**Master: Ingeniería de Software**

**Asignatura:** Desarrollo de Software

**Apellidos: Gil**

**Nombre: Jonas**

|  |
| --- |
| **Declaración personal de no plagio** |
| **1. Tengo conocimiento de que plagiar supone usar el trabajo de otro y presentarlo como propio, y de que constituye una infracción de los derechos de propiedad intelectual.**  **2. Declaro que lo que aquí presento es fruto mi propio trabajo.**  **3. No he permitido, y no permitiré, que nadie copie mi trabajo con la intención de hacerlo pasar como su propio trabajo.** |

**Tarea a realizar:**

Realizar y entregar en PRADO una propuesta simple de diseño arquitectónico para un sistema, real o ficticio, que el estudiante escoja siguiendo la propuesta que se ha presentado en clase acerca de realizar el diseño arquitectónico de un sistema software basado en componentes (en particular la propuesta se describe en las diapositivas 27-41). Los puntos más importantes a considerar en relación con los contenidos a incluir en la propuesta son los siguientes:

1. Se pueden incluir uno o varios diagramas que describan las partes más importantes de la arquitectura del sistema. Dichos diagramas serán bocetos en los cuales el propio estudiante decidirá la notación a utilizar para los diferentes elementos (cajas con diferentes formas y/o colores, líneas de diferentes estilos, etc), pero incluyendo una leyenda que identifique a que corresponde cada tipo de elemento utilizado. Es decir, la idea es no utilizar notaciones como UML ya que pueden ser más complejas y tienden a incluir más detalles.
2. Los diagramas pueden describir diferentes vistas del sistema que fundamentalmente serían las siguientes (pero el estudiante puede decidir libremente si otro tipo de vista es importante en su caso):

o Estructura: Diagramas de sistema, contenedores, componentes, y algún componente en especial es complejo y de interés entonces también su correspondiente diagrama de clases.

o Comportamiento: Diagramas (actividades, estados, interacción, secuencia, etc) para describir como interaccionan los componentes de alguno de los diagramas anteriores.

o Tecnología: Se puede incluir cierta información muy básica acerca de la tecnología que se seleccionaría (protocolos de comunicación, framework, lenguajes de programación, etc), y su justificación, en algunos de los diagramas anteriores o aparte en la documentación.

1. . Los estilos (o patrones) arquitectónicos que se han considerado, así justificando las posibles decisiones de diseño que se han tomado para la arquitectura que se propone (diapositivas 50-62).
2. Se debe entregar una documentación breve que ayude a completar la descripción de la propuesta según se comenta en las diapositivas 41-45. En particular las diapositivas 42 y 45 proporciona información acerca de las secciones de las que podría constar la documentación, quizás bastaría con 1 ó 2 párrafos (o lista de puntos) para describir lo más importante en relación con cada una de las secciones que sean necesarias o relevantes incluir

En caso de ser necesario, el estudiante debe consultar la bibliografía que se incluye en las diapositivas para encontrar y estudiar información más detallada. Se valorará el ejemplo seleccionado, propuesta y su descripción breve en el documento según el objetivo del ejercicio que se comenta a continuación. El objetivo es demostrar que se puede describir la arquitectura inicial de un sistema software de una forma lo suficientemente intuitiva (varios diagramas simples con documentación mínima) para que rápidamente trasladar al equipo de desarrollo una visión común del sistema a desarrollar haciendo uso de metodologías ágiles. Por supuesto la arquitectura iría evolucionando durante el desarrollo de forma natural. Por tanto, el ejercicio consiste en decidir que sería lo realmente importante acerca de poder proporcionar esa visión al comienzo dedicando el esfuerzo y tiempo justos.

WeChat Mini Program Platform Architecture

1 Diagramas

MVC Patterns

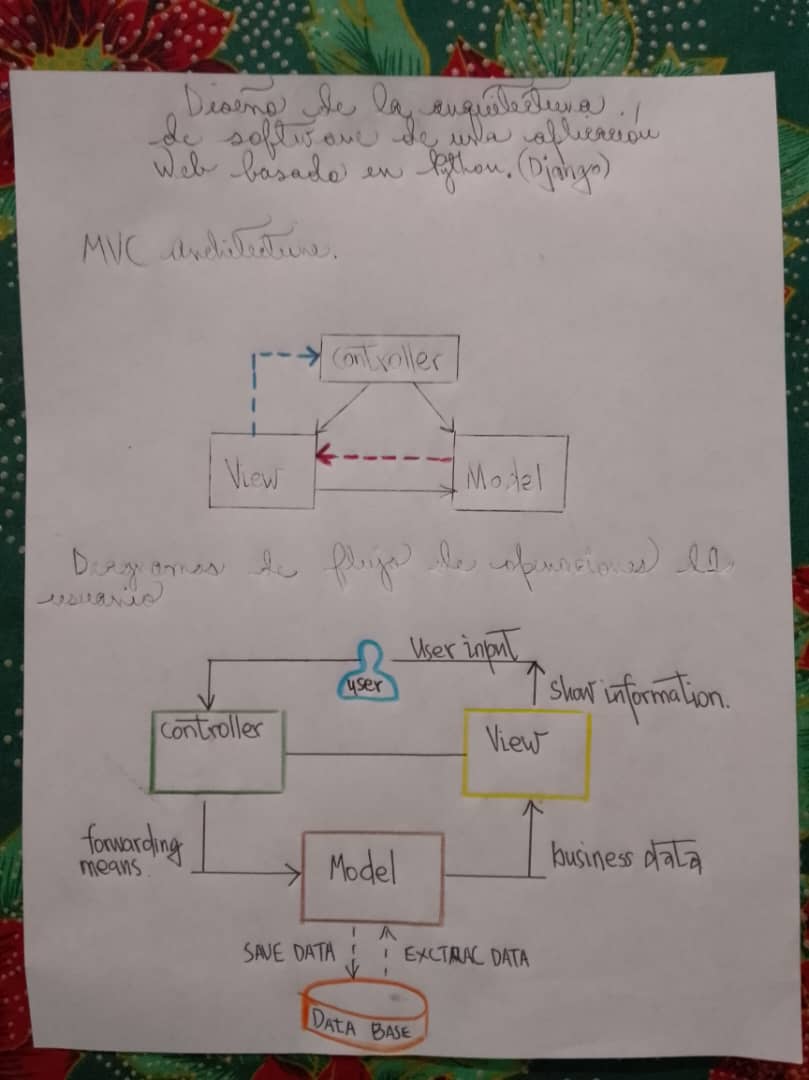
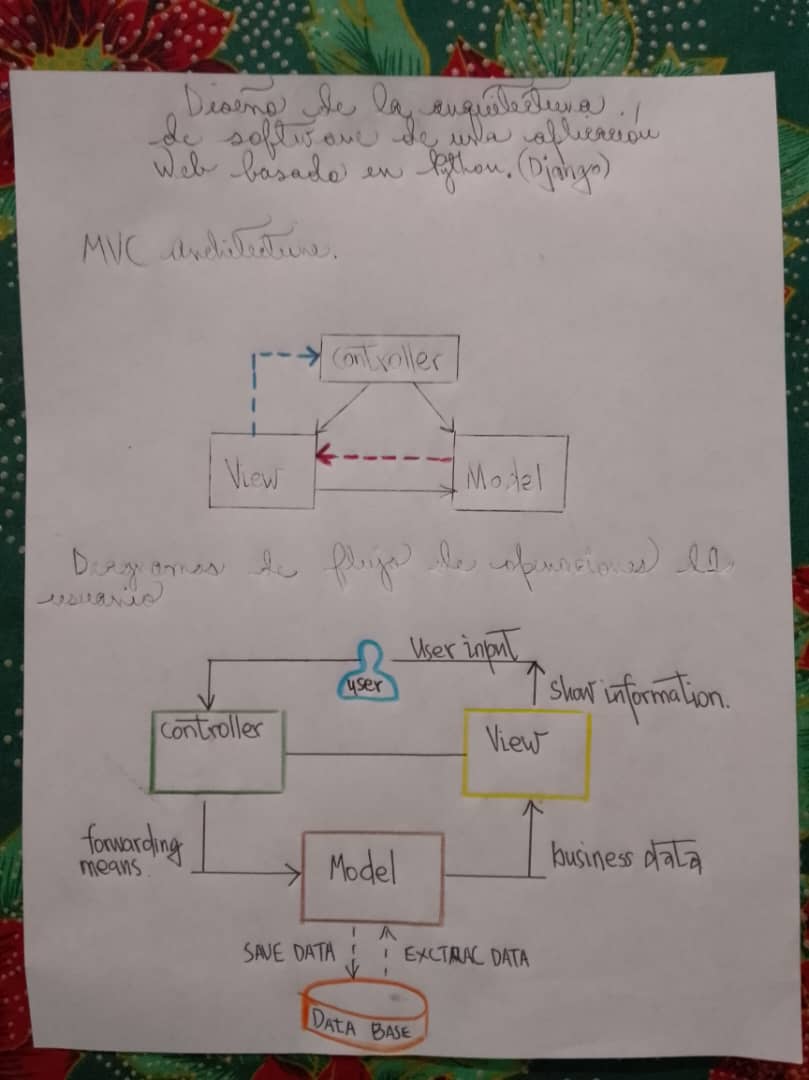
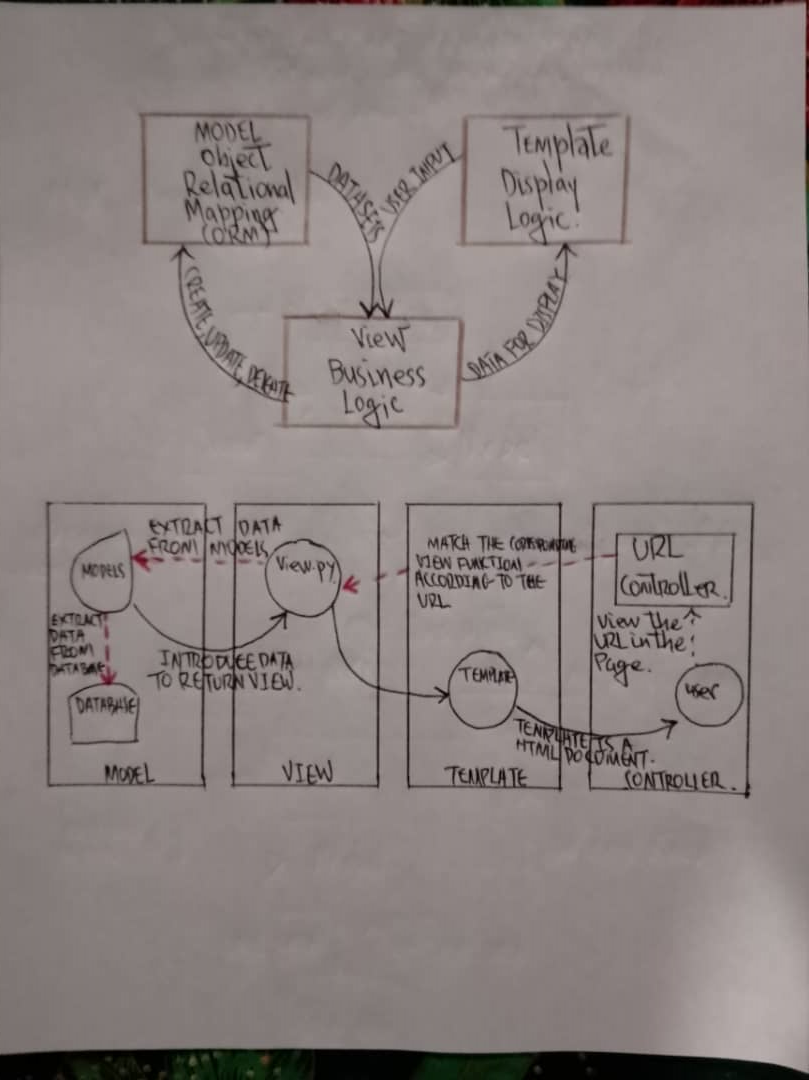


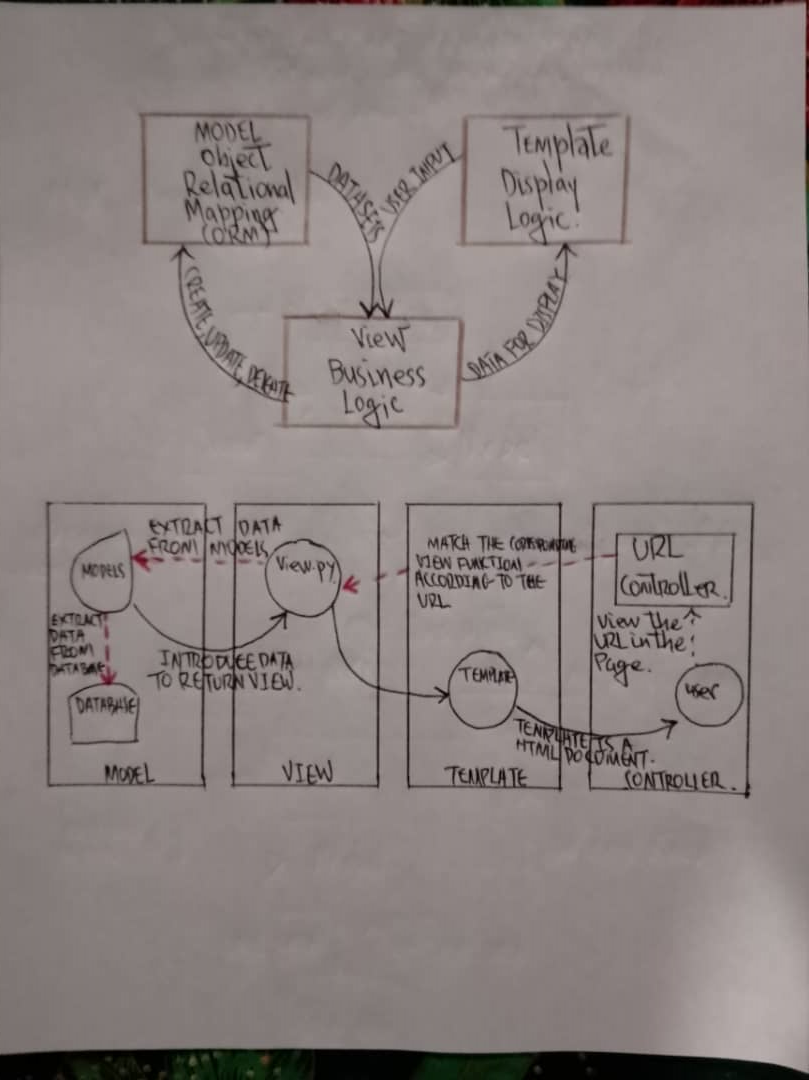
Diagrama de flujo de Operaciones del Usuario



MTV Model



User operation flow chart



2. Los estilos (o patrones) arquitectónicos que se han considerado, así justificando las posibles decisiones de diseño que se han tomado para la arquitectura que se propone.

Los patrones arquitectónicos considerados para el diseño de una aplicación Web basado en Python son los Modelos arquitectónicos MVC y MTV

modelo MVC

El patrón MVC (Modelo-vista-controlador) es un patrón de arquitectura de software en ingeniería de software, que divide el sistema de software en tres partes básicas: Modelo, Vista y Controlador.

MVC está conectado entre sí de forma enchufable y débilmente acoplada.

Modelo (M): escribe las funciones que debe tener el programa y es responsable del mapeo de objetos comerciales y bases de datos (ORM).

Ver (V): la interfaz gráfica responsable de la interacción con el usuario (página).

Controlador (C): responsable de reenviar solicitudes y procesarlas.

modelo MTV

El patrón MTV de Django es esencialmente el mismo que MVC. También tiene como objetivo mantener una relación de acoplamiento flexible entre los componentes, pero existen ligeras diferencias en las definiciones. MTV de Django se refiere a:

M representa modelo (Modelo): escribe las funciones que debe tener el programa y es responsable del mapeo de objetos comerciales y bases de datos (ORM).

T significa plantilla (Plantilla): responsable de cómo mostrar la página (html) al usuario.

V representa Vista: responsable de la lógica empresarial y de llamar al Modelo y Plantilla cuando sea apropiado.

Además de las tres capas anteriores, también se necesita un distribuidor de URL. Su función es distribuir las solicitudes de página de cada URL a diferentes Vistas para su procesamiento. Luego, la Vista llama al Modelo y Plantilla correspondientes. El modo de respuesta de MTV es el siguiente:

3. Documentación

El diseño de la arquitectura de software para una aplicación web basada en Python puede seguir diferentes patrones y enfoques según los requisitos específicos del proyecto. Uno de los enfoques comunes es el uso de una arquitectura basada en el patrón MVC (Modelo-Vista-Controlador). Aquí hay una descripción general de cómo podrías organizar los componentes de una aplicación web Python:

El diseño de la arquitectura de software implementada está basado en Python y cumple con los patrones MVC (Modelo Vista Controlador) y MTV implementado con el framework Web Django.

1. **Modelo (Model):**
   * Representa la capa de datos y la lógica de negocio.
   * Incluye clases que manejan la manipulación y gestión de datos.
   * Puede incluir acceso a bases de datos, servicios web, o cualquier otra fuente de datos.
2. **Vista (View):**
   * Representa la interfaz de usuario de la aplicación web.
   * Incluye plantillas HTML, CSS y, en algunos casos, JavaScript para la interactividad del cliente.
   * No debe contener lógica de negocio, solo presenta la información al usuario.
3. **Controlador (Controller):**
   * Actúa como intermediario entre el modelo y la vista.
   * Gestiona la lógica de la aplicación y responde a eventos del usuario.
   * Puede ser implementado como funciones o clases que manejan las solicitudes del cliente y coordinan la interacción entre la vista y el modelo.
4. **Servicios:**
   * Pueden ser utilizados para realizar tareas en segundo plano o para operaciones que no están directamente relacionadas con la interfaz de usuario.
   * Ejemplos incluyen servicios para la autenticación, autorización, envío de correos electrónicos, etc.
5. **Bases de Datos:**
   * Almacenan y recuperan datos necesarios para la aplicación.
   * Puedes utilizar sistemas de gestión de bases de datos relacionales (como PostgreSQL o MySQL) o no relacionales (como MongoDB).
6. **Middleware:**
   * Puede incluir componentes que se interponen entre el cliente y el servidor para realizar funciones como el enrutamiento, la gestión de sesiones, la compresión de respuestas, etc.
7. **Framework web:**
   * Utiliza un framework web de Python, como Django que proporciona estructuras y utilidades para facilitar el desarrollo web.
8. **Librerías de Utilidades:**
   * Incluyen funciones comunes y utilidades utilizadas en toda la aplicación.
   * Pueden ser bibliotecas de terceros o creadas internamente.
9. **Tests (Pruebas):**
   * Incluye pruebas unitarias y pruebas de integración para garantizar la calidad del código.

-­­­­