Lisis Mariela Elvir Estrada Número de cuenta: 20201001503
José Eduardo Reyes Benites Número de cuenta: 20181032486
Diana Milena Martinez Banegas Número de cuenta: 20201002442

Informe sobre el Proyecto Final de Programación Orientada a Objetos

INTRODUCCION

En el presente informe se dará a conocer el proceso de creación de una aplicación web llamada Fastery, en el cual se documentará desde el inicio, donde se obtiene la idea inicial, hasta que se logra ejecutar los archivos sin problemas, dado que los mismos habrán sido resueltos,y se encuentre listo para su presentación en la clase de POO (Programación Orientada a Objetos).

A través de este proyecto, se busca optimizar el proceso de compra y entrega de productos, integrando funcionalidades que permiten realizar pedidos, seleccionar platillos, registro de usuarios, inicio y cierre de sesión y acceso a observar el historial de pedidos.

La aplicación se construyó siguiendo los principios fundamentales de la POO, como la herencia y el polimorfismo, lo que permitió estructurar el código de manera modular, re utilizable y fácil de mantener. Fastery incluye varios componentes esenciales, como la base de datos para la gestión de usuarios, pedidos y manejo de datos.

A lo largo de este informe se detallará el proceso de desarrollo de la aplicación, las tecnologías utilizadas, y las soluciones implementadas para asegurar que Fastery cumpla con los requisitos funcionales establecidos, garantizando la satisfacción de los usuarios y la eficiencia del servicio de delivery.

así mismo implementamos la seguridad que ya nos brindan las tecnologías utilizadas para la realización de este proyecto.

Lisis Mariela Elvir Estrada Número de cuenta: 20201001503
José Eduardo Reyes Benites Número de cuenta: 20181032486
Diana Milena Martinez Banegas Número de cuenta: 20201002442

metodología

Para la elaboración del proyecto, se tuvo que realizar la investigación de varias tecnologías para poder implementar ciertas funcionalidades.

Para el back-end:

1. **Python:** lenguaje de programación de tipado dinámico que nos permite trabajar con facilidad, por ende se nos asignó como base de nuestro proyecto.
2. **Django:** Es un framework escrito en el lenguaje de programación Python, para el manejo de back-end y conexión a una base de datos, utilizamos librerías de Django para mejorar la calidad de nuestro proyecto.
3. **Pillow:** es una biblioteca adicional gratuita y de código abierto para el lenguaje de programación Python que agrega soporte para abrir, manipular y guardar muchos formatos de archivo de imagen diferentes.
4. **Base de datos SQLLite:** es la base de datos de que esta conectada por defecto al proyecto creado, se utiliza para almacenar información del proyecto y de los usuarios.

Para Front-end:

1. **HTML:** lenguaje de marcado utilizado en la creación de páginas web, se utilizo para la estructuración principal de las páginas web del proyecto.
2. **CSS:** es un lenguaje informático especializado en definir y cohesionar la presentación de un documento escrito en un lenguaje de marcado como HTML, lo utilizamos para proveer un estilo a nuestra aplicación web.

3. **BOOTSTRAP:** es un framework multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web, se utilizaron las plantillas para implementar algunas mejoras a los archivos.
4. **Jinja2:** es un motor de plantillas orientado a Python, se utilizó para vincular los datos de SQLLITE a HTML.

Sistema de control de versiones y tele-trabajo:

1. **GIT:** sistema de control de versiones y ramificación de código para la realización de avances. Se utilizo en la totalidad del proyecto hasta llegar a la versión de entrega.
2. **GITHUB:** Es una plataforma donde se puede almacenar, compartir y trabajar junto con otros usuarios para escribir código, se utilizó en la totalidad del proyecto para distribuir el trabajo entre los integrantes del grupo.
3. **WHATSAPP:** Red social de mensajería y llamadas utilizada para la realización de reuniones y avances en vivo del proyecto.

Editores de Código:

1. **PYCHARM:** editor de código en python, para la modificación de todos los archivos .py.
2. **VISUAL STUDIO CODE:** editor de código orientado para distintas extenciones y lenguajes de programación, utilizado para trabajar los archivos HTML, CSS, PY, ETC.
3. **CMD:** terminal integrada en los sistemas operativos utilizada para la creación del entorno virtual y el ingreso de distintos comandos.

Lisis Mariela Elvir Estrada Número de cuenta: 20201001503
José Eduardo Reyes Benites Número de cuenta: 20181032486
Diana Milena Martinez Banegas Número de cuenta: 20201002442

DESARROLLO

Requerimientos y diseño: Se nos solicitó la creación de una aplicación web donde los usuarios puedan realizar un pedido a domicilio de un producto a un restaurante, y que se le asignará la tarea a un repartidor.

Sobre el diseño buscamos realizar una interfaz simple, sencilla y amigable al usuario, que fuese fácilmente escalable, y agradable a la vista en general.

Comenzamos realizando un diagrama UML para las clases, dentro del cual identificamos como manejar los datos para poder trabajar ordenadamente y realizar las diferentes funciones.

Luego realizamos un Wireframe, que nos sirvió de mapa para organizar nuestros templates.

A continuación se realizó un Mockup tomando de base el Wireframe y colocándole imágenes, colores, y ejemplos para poder hacernos a la idea de como sería el producto final.

Como primer paso se realizó la creación de la carpeta del proyecto, dentro de la misma se creó un entorno virtual, donde se instalaron la diferentes dependencias, siendo estas: Python 3 y Django.

Nos reunimos de manera virtual para decidir, datos importantes de la aplicación web, se determinó el nombre como Fastery, siendo este la mezcla de Fast que significa rápido, y Delivery que significa entrega a domicilio, así como unas frases de Slogan, entre otras sugerencias.

Una vez tuvimos el nombre de la aplicación web procedimos a crear la aplicación llamada servicio, y continuamos con la inclusion de la aplicación en el archivo settings.py.

EXPLICACIÓN DE LOS ARCHIVOS.PY

Models.py: se creó la clase abstracta Raíz con los atributos nombre, dirección y teléfono. Luego se creó la clase Cliente, que hereda de Raíz los tres atributos, y se le agregan atributos propios a la clase siendo estos apellido y clave. A continuación creamos la clase Repartidor, que también hereda de Raíz, y se le agregan los atributos de marca, serie y placa de la moto del repartidor como atributos propios de la clase, se le agrego también el atributo de tipo BooleanField, esto para manejar si un repartidos esta disponible o no.

Seguido de esto,creamos la clase Restaurante, que también hereda de raíz, y se le agregan atributos propios, la descripción del restaurante y el logo, siendo este ultimo de tipo ImageField (en el archivo settings.py importamos os, definimos media_URL que se utiliza para definir la ruta donde se insertarían las imágenes, se definió el media_Root, el cual es una configuración que define donde se almacenarán los archivos, se procedió a crear manualmente la carpeta de imágenes dentro del proyecto para mantener el orden) siendo el primer parámetro, upload_to='logos' siendo este el

encargado de la creación de una carpeta donde se almacenaran todas las imágenes que se suban a esta clase.

Tuvimos que crear una clase sin herencia llamada Plato, la cual tiene los atributos: nombre del plato, detalles, valor, imagen, siendo este campo ImageField, que sigue la lógica del atributo logo de la clase Restaurante, pero llevando las imágenes de esta clase a la carpeta 'platos', y el atributo restaurantes tipo ForeignKey, esta funcionalidad crea la conexión entre clases, y une la clase Plato con la clase Restaurante y por defecto se le asigna el primer restaurante según el ID que esta en la base de datos. Creamos la clase Cliente que hereda de Raíz, y se le agregan los atributos propios apellido, plato, siendo este un ForegnField que hace conexión con la clase Plato, y el atributo usuario, siendo este también un ForeignField que realiza la conexión, con User que viene de la librería que brinda Django contrib.auth, en el archivo admin.py se importaron todas las clases del models a excepción de la clase abstracta, todo esto con el fin de poder realizar exitosamente las migraciones para crear las tablas en la base de datos y poder ingresar manualmente los datos desde el admin que Django genera.

Views.py: para comenzar, se creó la función inicio esta función lo único que renderiza es la pagina inicio.html, se le definió su URL en el archivo urls.py se definió como la pagina de inicio.

Se creó la función registrar, para poder hacer esta función tuvimos que importar de **Django from Django.contrib.auth.forms import UserCreationForm** , AuthenticationForm, también se importó Rom **Django.contrib.auth.models import User, from Django.contrib.auth import login, logout , authenticate** primero se valida si el método que recibe la url es el método GET, renderice la pagina del

formulario, se colocan como parámetros el formulario con UserCreationsForm, y se pasa al HTML para generar el formulario por defecto de Django. Caso contrario, si los datos pasan por el método POST, primero se validará que las dos claves coincidan, se realiza un control de excepciones, dentro de las ellas se realiza la creación del usuario, se guarda con el método save en la base de datos, y con el método login se inicia sesión a ese usuario y se redirecciona a la pagina de inicio. En caso de faltar una excepción, se va a renderizar la página de formulario, se enviará el formulario de Django junto con un mensaje de error al HTML. Como segundo, en caso de que las contraseñas no coincidan, pasará lo mismo que en el anterior caso pero con un mensaje diferente, se creó su URL en el archivo urls.py con el nombre registrar.

Se creó la función res_list que es la lista de los restaurantes, primero se crea el objeto res, de la clase restaurante y se le asignan todos los restaurantes que estén en la base de datos, de ahí renderiza el HTML restaurantes con el objeto res como parámetro, se creó su url llamada restaurantes. A continuación hicimos la clase menu, primero se creó el objeto comida, de la clase Plato, y se filtran los platos que tengan la relación con el restaurante al cual pertenece el ID, de ahí se crea el objeto res de la clase Restaurante y se filtra por su ID y se obtiene mediante el método get, se renderiza el HTML producto junto con el diccionario que contiene el objeto comida y res, y se creó su url, llamada productos, dicha url pide un parámetro adicional de tipo Int que sería el ID.

también agregamos una función llamada no_implementada, la cual tiene de objetivo que en caso de que se esté creando una nueva funcionalidad renderice el HTML con el mensaje “esta función no está implementada.” para que el usuario no vea directamente un mensaje de error en su pantalla, se creó su url llamada no_found.

Se realizó la creación de la función pedido, la función recibe el ID como parámetro, se crea el objeto orden de la clase Plato, se filtra por ID, y se toma usando un método get, de ahí renderiza la pagina confirmar_pedido junto con el objeto orden. Se creó su url pedido, la cual recibe un parámetro de tipo int llamado id.

Se creó la función cerrar_sesión usando el método logout para cerrar la sesión del usuario y se redirecciona a la página de inicio, se creó su url llamada cerrar. Se creó la función iniciar_sesión, se valida si se accede a la página usando el método GET, en caso de acceder por ahí, se renderiza el HTML junto con el formulario de autenticación, caso contrario, si se accede por el método POST, se autentica el usuario. Con el siguiente condicional validamos si el usuario existe, caso de no existir, se renderiza la página de inicio de sesión, junto al formulario de iniciar secion y un mensaje de error, caso contrario, usando el método login se inicia sesión del usuario, y se redirecciona a la pagina de inicio, con la sesión iniciada, su url se llama iniciar.

Se importa un decorador propio de Django, llamado @login_required, el cual hace que la función requiera que el usuario tenga una sesión iniciada, se crea la función historial, se crea el objeto pedidos de la clase Cliente, filtrado por usuario, gracias a la ForeignKey del cliente con los user y se renderiza el historial.html junto con el objeto pedidos. El url se llama historial. Usando ese mismo decorador, se crea la función datos_entrega, esta función recibe el parámetro id, se crea el objeto plato,de la clase Plato, y usando el método get, se asigna por el id, se crea el objeto repartir de la clase Repartidor y se filtra por su estado en la base de datos, el cual en caso de ser True, se asigna ese repartidor al objeto repartir, en caso de que no hayan repartidores disponibles lanzara un mensaje HttpResponse que dice que no hay repartidores disponibles, en caso de que si hayan repartidores disponibles, a la variable asignado,

se le asignan los datos del primer repartidor,que encuentre disponible usando el método first, en caso de que los datos vengan del HTML por el método POST, a la variable entrega se le asignarán los datos de la clase, DatosGenerales con el parámetro request.POST. Para hacer posible el uso de esta clase se importó desde el archivo **formulario.py** la clase DatosGenerales, en el archivo formulario.py, importamos **from Django import forms** y del archivo models la clase Cliente **from .models import Cliente,** creamos la clase DatosGenerales que hereda de forms.ModelForm, dentro de la clase se crea la clase Meta, y se asigna el modelo que seria cliente para crear el formulario con los campos que tenga la clase cliente, de eso se encarga la variable fields, usamos el elemento help_texts que se encarga de eliminar los textos de ayuda del formulario, con fines estéticos. Se creó la función __init__ con los parámetros (self, *args, **kwargs): con el fin de eliminar los mensajes de requerimiento del formulario también con el fin de una mejor estética., continuando con la función **datos_entrega**, verifica primero con un condicional si los datos recibidos son válidos, en caso de que sea valido el objeto **asignado** cambia de estado a False, se guardan los cambios con el método save en la base de datos, esto para que se muestre al repartidos como ocupado, a la variable historial, se le asignan los datos de la variable entrega y se guardan con un commit=False, esto con el fin de que se guarden temporalmente los datos y no se suban instantáneamente a la base de datos, en historial.usuario, se almacenan los datos del user, en historial,plato almacena los datos del objeto plato, estas dos ultimas lineas son posibles gracias a la conexión del ForeignKey, se guardan los datos directamente en la base de datos, se renderiza el HTML realizado.html con los objetos **plato** y **asignado.** En caso de que los datos lleguen por el método GET, se almacenaran en la variable entrega y se renderizará el HTML DatosEntrega con los objetos, **entrega** y **plato**. Se creó la función

información, que renderiza un HTML llamado nosotros, el cual muestra los datos de los estudiantes que realizaron este proyecto.

LOS TEMPLATES:

Base.html: a este archivo se le instalaron los scripts de Bootstrap este documento contiene una navbar con los siguientes botones, el de cerrar sesión que redirige a la url cerrar , para finalizar la sesión. El botón historial que redirige a la url llamada historial. El botón de registro que redirige al formulario de registro, el botón de iniciar sesión, que redirige al formulario de iniciar sesión, y por ultimo un botón llamado Acerca de nosotros que redirige a la información de los estudiantes a cargo del proyecto.

Todos estos botones, cada uno de sus urls están instanciados con la sintaxis de Jinja. estos botones están sujetos a condicionales, con un condicional de Jinja si el usuario esta autenticado se mostraran los botones de cerrar sesión e historial, en caso de que no, se mostraran los botones de registro, e iniciar sesión, el botón de Acerca de nosotros aparecerá en ambos casos, para finalizar están los respectivos block content y endblock para que sea posible heredar este template a otros templates en este proyecto.

DatosEntrega: en este archivo el primer item esencial es el {% csrf_token %} y usando la sintaxis de Jinja traemos el formulario llamado entrega a este archivo HTML y se le asigna el botón de confirmar los datos para cerrar el formulario, este formulario se genera en la función DatosEntrega del archivo **views.py.**

formulario_registro: en este archivo el primer item esencial es el {% csrf_token %} y usando la sintaxis de Jinja traemos el formulario llamado formulario a este archivo HTML y se le asigna el botón de guardar los datos y tiene un condicional para manejar los errores, los mensajes de error y el formulario se generan en la función **registrar** del archivo **views.py.** para cerrar el formulario, utilizamos el botón guardar**.**

inicio_sesión: en este archivo el primer item esencial es el {% csrf_token %} y usando la sintaxis de Jinja traemos el formulario llamado sesión a este archivo HTML y se le asigna el botón de guardar los datos y tiene un condicional para manejar los errores, los mensajes de error y el formulario se generan en la función **iniciar_sesión** del archivo **views.py** para cerrar el formulario, utilizamos el botón de iniciar sesión.

Confirmar_pedido: primero importamos el Base.html usando extends, se abre el block content y lo cerramos antes del finalizar el body con un endblock, dentro del block content usamos el objeto orden para mostrar los diferentes datos del producto junto con su imagen y el botón contiene una url que lo redirige a datos. Todos los datos vienen de la función pedido del archivo **views.py**

Realizado.html: se importan los enlaces de Bootrstrap, a partir de ahí usando los objetos plato y repartir, mostramos los diferentes datos del producto, como del repartidor, esto sale de la función DatosEntrega del archivo **views,py.**

productos: se importa Base.html, se abre el Block content y adentro del primer card va el logo del restaurante usando el objeto res y los datos generales. En el menu se inicia un bucle for donde se almacenan los datos en la variable food, se muestran los distintos datos de los platillos junto con sus imágenes en los botones hay una validación, si el usuario esta autenticado, el botón ordenar lo redirige a la url pedido,

caso contrario, se le enviará a la url donde se muestra el formulario de inicio de sesión, se cierra el for y se coloca el endblock para cerrar el block content. Estos datos se obtienen de la función **menu** del archivo **views,py.**

restaurantes: se importa Base.html, se abre el Block content, se inicia un bucle for iterando sore el objeto res, almacenando los datos en la variable "i", se muestra el logo de cada restaurante, junto con sus datos, el botón de menu del restaurante, redirige a la url productos, mostrando los productos de este restaurante, se cierra el for y se coloca el endblock para cerrar el block content. Estos datos se obtienen de la función **res_list** del archivo **views,py.**

Historial.html: : se importa Base.html, se abre el Block content, se inicia un condicional para verificar si hay pedidos realizados por el cliente, en casi de que sea True, se inicia un bucle for que va ha recorrer el objeto pedidos y va a almacenarlos en la variable pedido, se muestra la imagen de cada pedido junto con sus datos, incluyendo el nombre del cliente y la dirección de entrega, se cierra el for, en el else, en caso de que no hayan pedidos se mostrará un mensaje de historial vacío. Se cierra el if, y se coloca el endblock para cerrar el block content. Estos datos se obtienen de la función **historial** del archivo **views,py.**

PROBLEMATICAS Y SUS SOLUCIONES:

Dificultades presentadas en la elaboración del proyecto
al comenzar a trabajar en este proyecto se recurrió a muchas investigaciones para entender completamente el funcionamiento del framework Django, a continuación se detallaran las complicaciones mas destacadas del trabajo y aprendizaje de este framework.

- La primera dificultad presentada fue la implementación de la clase abstracta para aplicar herencia de una forma correcta, normalmente al crear una clase abstracta se importa la librería ABC y se pasa como parámetro de la clase pero en Django es diferente, para poder definir la clase Raíz como clase abstracta se

crea una subclase dentro, la cual con el atributo abstract definido como True, esto trajo otra complicación ya que a este punto ya estaba realizada una migración a la base de datos, por lo que recurrimos a borrar la base de datos e iniciar de cero las migraciones practica que es incorrecta pero en ese momento fue la única solución ya que no sabíamos como revertir las migraciones

- En la revisión de los avances del proyecto el Ingeniero nos hizo algunas recomendaciones la primera era que la primera letra de cada clase siempre sea en mayúscula para así diferenciar una clase de una variable, la segunda era no mezclar ingles y español para los nombres de las variables ambas correcciones se hicieron y se realizaron las migraciones respectivas para actualizar las tablas de la base de datos.
- Otra observación que nos hizo en la revisión fue respecto a la forma en que se mostraban los datos de los platillos en los archivos HTML ya que la función correspondiente en el views (productos) lo que hacia era recibir de parámetro el nombre del restaurante y en el HTML con un condicional de Jinja filtraba los platillos y mostraba solo los que coincidieran con el dato del nombre del restaurante, la forma correcta de hacer esto era mediante el uso de las ForeignKey estas se encargan de hacer las relaciones en la base de datos mediante el id, al inicio fue un poco complejo de entender pero con lo que nos explico el ingeniero y las investigaciones personales pudimos implementarlas de forma exitosa y así el código ser mas eficiente.
- Otro inconveniente fue al tratar de implementar los formularios para registro e inicio de sesión, al hacer un formulario personalizado el resultado lo que daba era un formulario para guardar datos no encriptada claves ni hacia validación de usuarios, de allí encontramos información sobre los formularios propios de Django el inconveniente era que daban permisos de administrador a cualquier usuario que se registrara, se siguió investigando asta encontrar la forma correcta de realizar el formulario con toda la seguridad y validaciones de Django se estilizaron y se les eliminaron los textos de ayuda que vienen por defecto.
- La ultima funcionalidad que se implemento fue una forma de hacer que los repartidores tengan un estado y cuando sea asignado a una entrega a esto se agrego un BooleanField True seria disponible False seria ocupado, el inconveniente era cambiarle el estado al repartidor y de allí guardar la actualización de los datos en la base de datos, cuando tomaba un repartidor y le cambiaba el estado repetía el proceso y le cambiaba el estado a los demás, esto hacia una operación incorrecta, después de 2 horas haciendo cambios y probando se llego a la solución usando el método first() se asigna el primer repartidor que se encuentre disponible y se hace el cambio de estado dentro de un condicional que valida la entrada de los datos del pedido y se guardan las actualizaciones para luego mostrar los datos del pedido realizado junto con los datos del repartidor.

Lisis Mariela Elvir Estrada Número de cuenta: 20201001503
José Eduardo Reyes Benites Número de cuenta: 20181032486
Diana Milena Martinez Banegas Número de cuenta: 20201002442

CONCLUSIÓN

Fue necesario dedicar tiempo, recursos, investigación y sobretodo empeño a la realización de este proyecto, en el transcurso del desarrollo de nuestra aplicación web, comprendimos el proceso de trabajar en un proyecto real para un cliente, fue todo un reto que consideramos lo bastante difícil como para tomarlo con la seriedad necesaria.

Ha sido una experiencia muy grata y enriquecedora para nosotros la ejecución del trabajo en equipo, así como el trabajo individual y la responsabilidad que eso conlleva, así como trabajar en base a los requerimientos del cliente, trabajar con fechas limite, y la realización de correcciones en tiempo real de problemas que aparecían o la implementación de nuevas herramienta que facilitaban el trabajo ya hecho, y la creatividad para dar una imagen agradable a la vista del usuario.

Consideramos que el trabajo que se entrega cumple con lo solicitado, pero creemos que hay una infinidad de posibilidades que podrían implementarse para mejorar y pulir, hasta obtener algo mejor, dado el tiempo y las investigaciones invertidas, consideramos que tuvimos un crecimiento constante, un aprendizaje enriquecedor y sobretodo, que escalamos un pequeño peldaño para llegar a ahondar mas en el conocimiento que hay en los cimientos de un ingeniero en sistemas, es por ello que nos sentimos agradecidos por la experiencia.

ANEXO:

https://github.com/DevEduardoReyes/proyecto

https://www.youtube.com/watch?v=v7K-PbX3Lxg