Evento Evaluativo 2

Presentado por:

Camilo Vargas Parra

ID:816152

Presentado a:

Mgt Eddy J. Pérez Ó.

Sistemas Distribuidos

Universidad Cooperativa de Colombia

Ingeniería de Sistemas

Villavicencio-Meta

Octubre, 2024

Tabla de contenido

1. **Descripción General:** - Proporciona una visión general del sistema, incluyendo el propósito y los objetivos del proyecto.

El proyecto VSystem es un sistema desarrollado para gestionar y organizar diferentes tareas específicas esto permitiendo la interacción entre los usuarios y el manejo de datos, este sistema sigue una arquitectura modular, lo que lo hace escalable y adaptable a futuras necesidades, integrando componentes como lógica de negocios, acceso a datos y presentación de forma desacoplada. El propósito principal de este proyecto es crear una plataforma que funcione de manera eficiente y escalable para la gestión de procesos internos, así mismo facilitando el acceso y la modificación de diferente información de manera segura y rápida.

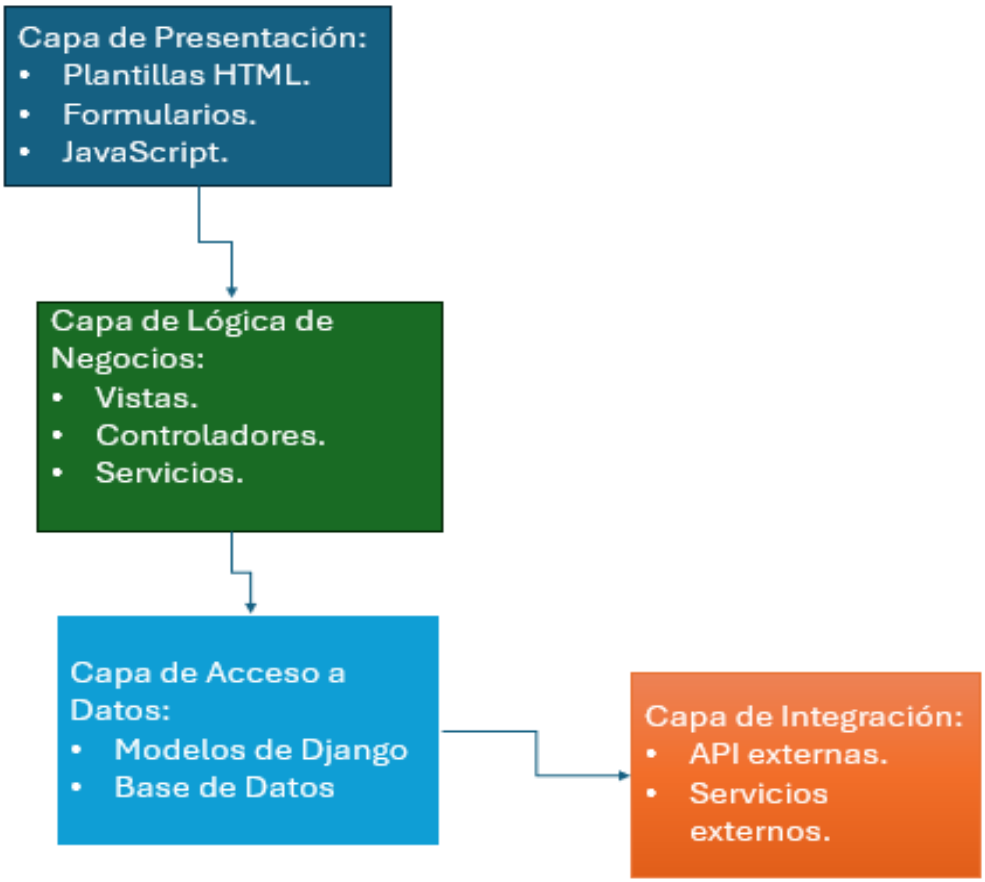
Como objetivos del proyecto tenemos:

Permitir que los usuarios puedan organizar, modificar o consultar información en un solo programa, así mismo permitiendo el acceso y la administración de datos.

He de asegurar que dicho sistema pueda adaptarse y crecer conforme aumenten los volúmenes de datos y usuarios, manteniendo un rendimiento optimo y sin fallas.

Facilitar una conexión con servicios de terceros para poder ampliar las funcionalidades del sistema, esto permitiendo una mayor operabilidad y personalización.

2. **Diagrama de Arquitectura:** - Crea un diagrama que ilustre las diferentes capas de la arquitectura y cómo interactúan entre sí. Puedes usar herramientas como Lucidchart, draw.io, o incluso diagramas simples en PowerPoint.



3. **Descripción de las Capas:** - Capa de Presentación: Describe los componentes que forman la interfaz de usuario, como vistas y templates. - Capa de Lógica de Negocios: Explica los servicios y lógica que maneja las reglas de negocio del proyecto. - Capa de Acceso a Datos: Detalla cómo se manejan las interacciones con la base de datos, incluyendo modelos y ORM. - Capa de Integración: Describe cualquier sistema externo con el que el proyecto se integre, como APIs de terceros.

1. Capa de presentación:

Esta capa ha sido la responsable de la interfaz de usuario y la forma en que los usuarios interactúan con el sistema, dentro del código del proyecto los componentes principales que forman la capa de presentación son:

Plantillas: Los archivos HTML que generan la interfaz visual para el usuario. En el directorio templates/, encontrará los archivos HTML que representan las páginas web. Estas plantillas contienen el diseño de las vistas, como formularios de entrada de datos, listas de registros o páginas de detalle.

Vistas: Las funciones o clases en views.py que manejan las solicitudes HTTP (GET, POST) y devuelven las respuestas, incluyendo la renderización de plantillas.

2. Capa de Lógica de Negocios:

Esta capa contiene la lógica que gobierna el comportamiento del sistema, incluyendo la gestión de reglas de negocio validaciones y manipulación de datos, los principales componentes de esta capa son:

Vistas (Views): Las vistas también juegan un rol importante aquí ya que coordinan cómo se manejan las peticiones y los datos que se presentan al usuario, Django permite definir vistas basadas en funciones o clases, ya que estas vistas pueden tener lógica para validar formularios, aplicar reglas de negocio o interactuar con la base de datos.

Formularios: Son parte de la lógica de negocio ya que definen qué datos espera el sistema del usuario y cómo deben ser validados, Django proporciona una herramienta de llamada forms.py para definir y gestionar formularios.

3. Capa de acceso a datos:

La capa de acceso a datos maneja todas las interacciones con la base de datos, donde los principales componentes son los modelos y el ORM (Object Relational Mapping) de Django, que traduce el código Python a consultas SQL.

Modelos: En Django los modelos representan tablas en la base de datos, ya que estos definen la estructura de los datos y las relaciones entre ellos, estos modelos se encuentran en models.py.

Consultas con el ORM: A través del ORM se pueden hacer las consultas sobre la base de datos sin necesidad de escribir SQL, ya que estas consultas se encuentran a menudo en las vistas o servicios.

4. Capa de Integración:

La capa de integración maneja la comunicación con sistemas externos como lo son las APIs de terceros, servicios web o cualquier otra fuente de datos externa, si el proyecto tiene alguna integración con APIs externas, sería en esta capa donde se gestionarán las conexiones, las solicitudes HTTP y el procesamiento de las respuestas.

Uso de API externas: Si el proyecto necesita conectarse a API externas, se utiliza la librería requests de Python para hacer solicitudes HTTP, o bien el propio soporte que ofrece Django para gestionar estas conexiones.

4. **Tecnologías y Herramientas:** - Lista las tecnologías y herramientas utilizadas en cada capa. Por ejemplo, Django para la lógica de negocio, HTML/CSS/JavaScript para la presentación, y Django ORM para la capa de acceso a datos.

Capa de presentación:

HTML: Se utilizo para estructurar las páginas web que son mostradas al usuario.

CSS: Fue el responsable de los diseños y las apariencias de las páginas.

JavaScript: Se uso para agregar interactividad a las páginas y mejorar las experiencias a los usuarios.

Plantillas Django: Se empleo para generar vistas dinámicas que permiten inyectar contenido de manera dinámica en el HTML a partir de las diferentes vistas.

Capa de lógica de Negocios:

Django: Es el framework web principal que maneja la lógica de negocio, esto proporcionando la estructura para la gestión de solicitudes http, las validaciones y coordinación entre la capa de presentación y la base de datos.

Python: Fue el lenguaje de programación utilizado en el backend, esto aprovechando para definir la logica del sistema y manejar las operaciones del servidor.

Capa de acceso a Datos:

Django ORM: El ORM de Django permite gestionar las interacciones entre los modelos y la base de datos, esto permitiendo la facilitación de poder realizar las diferentes consultas sin necesidad de escribir SQL.

PostgreSQL/MySQL: Fue el sistema de gestión de bases de datos que podrían estar integrados con el ORM de Django para almacenar y recuperar los datos.

Capa de Integración:

API externas: Para la continua comunicación con servicios externos que complementan la funcionalidad del sistema.

Solicitudes (Python): Se utiliza para realizar solicitudes HTTP cuando es necesario interactuar con API de terceros.

5. **Patrones y Principios:** - Documenta cualquier patrón de diseño o principio arquitectónico utilizado, como MVC (Modelo-Vista-Controlador) o DRY (Don't Repeat Yourself).

En el desarrollo del proyecto se aplicaron los diferentes patrones de diseño y principios arquitectónicos para garantizar que el sistema sea eficiente y mantenible, se describen de la siguiente manera. La separación de responsabilidades se hace para facilitar el mantenimiento y escalabilidad del sistema, ya que esto permite modificar un parte sin afectar a las demás:

1. **Patrón MVC (Modelo-Vista-Controlador):**

El sistema sigue el patrón arquitectónico MVC, en donde las distintas responsabilidades se separan.

Modelo: Los modelos de Django son representados en las tablas de la base de datos y donde manejan la logica de acceso y la manipulación de los datos, ya que este es el equivalente a la capa de acceso a datos en el sistema.

Vista: Las vistas en Django manejan las solicitudes HTTP donde se determinan que datos se muestran en las plantillas, actuando como un puente entre los modelos y plantillas siendo este el controlador en términos del patrón MVC.

Plantillas (templates): representa la capa de presentación, donde es definido como los datos se muestran al usuario.

2. **DRY (No te repitas):**

El principio DRY se aplicó para evitar la duplicación del código, en lugar de repetir la misma logica o funcionalidad en diferentes partes del sistema se crean las funciones que sean reutilizables, ya que esto simplifica el mantenimiento del código y reduce los posibles errores potenciales.

3. **KISS (Mantenlo simple):**

El principio KISS se siguió para llevar a cabo la simplicidad en el código, el sistema está diseñado lo más simple y entendible posible, sin agregar una complejidad que sea innecesaria, esto se ve reflejado en la forma de las vistas y modelos estructurados para poder asegurar que las soluciones sean directas y fáciles de entender.

4. **Inversión de Control (IoC) e Inyección de Dependencia:**

Aunque Django no sigue estrictamente la inyección de dependencias, sí utiliza el concepto de Inversión de Control (IoC) al permitir que el framework maneje aspectos como la gestión de la base de datos, el enrutamiento y la seguridad, delegando esas responsabilidades al framework en lugar de implementarlas manualmente en el código.

6. **Ejemplos de Flujo de Trabajo:** - Proporciona tres ejemplos de un flujo de trabajo típico, describiendo cómo los datos se mueven a través de las diferentes capas.

Flujo de Trabajo 1: Registro de un nuevo usuario

Capa de presentación:

El acceder a un formulario de registro en la interfaz web para ingresar sus datos (nombre, correo electrónico, contraseña, etc.). Este formulario es generado por Django Templates en HTML, después de completar el formulario, el usuario envía la solicitud (POST) al servidor.

Capa de lógica de negocios:

La vista correspondiente en Django recibe los datos enviados. Aquí se valida la información del formulario mediante un formulario Django (forms.py) para asegurar que los datos cumplan con los requisitos, si la validación es exitosa se crea un nuevo objeto de usuario.

Capa de acceso a datos:

Una vez validado, el formulario utiliza el ORM de Django para crear un nuevo registro en la base de datos (modelo `User`). El ORM traduce la solicitud en una operación SQL y guarda los datos en la base de datos.

Capa de presentación:

El sistema redirige al usuario a una página de confirmación de su perfil, mostrando los datos que acaba de registrar.

Flujo de Trabajo 2: Autenticación de un usuario (Iniciar sesión)

Capa de presentación:

Se introduce su nombre de usuario y contraseña en un formulario de usuario de inicio de sesión generado en HTML por Django Templates, se envía la solicitud (POST) para autenticar los datos.

Capa de lógica de negocios:

La vista correspondiente recibe los datos y utiliza el sistema de autenticación de Django (authenticate) para verificar las credenciales, si el nombre de usuario y la contraseña son correctos, el sistema crea una sesión para el usuario mediante la función login () de Django.

Capa de acceso a datos:

El sistema utiliza el ORM de Django para verificar las credenciales contra la base de datos de usuarios registrados.

Capa de presentación:

Si las credenciales son correctas, el usuario es redirigido a su panel de control o página de inicio personalizada. Si no lo son, se muestra un mensaje de error en la página de inicio de sesión.

Flujo de Trabajo 3: Consulta de datos (Listado de usuarios)

Capa de presentación:

Un administrador accede a la página que muestra la lista de usuarios registrados. La solicitud GET se envía desde la interfaz web (HTML), la vista mostrará una tabla con los usuarios disponibles, estructurada con Django Templates.

Capa de lógica de negocios:

La vista correspondiente procesa la solicitud GET y realiza una consulta al modelo User para obtener los datos de todos los usuarios, los datos obtenidos se pasan al template para ser renderizados en el navegador.

Capa de acceso a datos:

Django ORM interactúa con la base de datos ejecutando una consulta SQL para obtener todos los registros de usuarios, estos datos se estructuran en objetos y se devuelven a la vista para su procesamiento.

Capa de presentación:

La lista de usuarios se muestra en una tabla HTML permitiendo al administrador ver y gestionar los usuarios directamente desde la interfaz del sistema.