Informe de Laboratorio 08

Tema: Django Rest Framework

Nota

Estudiante	Escuela	Asignatura
Sebastian Arley Chirinos Negrón/ Gabriel Eduardo Soto Ccoya schirinosne@unsa.edu.pe / gsotocco@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Estructura de datos y Algoritmos Semestre: I Código: 1702124

Laboratorio	Tema	Duración
08	Django Rest Framework	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	26 Junio 2023	3 Julio 2023

1. Tarea

- Introducción En este informe, se presenta la implementación de B+.

2. URL de Repositorio Github

- URL para el laboratorio 04 en el Repositorio GitHub.
- `https://github.com/GabSoto/Pweb02_Lab08`

3. Tarea

3.1. Django Rest Framework

- En sus grupos de trabajo correspondientes. Elabore un servicio web que tenga un CRUD con el uso de este framework.

  En este párrafo, se solicita a los grupos de trabajo que elaboren un servicio web que implemente las operaciones CRUD (Create, Read, Update, Delete) utilizando un framework específico. El CRUD es una operación básica que se refiere a las acciones de crear, leer, actualizar y eliminar datos en una base de datos o sistema de almacenamiento. El servicio web debe ser capaz de realizar estas operaciones para gestionar los datos de manera eficiente.

- Create - POST

- Read - GET
- Update - PUT
- Delete - DELETE
- Aquí se describen los métodos HTTP asociados con cada una de las operaciones CRUD. Estos métodos son estándares en la comunicación REST (Representational State Transfer) y son utilizados para realizar diferentes acciones sobre los recursos en un servicio web. Por ejemplo, para crear un nuevo recurso, se utiliza el método POST; para leer o recuperar un recurso, se utiliza el método GET; para actualizar un recurso existente, se utiliza el método PUT; y para eliminar un recurso, se utiliza el método DELETE.
- Centrarse en el Core business de su aplicación web. Los más importante y necesario que esté disponible a través de un servicio web.En esta parte del texto, se destaca la importancia de centrarse en el Çore business"de la aplicación web al diseñar el servicio web. El término Çore business"se refiere a la parte central o núcleo de la aplicación, que contiene las funcionalidades más importantes y necesarias para el funcionamiento principal de la aplicación. Al implementar el servicio web, es esencial priorizar las funcionalidades principales y asegurarse de que estén disponibles a través del servicio web para que puedan ser utilizadas por los clientes y usuarios.

3.2. models

- Este código define un modelo de Django, que es una clase de Python que representa una tabla de base de datos. Django es un marco de trabajo web que te permite construir aplicaciones web utilizando el lenguaje de programación Python, y proporciona un sistema de mapeo objeto-relacional (ORM) para interactuar con bases de datos.
- Vamos a revisar el código paso a paso:
- from django.db import models: Esta línea importa los módulos necesarios del módulo de base de datos de Django. Esto te permite usar la funcionalidad de los modelos de Django para definir la estructura de la tabla de base de datos.
- class data user models.Model:: Esta línea define un modelo de Django llamado data user. Es una subclase de models.Model, lo que significa que hereda todas las funcionalidades de la clase de modelo base de Django. Al heredar de models.Model, esta clase tiene todas las funcionalidades de un modelo de Django.
- Dentro de la clase data user, defines varios campos que representan las columnas en la tabla de base de datos:
- dni Esto define un campo de caracteres (una cadena) con una longitud máxima de 8 caracteres. Representa el "dni"(número de documento) del usuario.
- nombre = models.CharField Esto define un campo de caracteres con una longitud máxima de 100 caracteres. Representa el "nombre"del usuario.
- apellidos = models.CharField Esto define un campo de caracteres con una longitud máxima de 100 caracteres. Representa los .ªpellidos"del usuario.
- contraseña = models.CharField: Esto define un campo de caracteres con una longitud máxima de 20 caracteres. Representa la çontraseña"del usuario.
- Cada campo corresponde a una columna en la tabla de base de datos y tiene un tipo de dato específico asociado.

- def str(self):: Este método está definido para controlar cómo se representan las instancias de la clase data user como cadenas. Cuando imprimes una instancia de esta clase o la muestras en la interfaz de administración de Django, mostrará el nombre y dni del usuario.
- Por ejemplo, si tienes una instancia de data user con nombre configurado como "Johnz dni configurado como "12345678", al llamar a print(instancia) o mostrarla en la interfaz de administración, se mostrará: "John dni: 12345678".
- En resumen, este código define un modelo de Django llamado data user, que representa una tabla de base de datos con columnas para "dni", "nombre", .ªpellidosz çontraseña". El método str está implementado para proporcionar una representación legible para humanos de las instancias de data user. Este modelo se puede utilizar para almacenar y recuperar datos relacionados con usuarios en una aplicación web de Django.

Listing 1: models.py

```python
from django.db import models

# Create your models here.

class data_user(models.Model):
    dni = models.CharField(max_length=8)
    nombre = models.CharField(max_length=100)
    apellidos = models.CharField(max_length=100)
     contrasea  = models.CharField(max_length=20)

    def __str__(self):
        return self.nombre + f" dni: {self.dni}"
```

3.3. forms

- login_form(forms.Form): Esta clase representa un formulario de inicio de sesión. Contiene dos campos: "dniz "password", donde los usuarios ingresarán su número de documento (dni) y contraseña para iniciar sesión.
- dni = forms.CharField(label=, max_length=8, widget=forms.TextInput(attrs='placeholder': 'Ingresa tu DNI', 'data-lpignore': 'true')): Este campo es un campo de texto (CharField) que tiene una etiqueta vacía (label=) para que no se muestre ninguna etiqueta al usuario. Tiene una longitud máxima de 8 caracteres (max_length=8). El widget asociado es un TextInput, que muestra un campo de entrada de texto en el formulario. Además, se le asigna un atributo placeholder con el valor Ïngresa tu DNI", que es un texto de ejemplo que se muestra en el campo antes de que el usuario ingrese su DNI. También tiene el atributo data-lpignore con valor "true", que podría estar relacionado con algún script de terceros o complemento utilizado en la página.
- password = forms.CharField(label=, widget=forms.PasswordInput(attrs='placeholder': 'Introduce tu contraseña', 'data-lpignore': 'true')): Este campo es similar al anterior, pero es para ingresar la contraseña. Tiene un widget de PasswordInput, que oculta el texto ingresado para proteger la privacidad de la contraseña. También tiene un atributo placeholder con el valor Ïntroduce tu contraseña", que muestra un texto de ejemplo en el campo. Al igual que el campo anterior, tiene el atributo data-lpignore con valor "true".
- register: Esta clase representa un formulario de registro. Contiene cuatro campos: "dni", "name", "surnamez "password", donde los usuarios ingresarán su número de documento, nombres, apellidos y una contraseña para crear una cuenta.

- dni : Este campo es similar al campo "dni.en el formulario de inicio de sesión, pero en este caso, tiene el atributo autocomplete con valor .ºff". Esto indica al navegador que no debe autocompletar este campo, lo cual es útil para campos sensibles como el número de documento.
- name : Este campo es para ingresar los nombres del usuario. Tiene un atributo autocomplete con valor .ºff"para evitar el autocompletado del navegador.
- surname: Este campo es para ingresar los apellidos del usuario. También tiene el atributo autocomplete con valor .ºff".
- password: Este campo es para crear una contraseña para la cuenta. Tiene un widget de PasswordInput, y también tiene el atributo autocomplete con valor .ºff"para proteger la privacidad de la contraseña.
- En resumen, este código define dos clases de formularios de Django: login form y register form. Estos formularios se pueden utilizar para crear páginas de inicio de sesión y registro en una aplicación web, y están diseñados para recopilar y validar información como el número de documento, nombres, apellidos y contraseñas de los usuarios. Los atributos placeholder y autocomplete se utilizan para proporcionar sugerencias y mejorar la experiencia del usuario al interactuar con los formularios.

Listing 2: forms.py

```
1  from django import forms
2
3  class login_form(forms.Form):
4      dni = forms.CharField(label="", max_length=8,
           widget=forms.TextInput(attrs={'placeholder': 'Ingresa tu DNI', 'data-lpignore':
           'true'}))
5      password = forms.CharField(label="", widget=forms.PasswordInput(attrs={'placeholder':
           'Introduce tu contrasea', 'data-lpignore': 'true'}))
6
7  class register_form(forms.Form):
8      dni = forms.CharField(label="", max_length=8,
           widget=forms.TextInput(attrs={'placeholder': 'Ingresa tu DNI', 'autocomplete':
           'off'}))
9      name = forms.CharField(label="", max_length=100,
           widget=forms.TextInput(attrs={'placeholder': 'Ingresa tus nombres', 'autocomplete':
           'off'}))
10     surname = forms.CharField(label="", max_length=100,
           widget=forms.TextInput(attrs={'placeholder': 'Ingresa tus apellidos', 'autocomplete':
           'off'}))
11     password = forms.CharField(label="", widget=forms.PasswordInput(attrs={'placeholder':
           'Crea tu contrasea', 'autocomplete': 'off'}))
```

3.4. view

- Este código es parte de un proyecto Django que contiene algunas vistas y funciones relacionadas con el inicio de sesión y el registro de usuarios. A continuación, te explico las funciones y vistas presentes en el código:
- showHome(request): Esta vista renderiza una plantilla llamada "home.html". Esta función parece ser una página de inicio o página principal del sitio web.

- showForm(request): Esta vista renderiza una plantilla llamada "login.htmlz pasa el formulario login_form como contexto. Esta función maneja el proceso de inicio de sesión de los usuarios.
- showRegister(request): Esta vista maneja el proceso de registro de nuevos usuarios. Si la solicitud es una petición "GET", renderiza la plantilla register.htmlz pasa el formulario register_form como contexto. Si la solicitud es una petición "POST", extrae los datos ingresados por el usuario (dni, nombre, apellidos y contraseña) y realiza algunas verificaciones.
- isValid(req, api): Esta función auxiliar recibe dos cadenas de texto como entrada (req y api) y realiza una comparación entre ellas después de eliminar espacios y convertir ambas cadenas a mayúsculas. Se utiliza para verificar si el nombre y apellidos ingresados por el usuario coinciden con los datos obtenidos de una API externa que proporciona información sobre el número de documento (DNI) ingresado.
- El código hace uso de algunos módulos y clases de Django, como render, redirect, HttpResponse, messages, data_user, login_form y register_form. También utiliza la biblioteca urlopen para realizar solicitudes a una API externa y la biblioteca json para manejar las respuestas en formato JSON.
- Sin el contexto completo del proyecto y las plantillas asociadas, es posible que no se pueda obtener una imagen completa de todas las funcionalidades y flujos de la aplicación. Pero, en resumen, parece estar implementando un sistema de inicio de sesión y registro de usuarios utilizando formularios de Django y validaciones adicionales a través de una API externa para el DNI.

Listing 3: views.py

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from .forms import login_form, register_form
from django.contrib import messages
from .models import data_user
from urllib.request import urlopen
import json

# Create your views here.

def showHome(request):
    return render(request, "html/home.html")


def showForm(request):
    return render(request, "html/login.html", {
    'form': login_form
    })

    '''
    if(request.method == 'GET'):
        return render(request, "html/login.html", {
        'form': login_form
        })
    else:

        dni = request.POST['dni']
         contrasea  = request.POST['password']
        found_user = data_user.objects.get(dni=dni)

```

```
        if (contrasea == found_user.contrasea):
            return redirect("showHome")

        return render(request, "html/login.html", {
        'form': login_form
        })
    '''

def showRegister(request):
    if(request.method == 'GET'):
        return render(request, "html/register.html", {
            'form': register_form
        })
    else:
        dni = request.POST['dni']
        nombre = request.POST['name']
        apellidos = request.POST['surname']
         contrasea  = request.POST['password']

        url = "https://dniruc.apisperu.com/api/v1/dni/" + dni +
            "?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImdhYnJpZWxzb3RvY2NveWFAZ21haWwuY29tIn0.BRxE


        response = urlopen(url)
        data = json.loads(response.read())
        nombres = data.get("nombres", "")
        Api_apellidos = data.get("apellidoPaterno","") + data.get("apellidoMaterno","")

        if(isValid(nombre, nombres) and isValid(apellidos, Api_apellidos)):
            messages.success(request, "True")
            data_user.objects.create(dni=dni, nombre=nombre, apellidos=apellidos,
                contrasea=contrasea)
            return redirect(to="showForm")

        else:
            messages.error(request, "False")
            return redirect(to="showRegister")

def isValid(req, api):

    if((req.upper()).replace(" ", "").encode('utf-8') == (api.upper()).replace(" ",
        "").encode('utf-8') ):
        return True
    else:
        return False
```

3.5. quickstart.view

- Este código muestra dos clases de serializadores utilizando la biblioteca rest framework de Django para manejar la serialización y deserialización de objetos en una API web.
- UserSerializer: Esta clase de serializador se utiliza para convertir objetos del modelo User de Django en formato JSON, y viceversa, para su uso en una API web.
- GroupSerializer: Esta clase de serializador se utiliza para convertir objetos del modelo Group de Django en formato JSON, y viceversa, para su uso en una API web.

- Para configurar qué campos del modelo se incluirán en la serialización, se utiliza la clase interna Meta, que especifica el modelo que se está serializando (model) y la lista de campos que se deben incluir en el JSON resultante (fields).
- En el caso del UserSerializer, se incluyen los campos 'url', 'username', 'email' y 'groups' del modelo User, lo que significa que estos campos serán incluidos en el JSON resultante cuando se serialice un objeto User.
- En el caso del GroupSerializer, se incluyen los campos 'url' y 'name' del modelo Group, lo que significa que estos campos serán incluidos en el JSON resultante cuando se serialice un objeto Group.
- Estos serializadores son útiles cuando se desea exponer datos de modelos Django como parte de una API web, lo que permite que los clientes realicen solicitudes HTTP para obtener, crear, actualizar y eliminar objetos del modelo a través de la API.

Listing 4: views.py

```python
from django.contrib.auth.models import User, Group
from rest_framework import viewsets
from rest_framework import permissions
from ApiDjangoProject.quickstart.serializers import UserSerializer, GroupSerializer


class UserViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows users to be viewed or edited.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer
    permission_classes = [permissions.IsAuthenticated]


class GroupViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows groups to be viewed or edited.
    """
    queryset = Group.objects.all()
    serializer_class = GroupSerializer
    permission_classes = [permissions.IsAuthenticated]
```

3.6. serializer

- Este código muestra dos clases de vistas basadas en conjuntos (viewsets) para manejar las operaciones CRUD (crear, leer, actualizar y eliminar) en los modelos User y Group de Django utilizando la biblioteca rest framework.
- UserViewSet: Esta clase de vista define un conjunto de vistas para el modelo User. Específicamente, permite ver y editar objetos User a través de la API web. La clase hereda de viewsets.ModelViewSet, que proporciona las funcionalidades necesarias para realizar operaciones CRUD.
- queryset = User.objects.all().order by('-datejoined'): Esto define el conjunto de objetos User que serán accesibles a través de la API. En este caso, se obtienen todos los objetos User y se los ordena por la fecha de unión (date joined) en orden descendente.

- serializer class = UserSerializer: Esta propiedad define el serializador que se utilizará para convertir los objetos User en formato JSON y viceversa. Utiliza el UserSerializer definido previamente.
- permission classes = [permissions.IsAuthenticated]: Esta propiedad define las clases de permisos requeridas para acceder a las vistas. En este caso, se requiere que el usuario esté autenticado (IsAuthenticated) para acceder a estas vistas, lo que significa que solo los usuarios autenticados tendrán permiso para ver y editar objetos User.
- GroupViewSet: Esta clase de vista define un conjunto de vistas para el modelo Group. Permite ver y editar objetos Group a través de la API web. Al igual que UserViewSet, hereda de viewsets.ModelViewSet y proporciona las funcionalidades CRUD necesarias.
- queryset = Group.objects.all(): Esto define el conjunto de objetos Group que serán accesibles a través de la API. En este caso, se obtienen todos los objetos Group.
- serializer class = GroupSerializer: Esta propiedad define el serializador que se utilizará para convertir los objetos Group en formato JSON y viceversa. Utiliza el GroupSerializer definido previamente.
- permission classes = [permissions.IsAuthenticated]: Al igual que en UserViewSet, se requiere que el usuario esté autenticado para acceder a estas vistas de grupos.
- Estas vistas basadas en conjuntos simplifican en gran medida la creación de una API REST para los modelos User y Group. Con estas configuraciones, las rutas y puntos finales de la API serán generados automáticamente, permitiendo realizar operaciones CRUD en los objetos User y Group a través de los métodos HTTP correspondientes (GET, POST, PUT, DELETE, etc.). Además, se asegura de que solo los usuarios autenticados puedan acceder a estas vistas.

Listing 5: serializers.py

```python
from django.contrib.auth.models import User, Group
from rest_framework import serializers


class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'username', 'email', 'groups']


class GroupSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Group
        fields = ['url', 'name']
```

3.7. urls

- Este fragmento de código muestra cómo se configuran las URLs (rutas) para el enrutamiento en una aplicación Django. La variable `urlpatterns` es una lista de rutas que mapean las URL de la aplicación a las vistas correspondientes que se deben mostrar cuando se accede a esas rutas.
- `from django.urls import path`: Esta línea importa la función `path` del módulo `django.urls`, que se utiliza para definir rutas (URL) dentro de la aplicación Django.

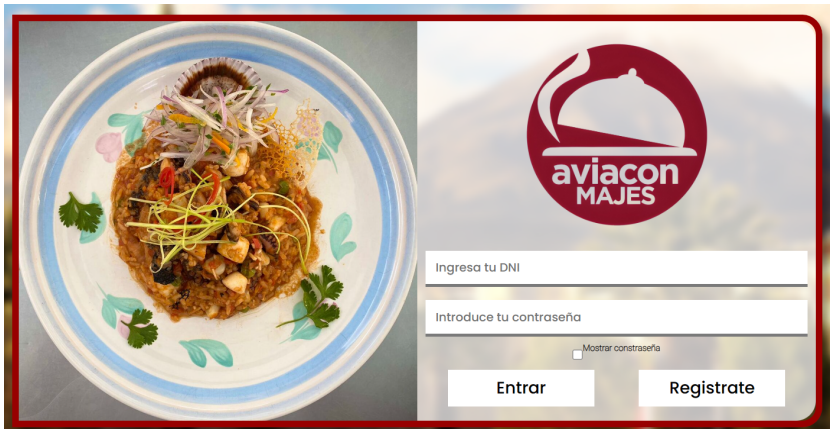
- `from . import views`: Esta línea importa las vistas definidas en el archivo `views.py` del directorio actual (`.`). Las vistas son funciones que manejan las solicitudes HTTP y devuelven las respuestas apropiadas.
- `urlpatterns`: Esta es una lista que contiene todas las rutas definidas para la aplicación Django.
- `path("login/", views.showForm, name="showForm")`: Esta línea define una ruta para la página de inicio de sesión. Cuando un usuario accede a la ruta `/login/` en la aplicación, la función `showForm` dentro del archivo `views.py` será llamada para manejar la solicitud. El parámetro `name="showForm"` se utiliza para darle un nombre a esta ruta, lo cual es útil para referirse a ella de manera más sencilla en otras partes del código.
- `path("home/", views.showHome, name="showHome")`: Esta línea define una ruta para la página principal del sitio web. Cuando un usuario accede a la ruta `/home/`, la función `showHome` dentro del archivo `views.py` será llamada para manejar la solicitud.
- En resumen, el fragmento de código configura las rutas de la aplicación Django de acuerdo con las funciones de vistas definidas en `views.py`. Cuando un usuario ingrese a una ruta específica en el navegador, la función de vista correspondiente se ejecutará para mostrar la página asociada a esa ruta. Estas rutas son útiles para navegar y acceder a diferentes páginas y funcionalidades dentro de la aplicación web.

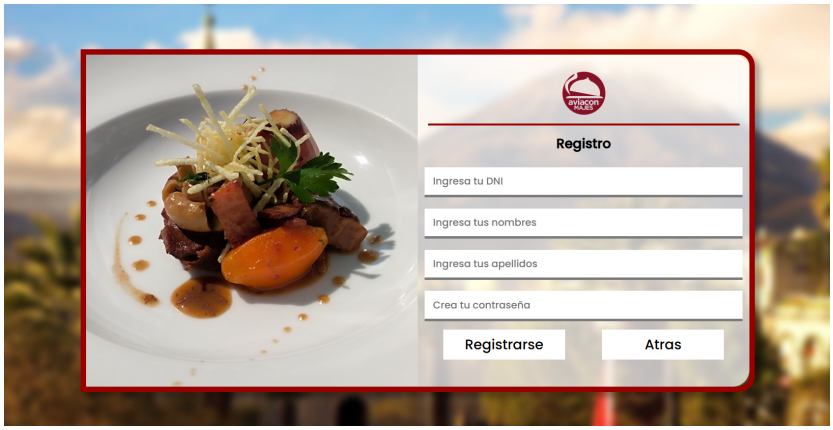
Listing 6: urls.py

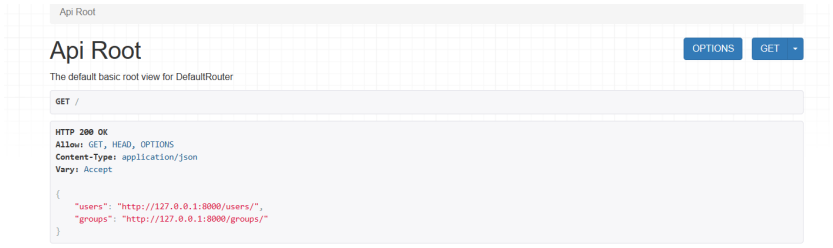
```python
from django.urls import path
from . import views

urlpatterns = [
    path("login/", views.showForm, name="showForm"),
    path("register/", views.showRegister, name="showRegister"),
    path("home/", views.showHome, name="showHome")
]
```

3.8. Capturas del trabajo

-







3.9. Commits del trabajo

- Acá estan algunos de los commits mas importantes en este trabajo:

```
PowerShell 7.3.6
PS C:\Users\Sewas\Desktop\Pweb02_Lab08> git log
commit 1f445d2cd05cb571ec9e5f13b73a7ef008a4a45a (HEAD -> main, origin/main, origin/HEAD)
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 16:35:56 2023 -0500

    Update login.

commit 25f3c2896c59a96289762e7fd30a1d5578b06f44
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 09:49:30 2023 -0500

    Finalizndo el login.

commit 13d425fbf893bcfa59e74ee5cbdc41c8ba9abbb9
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 04:54:12 2023 -0500

    Terminando el frontend de mi login.

commit 21f54e2352e4d7cb6deb62656b4025eed1488463
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 03:17:44 2023 -0500

    Avance de login

commit bb033a45e898594071547f9281554b5295ceea6f
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Mon Jul 24 18:39:30 2023 -0500

    Iniciando el ejercicio 2 de la tarea con GET
:...skipping...
commit 1f445d2cd05cb571ec9e5f13b73a7ef008a4a45a (HEAD -> main, origin/main, origin/HEAD)
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 16:35:56 2023 -0500

    Update login.

commit 25f3c2896c59a96289762e7fd30a1d5578b06f44
Author: GabSoto <gsotocco@unsa.edu.pe>
Date:   Tue Jul 25 09:49:30 2023 -0500
```

3.10. Estructura de laboratorio 08

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab08/
|   .gitignore
|  ApiDjangoProject

\---latex
    | EDA_lab08.pdf
    |   EDA_lab08.tex
    |
    +---img
    |       Commits.png
    |       1.png
    |       2.png
    |       3.png
    |       4.png
    |       logo_abet.png
    |       logo_episunsa.png
    |       logo_unsa.jpg
    |
    \---src
   |     views.py
   |     models.py
   |     forms.py
   |     urls.py
```

4. Rúbricas

4.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

4.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	x	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	x	2	
Total		20		20	

5. Referencias

- https://www.geeksforgeeks.org/introduction-to-avl-tree/
- https://www.geeksforgeeks.org/insertion-in-an-avl-tree/
- https://www.geeksforgeeks.org/deletion-in-an-avl-tree/
- https://docs.oracle.com/javase/tutorial/java/generics/types.html
- https://algorithmtutor.com/Data-Structures/Tree/AVL-Trees/