Práctica Acceso a Datos – Stock Almacén

Análisis Técnico.

Estructura de la BD (StockERP)

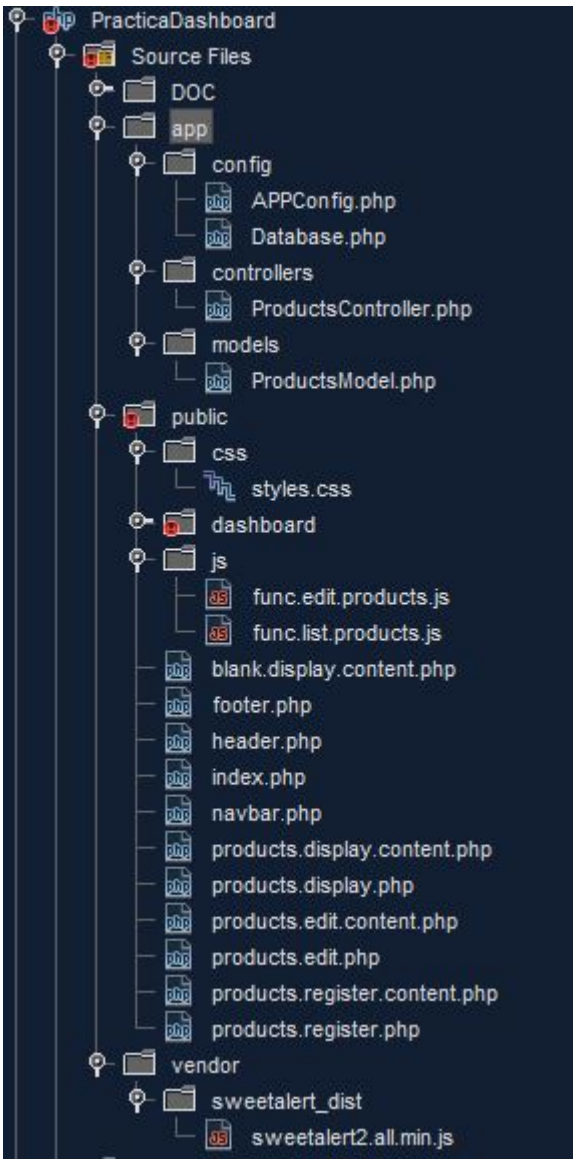
- Tabla ***Products***
  IDProduct => int not null Primary Key autoincrement
  name => varchar(75) not null
  description => varchar(150)
  IDCategory => int not null (Foreign Key que apuntará al PK de la tabla ***Categories***)
  price => float not null
  stock => int not null
- Tabla ***Categories***
  IDCategory => int not null Primary Key autoincrement
  description => varchar(75) not null
- Tabla ***Tickets***
  IDTicket => int not null Primary Key autoincrement
  amount => float not null
- Tabla ***TicketDetails***
  IDTicketDetail => int not null Primary Key autoincrement
  IDTicket => int not null (Foreign Key que apuntará al PK de la tabla ***Tickets***)
  IDProduct => int not null (Foreign Key que apuntará al PK de la tabla ***Products***)
  product_quantity => int not null
  amount => float not null

El nombre de la BD es orientativo , se puede escoger otro así como cambiar el idioma de los nombres de tablas y campos. Las tablas Tickets y TicketDetails las usaremos más adelante cuando se añada a esta practica la parte de simular compras.

Así como si se considera añadir más campos se puede hacer. Esta sería la estructura mínima de la práctica.

Recordar que desde workbench podemos crear facilmente las claves foráneas cuando estemos editando la estructura de la tabla , sería en la pestaña Foreign Keys justo al lado de la pestaña Indexes.

Estructura de directorios del proyecto y explicación de los archivos:



Carpeta **DOC** , documentación de la práctica (tanto el análisis funcional como este)

Carpeta **app → config** , aquí tenemos nuestros archivos de configuración que hemos ido usando en el resto de prácticas , solo tendremos que completar ***APPConfig.php*** para poner los datos de la BD que creemos.

Carpeta **app → controllers** , aquí se encuentra ***ProductsController*** , deberemos completar el metodo *initRequest* , aquí como ya hemos hecho anteriormente tendremos un switch que comprobará que tipo de consulta recibimos desde la petición ajax desde aquí llamaramos al método que le corresponda , en este caso tanto loadProducts() como registerProduct() no habría que añadir nada , en estos 2 metodos llamaremos a los metodos que correspondan del modelo.

Así mismo si nos hacen falta más metodos como por ejemplo para traernos los datos de la tabla *Categories* para renderizar su correspondiente select los crearemos aquí y en el modelo.

Cuando se añada la parte de simular la compra de productos lo correcto sería añadir en este directorio un archivo por ejemplo ShopController para crear los métodos de esa parte , pero en principio nos centraremos en la parte de productos.

Carpeta **app → models** , aquí se encuentra ***ProductsModel*** , en este archivo primero deberemos establecer la variables de clase *$table* y *$fields* como vimos en prácticas anteriores para poder usarlas en los métodos *_getProducts()* . Hay que completar el método *_insNewProduct($params)* , habrá que construir la consulta sql que nos permita insertar un nuevo producto en BD , cuidado con los errores 500 y hay que tener en cuenta también que al insertar un nuevo producto este tendrá una categoria por lo que habrá que incluir el ID que corresponda en el campo *IDCategory.*

De igual forma que con el controlador lo correcto cuando lleguemos a la parte de shop será crear un modelo aparte que la gestione de forma independiente.

Carpeta **public → css** , algunas clases css propias.

Carpeta **public → dashboard** , este el directorio con todos los archivos necesarios de la plantila que he usado para montar el front de la práctica (**AdminLTE**) , no tendremos que tocar nada aquí.

Solo tener en cuenta que si vamos a usar esta plantilla u otra similar en proyectos que no sean a nivel educativo o para proyectos personales hay que mirarse bien que tipo de licencia tienen por si queremos darle un uso comercial a nuestro proyecto.

Carpeta **public → dashboard → func.list.products.js** , este archivo js tendremos funciones para cargar los productos , hay que completar las funciones loadProducts() , renderTableProducts() de forma similar a como ya lo hemos hecho, tener en cuenta de que campos tendremos que mostrar.

Carpeta **public → dashboard → func.edit.products.js**, este archivo lo usaremos para las operaciones de insertar un nuevo producto , editar uno ya existente o borrarlo. Está por completar entero , aquí pondriamos tambien las funciones para hacer las validaciones que que nos piden en el analisis funcional, he incluido un ejemplo de como podriamos cargar un selector con datos que traemos de la BD.

Carpeta **public** , en el raiz de public he puesto los ficheros de las vistas para no complicar mucho la estructura aunque podríamos reorganizarlas y ponerlas por ejemplo en un directorio de app.

Public → blank.display.content.php , es solo una maqueta por si tuviésemos que crear otras vistas.
Public → footer.php,header.php , la estructura que incluiremos en nuestras vistas para el footer y el header.
Public → navbar.php , esta estructura es la que corresponde al menú horizontal que tenemos a la izquierda y sirve para enlazar con las distintas vistas.

Public → index.php , nuestro punto de entrada a al aplicación , en este caso como vereis enlazo directamente con *products.display.content.php* , para que muestre como pide el analisis funcional el listado completo de productos al entrar.

Public→ products.display.php | products.display.content.php , el primer archivo **products.display.php** sirve para enlazar header,navbar,footer y a su vez con **products.display.content.php** , se hace de esta forma para evitar que dentro del archivo principal tuviésemos que estar abriendo etiquetas php donde tocase para incluir ese navbar o header etc.

products.display.content.php es el archivo donde rendizaremos en este caso el listado de productos dentro de tal y como ya hemos visto. La tabla que hay en este archivo es maqueta para que tengais una guia de lo que debeis renderizar desde nuestro archivo javascript

para los fichero de editar y registrar seguimos el mismo sistema. Tened en cuenta que para las vistas de editar y registrar compartimos fichero js , he puesto una explicación en el fichero fun.edit.products.js en las lineas 10-13 para que no salten errores javascript al no encontrar elementos de una vista en la que no estemos.

La práctica puede abrumar un poco al principio por todo el contenido que tenemos que crear, como consejo leedlo todo muy bien tanto este análisis como el funcional , estudiad el código que ya teneis para que ver como está configurado , es lo que ya hemos hecho solo que un poco más extenso.

Para abordarlo primero construir la BD, e id completando por partes , es decir primero centraros en la parte de listar , completar el fichero func.list.products.js , crear las peticiones ajax correspondientes , revisad el controlador y el modelo para ver que todo está ok .

Después como consejo podríamos pasar a la parte de insertar un nuevo producto , aquí tendreis que poner mucha atención porque en el modelo hay que crear la sentencia sql correspondiente.

Id paso a paso y con mucha calma , visualizando desde la vista en que estemos que camino seguimos , es decir por ejemplo desde que pulso el boton de registrar que necesito → voy a mi fichero js correspondiente para hacer la peticion ajax→ que a su vez nos lleva al controlador que consultará al modelo.