwise renders not_found.html.

3. **auth Function**

```
def auth(request):
    email = request.POST.get('email')
    password = request.POST.get('password')
    userData = Users.objects.get(username=email)
    is_password_correct = check_password(password, userData.password)

    if is_password_correct:
        request.session['username'] = email
        keygen(email)
        return JsonResponse({'status': 'Success'})
    else:
        return JsonResponse({'status': 'Error'})
```

- Authenticates users based on email and password.
- If authentication is successful, it generates RSA keys for the user and stores the email in the session.

4. **keygen Function**

```
def keygen(email):
    print('Generating.....')
    if not os.path.exists('keys'):
        os.makedirs('keys')
at_index = email.index('@')
    fn = email[:at_index]+'.key'
    filepath = os.path.join('keys/', fn)

    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()

    with open(filepath, 'wb') as f:
        f.write(private_key)

    filepath = os.path.join('keys/', 'authorized_keys')
    with open(filepath, 'wb') as f:
        f.write(public_key)
```

- Generates RSA keys (2048 bits) and saves the private key to a file named after the user's email (before the '@').
- Saves the public key to a file named authorized_keys.

5. **home Function**

```
def home(request):
    folder_path = 'public'
    folders = []
    files = []
    for item in os.listdir(folder_path):
        item_path = os.path.join(folder_path, item)
        if os.path.isdir(item_path):
            folders.append(item)
        else:
            files.append(item)
context = {
        'folders': folders,
        'files': files
    }
return render(request, 'home.html', context)
```

- Lists folders and files in the public directory.
- Renders home.html with the list of folders and files.

6. **folder Function**

```
def folder(request):
    is_allowed = check_keys(request)
    admin = is_admin(request)
    if is_allowed and admin:
        name = request.POST.get('name')
        folder_path = os.path.join('public', name)
        try:
            os.makedirs(folder_path)
            return JsonResponse({'status': 'Success'})
        except OSError:
            return JsonResponse({'status': 'Error'})
    else:
        return JsonResponse({'status': 'Error'})
```

- Creates a folder in the public directory if the user is authenticated and an admin.
- Responds with success or error status.

7. **upload_folder Function**

```
def upload_folder(request):
    print('Uploading.....')
    is_allowed = check_keys(request)
    admin = is_admin(request)
    if is_allowed and admin:
        if request.method == 'POST':
            files = request.FILES.getlist('folder')
            folderName = request.POST.get('folder_name')
            folderPath = os.path.join('public', folderName)
            if not os.path.exists(folderPath):
                os.makedirs(folderPath)
for f in files:
                destination = os.path.join(folderPath, f.name)
                with open(destination, 'wb+') as destination_file:
                    for chunk in f.chunks():
                        destination_file.write(chunk)
return JsonResponse({'status': 'Success'})
    else:
        return JsonResponse({'status': 'Error'})
```

- Handles uploading multiple files to a specified folder in the public directory if the user is authenticated and an admin.

8. **get_files Function**

```
def get_files(request):
    folder = request.GET.get('folder')
    if folder:
        folder_path = os.path.join('public', folder)
        files = os.listdir(folder_path)
    else:
        files = []
context = {
        'folder': folder,
        'files': files,
    }
    return render(request, 'files.html', context)
```

- Lists files in a specified folder within the public directory and renders them in files.html.

9. **download_file Function**

```
def download_file(request):
    print('Uploading.....')
    is_allowed = check_keys(request)
    if is_allowed:
        file_name = request.GET.get('file')
        folder_name = request.GET.get('folder')
        if folder_name is not None:
            file_path = os.path.join('public', folder_name, file_name)
            if os.path.exists(file_path):
                with open(file_path, 'rb') as file:
                    response = HttpResponse(file, content_type='application/octet-stream')
                    response['Content-Disposition'] = f'attachment; filename="{file_name}"'
                    return response
        else:
            raise Http404("File not found")
    else:
        return JsonResponse({'status': 'Error'})
```

- Allows downloading a file from the public directory if the user is authenticated.

10. **upload_file Function**

```
def upload_file(request):
    print('Uploading.....')
    is_allowed = check_keys(request)
    admin = is_admin(request)
    if is_allowed and admin:
        if request.method == 'POST':
            fileName = request.FILES.get('file')
            folder = request.POST.get('folder')
            file_path = os.path.join('public',folder, fileName.name)
            with open(file_path, 'wb') as destination:
                for chunk in fileName.chunks():
                    destination.write(chunk)
            return JsonResponse({'status': 'Success'})
    else:
        return JsonResponse({'status': 'Error'})
```

- Handles uploading a single file to a specified folder in the public directory if the user is authenticated and an admin.

11. check_keys Function

```
def check_keys(request):
    print("Verifying.....")
    email = request.session.get('username')
    at_index = email.index('@')
    private_key_filename = email[:at_index] + '.key'
    private_key_filepath = os.path.join('keys/', private_key_filename)
    public_key_filepath = os.path.join('keys/', 'authorized_keys')
with open(private_key_filepath, 'rb') as f:
        private_key = RSA.import_key(f.read())
with open(public_key_filepath, 'rb') as f:
        public_key = RSA.import_key(f.read())
return private_key.publickey().export_key() == public_key.export_key()
```

- Verifies that the public key matches the private key for the authenticated user.

12. is_admin Function

```
def is_admin(request):
    email = request.session.get('username')
    userData = Users.objects.get(username=email)
    return userData.is_admin == 1
```

- Checks if the authenticated user has admin privileges.
- Overall, this code manages user authentication, file and folder operations, and key management using RSA encryption in a Django application.