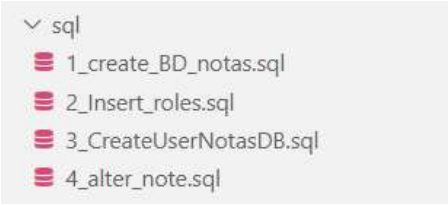
Desarrollo web en entorno servidor 2023-24 (DAW-DUAL-A)

Tarea06.1 (20% de la UD6)

Esta tarea tendrá una calificación máxima del 20% de la UD6

Se os facilita un código de partida que está en el repositorio https://github.com/dudwcs/Tarea06.1-enunciado.git

Es muy parecido al código básico de MVC con ficheros JSON, pero en esta ocasión, se trabajará contra la BD relacional **notas**. Tenéis los scripts de creación en la carpeta **sql** del repositorio:



Si ya habíais creado la BD en la UD4, el único script adicional que tendríais que ejecutar es el 4.

En dichos scripts se crean 2 usuarios:

- user1@edu.es con roles **user y admin** y contraseña de acceso abc123.
- user2@edu.es con rol **user** y contraseña de acceso abc123.

El código del proyecto de partida está estructurado con el patrón MVC y usa acceso a datos con PDO. Utiliza la siguiente arquitectura:

Las principales modificaciones tienen lugar en la capa de Repositorios:

Se ha creado:

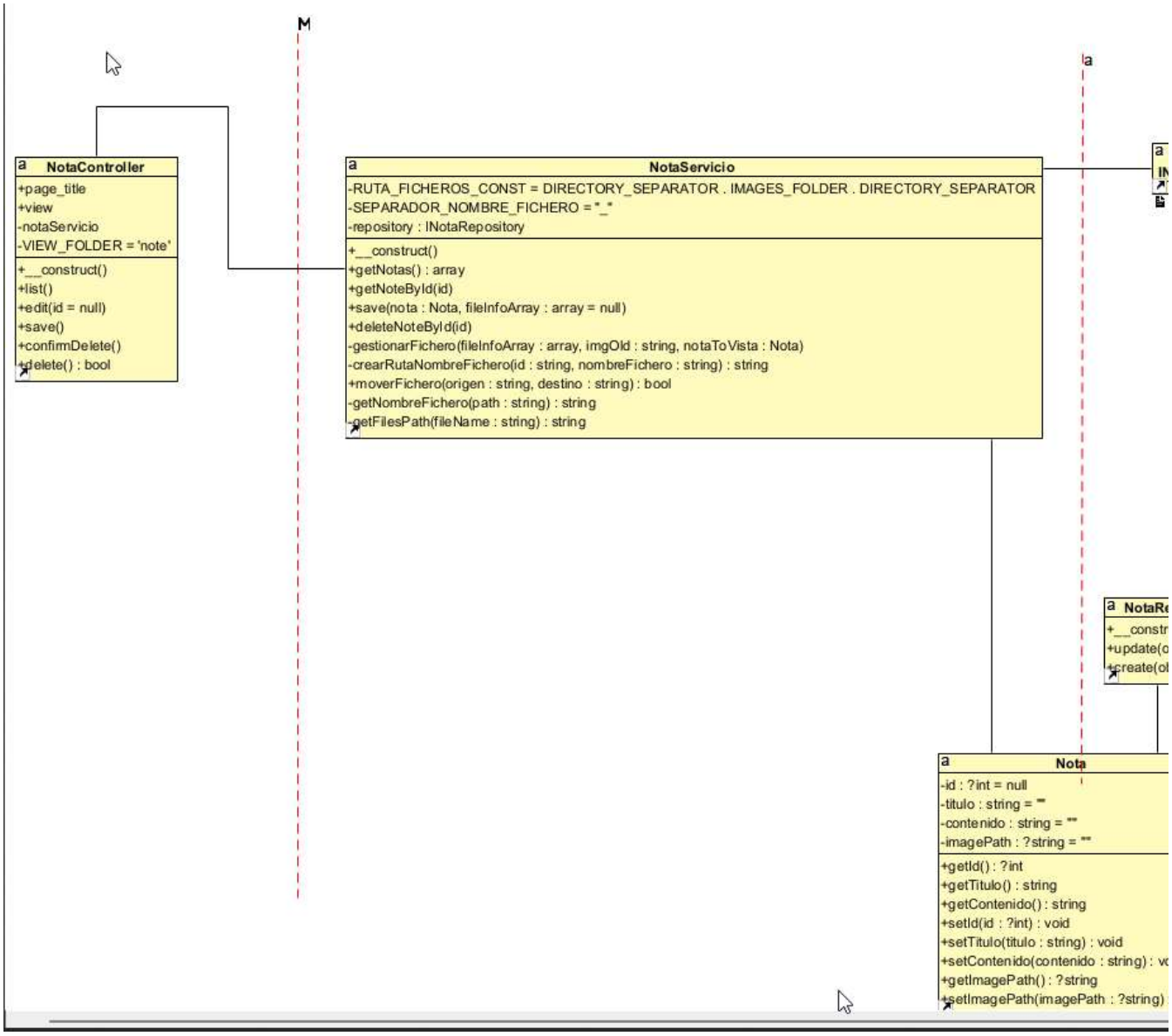
- una interfaz **IBaseRepository** con las 4 operaciones CRUD + findAll() para obtener todos los registros de una tabla
- una clase abstracta **BaseRepository**, que implementa IBaseRepository. Para los métodos fácilmente generalizables para cualquier tipo de entidad (delete, read por id, findAll) los métodos tienen un cuerpo que dependerá de 4 atributos protegidos:
  - **table_name**: Nombre de la tabla de BD de donde se obtendrán/modificarán los datos
  - **pk_name**: Nombre de la clave primaria (PK) de la tabla table_name
  - **class_name**: Nombre de la clase de POO con la que se mapeará la tabla table_name
  - **default_order_column**: Nombre de la columna por la que se ordenarán por defecto los resultados del método findAll

Para los métodos update y create que son muy dependientes del número de columnas de la tabla y del número de propiedades, se dejarán abstractos. La clase **BaseRepository** obtiene una conexión a BD PDO a través de **PDOSingleton.** Un **Singleton** es un patrón de diseño creacional que nos asegura que solo se creará un objeto de una determinada clase. El patrón *Singleton* provee una única instancia global gracias a que:

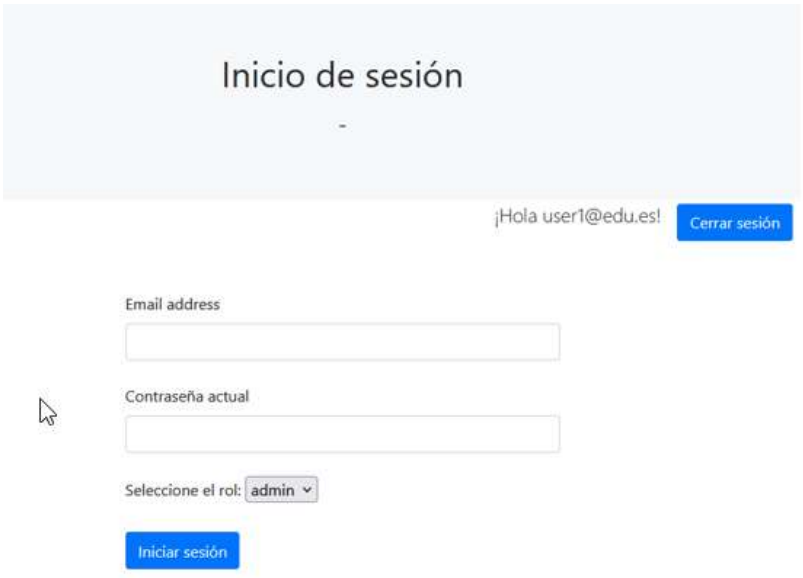
- La propia clase **es responsable de crear la única instancia**.
- Permite el **acceso global** a dicha instancia mediante **un método de clase (estático),** típicamente con **getInstance()**
- Declara el **constructor** de clase como **privado** para que no sea instanciable directamente.

Por cada entidad persistente habrá:

- una nueva interfaz **IXRepository**, donde X es el nombre de la entidad que extenderá de **IBaseRepository.** Si solo existen los 4 métodos CRUD + findAll para una entidad, esta interfaz estará vacía. Si se necesitan más métodos relacionados con una entidad X, se añadirán en IXRepository. Por ejemplo, findNoteByText(string $texto): array**;**
- una implementacion de **IXRepository** llamada **XRepository** con, al menos, la implementación de los métodos *update* y *create* y cualquier otro método extra que se necesite para la entidad X**,** definido en **IXRepository**



En la Tarea05.1 se pide, que completéis el código, para lograr iniciar sesión con cualquier rol:



Tendrá que llevarse a cabo, concretamente, lo siguiente:

1. Crea la interfaz IUsuarioRepository en la carpeta repository con el método:

findUsuarioByEmail($email): Usuario; **(0,25 puntos)**

La interfaz IUsuarioRepository debe extender la interfaz IBaseRepository **(0,25 puntos)**
2. Crea la clase UsuarioRepository que implemente la interfaz IUsuarioRepository **(0,25 puntos)** y extienda de BaseRepository **(0,25 puntos)**
a) Completa el constructor estableciendo los valores correspondientes para las propiedades

- table_name **(0,25 puntos)**
- pk_name **(0,25 puntos)**
- class_name **(0,25 puntos)**
- default_order_column para que se ordene por corre electrónico **(0,25 puntos)**

Observa cómo están implementados los repositorios NotaRepository y RolRepository por si te pueden resultar de ayuda.
b) Deja los métodos create y update heredados vacíos, simplemente con una sentencia return null y return false repectivamente. (Habría que implementarlos para proporcionar esas funcionalidades, pero no será necesario en esta tarea). **(0,2 puntos)**
c) Implementa el método ***findUsuarioByEmail($email): Usuario*** para que obtenga un objeto Usuario (con id, email y pwdhash) de la base de datos notas filtrado por $email. Si no se encuentra un usuario con ese email, deberá devolver null. Deberá realizarse con PDO, como los demás repositorios **(1 punto)**
3. En la clase **UsuarioServicio**:
a) Añade 2 propiedades

- $userRepository de tipo IUserRepository **(0,1 puntos)**
- $rolRepository de tipo IRolRepository **(0,1 puntos)**

b) Instancia esas 2 propiedades en el constructor vacío **(0,2 puntos)**
c) Implementa el método público ***function getUsuarios(): array*** para que, haciendo uso de los métodos de los repositorios ya creados, recupere todos los usuarios de la BD y para cada uno, recupere, a su vez, sus roles, estableciendo para cada objeto Usuario, sus roles a través del método setter.
Deberá devolver un array de objetos Usuario, donde cada objeto Usuario posee ya un array de objetos Rol.
No es necesario crear métodos nuevos en los repositorios para este apartado. **(1 punto)**
d) Implementa el método público ***function login(string $user, string $pwd, $rolId): ?Usuario*** para que haciendo uso de los métodos de los repositorios:

- Recupere el usuario filtrando por email.
- Compruebe con *password_verify* si la contraseña es correcta.
  - Si es correcta, deberá recuperar los roles del usuario y establecerlos en el objeto Usuario a través del método setter. Deberá comprobar si el rol indicado en el formulario html está entre los roles permitidos de la BD usando el método *isUserInRole* ya proporcionado. Si el rol está dentro de los permitidos en BD, devolverá el objeto usuario con los roles incorporados. En caso contrario, devolverá null.
  - En caso contrario, devolverá null.

(2 puntos)
4. En la clase **UsuarioController**, completa la implementación del método **login()** para que:
a) si vienen por el método HTTP POST los datos email, contraseña y rol, compruebe a través del servicio si las credenciales son válidas.

- Si no lo son, en loginViewData invoca el método setStatus a la constante Util::OPERATION_NOK
- En caso de que el login del servicio sea correcto, se guardarán en la sesión el id del usuario, el id del rol seleccionado y el email del usuario. A continuación, se llamará al método *redirectAccordingToRole* para que redirija al listado de usuarios si se usa el rol admin o al listado de notas si se usa el rol user.

Si no vienen por POST esos 3 datos, se devolverá $loginViewData
(2 puntos)
5. Modifica la vista **header.php** para que el botón cerrar sesión esté dentro de un formulario que invoque el método **logout** del controlador **UsuarioController** con el método HTTP POST **(0,5 puntos)**
Comprueba si eres capaz de iniciar sesión con cada uno de los roles (admin, user) y cerrar sesión con cualquiera de ellos.

- Si es posible iniciar sesión desde el formulario html y la aplicación funciona correctamente **(0,7 puntos)**
- Si es posible cerrar sesión a través del formulario html y la aplicación funciona correctamente **(0,2 puntos)**

INSTRUCCIONES DE ENTREGA

- Se espera un único fichero comprimido con el código PHP de tu solución

PLAZO MÁXIMO DE ENTREGA: domingo 03/03/2024 a las 23:59h

Estado da entrega

Estado da entrega	Sen intentos
Estado das cualificacións	Sen corrixir
Data límite	Domingo, 3 de Marzo de 2024, 23:59
Tempo restante	4 días 5 horas
Última modificación	-
Comentarios a entrega	Comentarios (0)

Engadir entrega

Vostede aínda non fixo ningunha entrega

◄ Actividad 6.8 Estructura MVC con ficheros JSON como persistencia

Ir a...

Foro UD7 ►