ARQUITECTURA DE SISTEMAS DE INFORMACIÓN

PRIMER PARCIAL

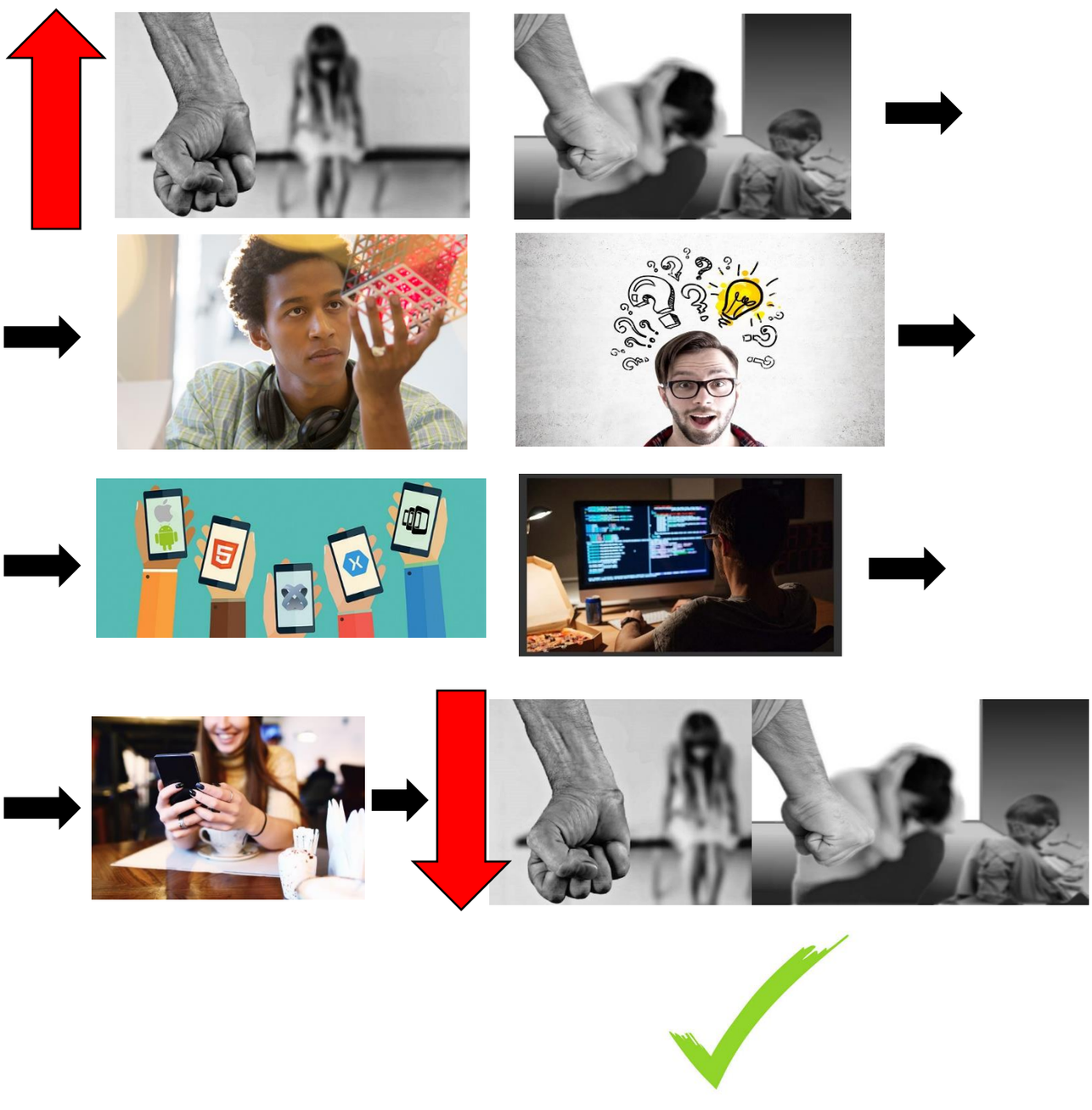
POR: ROBERTO ESTRADA NIETO

1) ¿Qué es una arquitectura de software y cuál es su relación con los sistemas de información? Explique en detalle.

La arquitectura de software es la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad), y servir como guía en el desarrollo. Al igual que en la ingeniería civil, las decisiones críticas relativas al diseño general de un sistema de software complejo deben de hacerse desde un principio.

La relación que manejan las arquitecturas de software con los sistemas de información es que, los sistemas de información al ser una herramienta que ayuda a administrar, recolectar, recuperar, procesar, almacenar y distribuir información relevante para los procesos fundamentales y las particularidades de cada organización, al momento de hacer un desarrollo deben implementar alguna arquitectura (la mejor según el proyecto) para poder tener de manera organizada la forma como el sistema va a interactuar.

2) Describa el proyecto que realizará en el curso. Apóyese en alguna figura.



3) Argumente en un audio de mínimo un minuto y máximo tres, por qué el Django y Angular son apropiados como backend y frontend, respectivamente, para el desarrollo de su proyecto.

Archivo de audio adjunto.

4) Explique las ventajas y desventajas de usar una API Rest en el proyecto que está desarrollando.

Ventajas:

Una forma de ver cómo REST se compara con otros en el paquete es compararlo con su rival más cercano, Simple Object Access Protocol (SOAP). En comparación con SOAP, se observa que REST aprovecha menos ancho de banda, y esta es una de las razones principales por las que los usuarios lo adaptan para el uso de Internet. A diferencia de SOAP, que solo facilita la devolución de datos en formato XML, REST puede devolver datos en formatos dispares como JSON, YAML u otros.

Otra ventaja es su adaptabilidad con la tecnología en la nube. Las API RESTful se prestan particularmente bien a dispositivos de perfil restringido, lo que lo convierte en una bendición para el desarrollo que involucra aplicaciones móviles.

Desventajas:

Pero como cualquier otro esquema, el diseño de API RESTful también tiene sus límites; de hecho, la teoría de Fielding sobre las API RESTful implica definirlas mediante seis restricciones arquitectónicas, a saber, el uso de una interfaz uniforme, su naturaleza como cliente-servidor basado, su capacidad para operaciones sin estado (llamadas que se pueden hacer independientemente unas de otras, además de contener todos los datos necesarios para completarse con éxito), su dependencia del almacenamiento en caché, su arquitectura de sistema en capas y su transmisión de código bajo demanda. Estas llamadas restricciones pueden considerarse como pautas sobre el carácter del servicio web y si es aconsejable adoptarlo en función de su diseño innato.

Además, el uso de un diseño de API REST puede requerir que los nuevos desarrolladores pasen por una curva de aprendizaje empinada. Los desarrolladores que no comprenden los límites de una API RESTful pueden sentirse frustrados en determinadas situaciones, como cuando pierden la capacidad de mantener el estado dentro de las sesiones.

5) Los proyectos y aplicaciones generados con el CLI de Django generan una serie de carpetas y archivos base. Explique la estructura de ficheros y carpetas, describa los elementos clave y por qué son importantes desde el punto de vista arquitectónico.

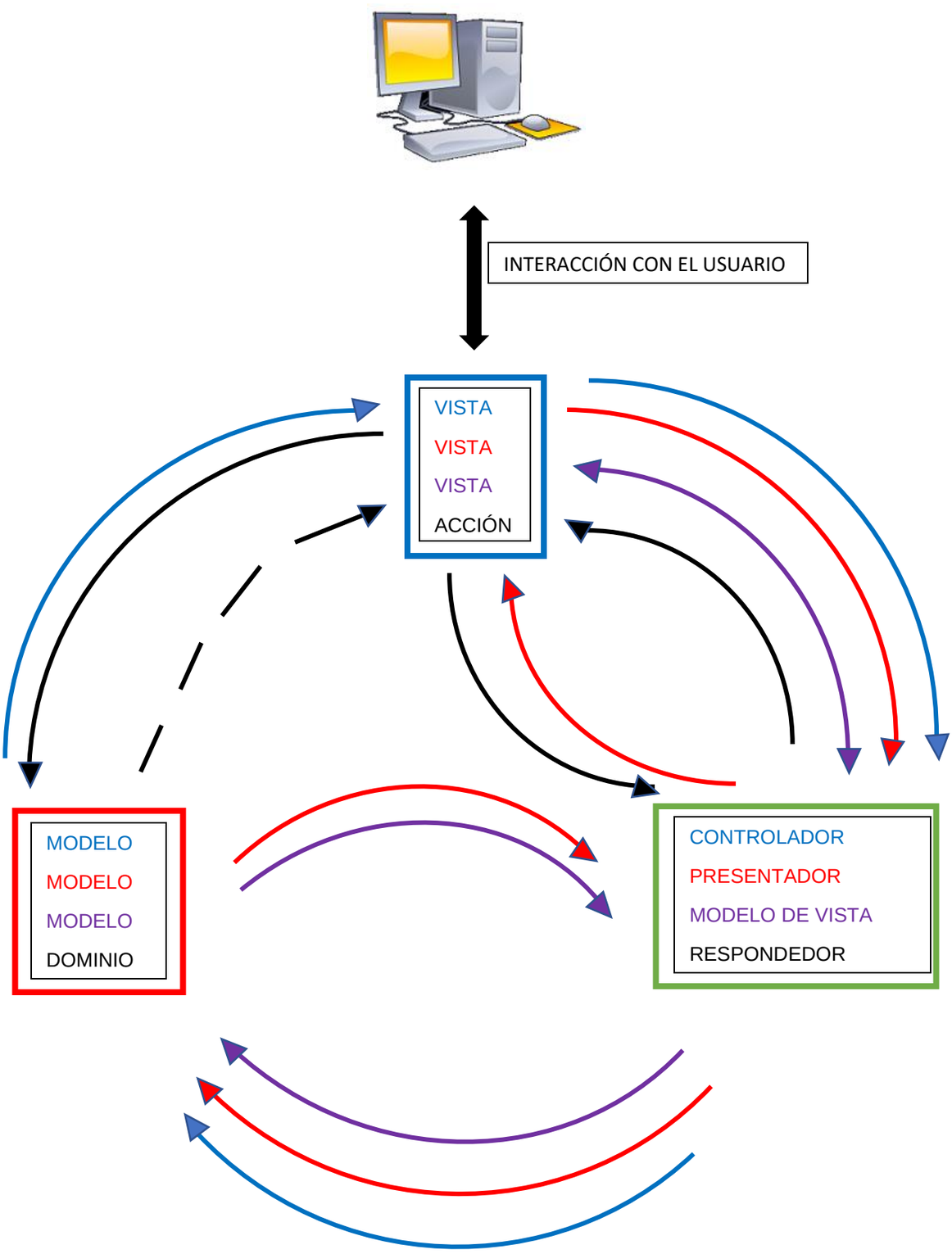
- **view.py**: Me ayuda en el componente visual de la página a desarrollar.
- **url.py**: Me ayuda a configurar todas las rutas de las views de la página a desarrollar.
- **manage.py**: Me ayuda a configurar, manejar e inicializar el programa de la página a desarrollar.

6) Grabe un audio en el que explique cómo instalar y crear un primer proyecto con Django. Hoy hay límite de tiempo.

Archivo de audio adjunto.

7) Realice un cuadro comparando las arquitecturas Model-View-Controller, Model-View-Presenter, Model-View-ViewModel y Action-Domain-Response.

COLOR	MODELO
	MVC
	MVP
	MVVM
	ADR



MVC: La vista hace llamados al controlador, el controlador manipula el modelo, el modelo lanza eventos a la vista.

MVP: La vista hace llamados al presentador, el presentador manipula el modelo, el modelo lanza eventos al presentador, y el presentador actualiza la vista.

MVVM: La vista y el modelo de vista se comunican entre ellos, el modelo de vista manipula al modelo, y el modelo lanza eventos al modelo de vista

ADR: La acción recibe la solicitud al dominio quien posiblemente regrese datos a la acción, la acción pasa el template al respondedor, el respondedor inyecta datos al template y devuelve una respuesta a la acción. quien finalmente se la pasa al cliente