Prueba técnica de Magento 2- Paola Alicia Díaz Aliaga

Crear un nuevo módulo que va a usarse para gestionar las notas de los alumnos en un examen, para ello hay que crear una nueva entidad, un service contract que la administre, un data patch que permita añadir datos a la tabla y un controlador que muestre la información guardada para posteriormente añadirle js y darle estilos.
Comandos:

Sudo Docker exec -it magento-php-course-2020 bash
Sudo Docker-compose start o stop
Rm -rf generated/*

1- Crear un nuevo módulo cuyo nombre sea tu apellido (sin tildes) y el vendor sea Hiberus, por ejemplo: Hiberus_Garcia.

Creamos el modulo en app->code->Hiberus llamado **DiazAliaga** que contenga el archivo **registration.php** y etc->**module.xml** para que magento lo reconozca como modulo.

2- Crear una única tabla llamada hiberus_exam que responda exactamente a la siguiente estructura:

Creamos el fichero **db_schema** dentro del directorio etc para definir la tabla de bd en la que se le añadirá las columnas que se requieren y estableciendo como llave primaria “id_exam”. Una vez que se han rellenado los atributos de las columnas hay que ejecutar dentro del contenedor de php el comando “bin/magento setup:upgrade”.
Para comprobar que se ha creado el modulo y la tabla podemos ejecutar dentro del contenedor php el comando “bin/magento module:status” o acceder al workbench y buscar el nombre de la tabla para ejecutarlo y ver que se ha creado (todavía con campos vacios).

3- Crear el Service Contracts y ORM que gestione esta entidad.

Serán necesarias tantas **interfaces** de modelos, repositorios y busqueda como tablas se tengan, en este caso 1 de cada:

- Dentro del directorio Api->Data creamos el fichero **ExamInterface.php** para definir la interfaz del modelo que contendrá los getters y setters de las columnas que tiene la entidad
- Creamos la interfaz del repositorio dentro del directorio Api en el fichero llamado **ExamRepositoryInterface.php** que define las operaciones que se van a hacer en la bd (como guardar, borrar, obtener un listado).
- Creamos la interfaz de búsqueda **ExamSearchResultsInterface.php** dentro del directorio Api->Data que obtendrá un listado de entidades en base a un criterio de búsqueda

También necesitamos crear las clases que implementen las interfaces del **modelo** y del repositorio que almacena y coge los datos de la entidad:

- Creamos el directorio Model->ResourceModel y dentro de el para cada una de las interfaces los archivos que enlazaran el modelo con el resourcemodel y este con la tabla. Creamos el archivo **Exam.php** que extiende del abstract y se va a encargar de guardar, cargar y eliminar los datos del modelo
- Dentro de Model creamos los archivos **Exam.php** (que extenderá del abstractModel (hereda propiedades del core que necesita una clase para ser un modelo), implementará de la interfaz Exam (con las funciones que se definieronp para completarlas aqui) y el constructor que enlaza el modelo con el resourcemodel) y también se creara el archivo **ExamRepository.php** (que hara las funciones que se definieron en la interfaz).
- Creamos el fichero **Collection.php** en la ruta Model->ResourceModel->Exam (mismo nombre de la entidad) que contendrá un constructor con el modelo y el resourcemodel con el que se podrá manejar la entidad

Para enlazar las **interfaces** con los **modelos** necesitamos la clase **di.xml** que estará dentro del directorio etc.

4- Crear un Setup (Db Schema y Data Patch) para introducir datos que introduzca en la tabla creada utilizando los service contracts. Por defecto podéis construir un array con la información a añadir.

Creamos el directorio Setup->Patch->Data para crear los Data Patch:

- El PopulateDataModel.php
- El LinkDataModel.php

* Como alternativa opcional, podéis traer esa información (nombre y apellido) desde un CSV, como pista, ese csv puede estar dentro de vuestro módulo en "Vendor\Apellido\Setup\data\import.csv" y leerlo haciendo uso de la clase de Magento 2: Magento\Framework\File\Csv y su función "getData($file)" a la cual le pasáis un path de fichero y os devuelve un array de lo que ha leído en el fichero. Únicamente el csv debe tener las columnas nombre y apellido.
La nota deberá introducirse de manera aleatoria, lo ideal sería que generase notas del 0 al 10 con 2 decimales.*

5- Crear un nuevo controlador de frontend, el front name debe llamarse como tú apellido (ignorad tildes). Haz de momento que simplemente diga echo "Hola", posteriormente lo modificaremos.

Creamos el directorio Controller->Mensaje y el archivo **Index.php** que extenderá del Action y mediante inyección de dependencias con el pageFactory se traerá el fichero xml para renderizar su contenido.
Para crear el contenido que ira en la pagina creamos el fichero **diazaliaga_mensaje_index.xml** (que debe tener la estructura del

routeid_controllerName_action) en la ruta view->frontend->layout que contendrá un contenedor cuyo name es “content y un bloque con un nombre y un template que referencie a la plantilla (**mensaje.phtml**)que se encuentra en view->frontend->template que contendrá el mensaje Hola. Borramos cache antes de probar la url que será diazaliaga/mensaje.

6- Asociar un layout, bloque y template a nuestro controlador de acción para poder devolver un listado de los exámenes de los alumnos en el frontend.

Se deben seguir las siguientes especificaciones:

1. Añadir un título h2 a la página con el class=”title”.

   Sigo los mismos pasos del ejercicio anterior, en este caso creo un controlador Controller->Listado->**Index.php**, creo el layout view->frontend->layout->**diazaliaga_listado_index.php** al que añado el bloque que realiza la lógica que mostrara el template, un template view->frontend->template->**listado.phtml** y el bloque en el directorio Block->**Block.php**

2. En el listado (ul) debe mostrarse el nombre, apellido y la nota.

   Desde el template, dentro del listado (ul) y de la etiqueta li hacemos un bucle foreach en la que llamamos a la función *getListExams()* y en cada párrafo llamamos a la función del nombre, el apellido y la nota. Para diferenciar una lista de otra añado una línea (<hr>) al final de la etiqueta li.

3. Debajo del listado, añadir un texto “Total number of students: XX.” donde XX debe corresponder al total de alumnos almacenados.

   Declaro un contador a 0 antes del bucle para que se incremente en cada iteración y lo llamo en un párrafo fuera del bucle y del listado.

4. Añadir una traducción al español a nivel de módulo para el literal anterior, de modo que el texto mostrado sea: “Total: XX alumnos.”

   Creo la carpeta i18n y dentro de ella creo el archivo **es_ES.csv**. El texto original aparece primero entre comillas (“”), a continuación una “,” y de nuevo entre comillas el texto traducido al español.

7- Asociar un js por require a la página para que desde un botón se pueda ocultar y desocultar las notas de los alumnos.

Añado un botón después del listado y un script que llama a la clase del listado y a la clase que ejecuta la acción que se encuentra en view->frontend->web->js/**notas.js** en la que comprueba que si es visible se oculte y sino lo muestre (habiendo añadido previamente un div y su clase que es la que llama desde el js).

8- Maquetar el listado usando less en el módulo siguiendo las siguientes especificaciones:

1. El título debe tener por defecto un color y a partir de 768 píxeles ponerse de otro.

   En el directorio view->frontend->web->css->source creo un archivo **_module.less** en la que defino un color naranja al titulo por defecto con la clase & when (@media-common = true) y con la clase .media-width(….) le establezco el color azul.

2. Dejad el listado con la mejor apariencia que te parezca.

   Para ello coloco tanto el titulo como la lista centrada.

3. Haced por css que los impares tengan un margen izquierdo de 20px, este valor debe estar definido como variable al principio del less, por ejemplo @margin-left-primary.

   Declaro el valor del margen (20px) y llamando a la lista le aplico la pseudo-clase nth-of-type(2+1) para que las listas impares tengan ese margen.

9- Añadir un nuevo botón que muestre la nota más alta de todos los alumnos en un alert, busca la manera más eficiente para el servidor. (EXTRA: Utiliza el jQuery widget "alert" para mostrar esta nota)

Utilizando el mismo script del ejercicio 7 en la clase **listado.phtml**, añado la variable de la nota más alta y otro input con la id "boton_alta" que se utilizara en el archivo js **notas.js** para añdirle ahí el alert.

10- Sacar en una nueva fila la media de notas que ha sacado la clase.

En el template **listado.phtml** creo la variable $media antes del bucle a 0 para que en cada iteración vaya sumando la nota y después la llamo desde el párrafo fuera del bucle que divide la variable de la suma de las notas entre el contador.

11- Crear un plugin que ponga un 4.9 a todos los alumnos que hayan suspendido (no se tiene que guardar en db).

Creo el directorio Plugin y dentro el archivo **Nota.php** que sobrescriba el comportamiento de la nota.

12- Que los alumnos aprobados aparezcan en un color y los suspensos en otro. (EXTRA: Define estos estilos mediante un LESS MIXIN, de modo que el color a aplicar sea un parámetro de entrada del mixin)

En el template **listado.phtml** dentro del contenedor div que incluye la llamada al método getMarks() y mediante php compruebo que si la nota es mayor que 5 el fondo en el style sea verde y sino rojo.

13- Que además los 3 mejores aparezcan destacados de otra forma aún más destacada, podéis utilizar cualquier forma que se os ocurra, js, php...

En el template **listado.phtml** creo la variable $mejores a 3 para que en cada iteración se decremente; el listado de las notas esta ordenado de forma descendente para que salgan las notas mas altas primero mediante el sortOrder en el bloque **listado.php**. En la etiqueta li hago la comprobación para que me aparezcan los 3 mejores poniéndoles un color diferente a los 3.

14- Crear un CLI command nuevo que permita ver los todos los datos de la tabla de exámenes, se valorará que NO se haga uso del object manager. Este se debe llamar como tu apellido bajo el namespace Hiberus (hiberus:apellido).

Crear el directorio Console->Command y dentro una clase **ShowExamsCommand.php** que extienda de la clase Command y deberá de contener 2 funciones esenciales: configure() (en la que definimos el nombre del comando, su descripción y la definición de los parámetros que puede recibir) y execute() que recibirá siempre por parámetro un InputInterface y un OutputInterface(para poder escribir en la terminal). En la función process es donde le indicaremos que datos debe mostrar.
Para que magento sepa que el comando pertenece a su lista de comandos hay que añadir la configuración al archivo etc->di.xml y borrar cache.

15- Crear 3 endpoint nuevo de Api Rest con Swagger:

- Permitir ver todos los datos de la tabla de exámenes.

  Creamos dentro de etc el archivo **webapi.xml** que tendrá todos los endpoint del modulo y tendrá los routes que se quieran añadir a la api de magento. Los routes tendrán una url y dentro el service con la interfaz del repositorio que queremos exponer y el método que queremos usar del repositorio. Tambien el tipo de autenticación en la etiqueta resource.

- Crear otro endpoint que permita borrar alumnos por id.

  Añadimos al webapi.xml otro route con el método “DELETE” y en el service el método “deleteById”.

- Crear otro que permita guardar un nuevo alumno y su nota.

  En el mismo archivo añadimos otro route con el método “POST” y en el service el método “save”.

16- Crear una nueva sección de configuración para vuestro módulo (con su tab asociada de Hiberus) que permita añadir los siguientes campos configurables:

- Poder configurar cuantos elementos mostraremos en el listado de exámenes que hemos creado en el frontend, en la nueva página.

  Creo dentro del directorio etc el directorio adminhtml con dos ficheros **menu.xml** (que mostrara en el admin el icono y la pestaña de configuración) y **system.xml** (en la que le paso el tipo de tab, la sección, el grupo y los campos.
  Añado el campo "eleList" que será de tipo text con prioridad 1 que indica los elementos que se mostraran en el listado de exámenes.
  Tambien hay que crear un Helper->**Config.php** que extiende del AbstractHelper donde se crean las funciones que hacen las modificaciones. Creo la función getEleList() que se trae la configuración del admin.
  Ahora lo podemos utilizar en el bloque dentro de la función getListExams() para que en la consulta se establezca el numero de elementos al que se defina en el administrador.

- Poder configurar cual es la nota que marca el aprobado (por defecto 5.0)

  En el system.xml añado el campo "notaAprobado" que será de tipo text con prioridad 2 para que este después del "eleList" que indica la nota minima necesaria para aprobar.
  Tambien es necesario crear el archivo config.xml en el directorio etc para definir la nota por defecto (5) que tendrá el identificador de la sección, identificador del grupo y sobre que campo queremos añadir la configuración por defecto.

17- Crear un custom Logger que registre cada vez que se accede a la página nueva del listado de exámenes y nos indique cuantos alumnos se van a mostrar y cuál es la nota media, para tener un control de qué se le está mostrando al cliente cuando accede a esta página.

- El fichero de log nuevo, deberá llamarse con tu apellido bajo el namespace hiberus: hiberus_garcia.log

  En el archivo etc/di.xml agregamos dos virtualtype:
  - Uno que extienda del Logger->Handler->Base. Es el que da la funcionalidad para escribir en el fichero y le pasamos como nombre de fichero hiberus_diazaliaga.log dentro del directorio var/log.
  - Otro que extienda del Logger->Monolog que utiliza la clase virtual anterior. Utiliza un array de handlers que son los que escribirán los ficheros. En este virtualtype le pasamos el anterior virtualtype mediante el array de handlers
  - Otro que va a dirigirse al Bloque->Listado y se le incluye el argument del Monolog. En el bloque se añade la función que llama al logger y al debug de

los virtualtypes y le paso los dos parámetros que en el template utilizo como alumnos y media.

- No se deberá crear ninguna clase nueva, se deberán utilizar los virtual types para solucionar este problema.

Por último, se pide que durante la realización de la prueba vayáis anotando las cosas que vais haciendo para redactar una memoria donde
se explique lo que hacéis y las tomas de decisiones que realicéis en cada uno de los ejercicios. Con esto se redactará un documento que se exportará a PDF y se adjuntará con el módulo desarrollado. A la hora de escribir la memoria, imaginad que os preguntan cara a cara qué habéis hecho en este módulo.
Como entregar la prueba Se creará un repositorio de Github público donde o bien vais incorporando progresivamente todo lo que vais haciendo durante los días o bien lo subís todo de golpe al final, de esta manera luego revisarlo es mucho más accesible y por otro lado os sirve para desenvolveros con este control de versiones que vais a necesitar usar en cualquier trabajo.
En este repositorio deberán estar tanto el módulo desarrollado, como el PDF de la memoria escrita.
Finalmente lo que nos tendréis que mandar es tan solo el enlace a vuestro repositorio al correo que os indiquemos.