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

He creado un módulo nuevo. Para ello he creado una carpeta llamada Hiberus en la ruta App/Code, la cual sería el vendor. Y dentro de ella otra carpeta con el nombre del módulo, Carrero.
Además de esto, el módulo para que el módulo sea reconocido por Magento, debe llevar otros dos ficheros que lo activan como módulo: El fichero etc/di.xml y registration.php.

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

Field	Type	Null	Key	Default	Extra
id_exam	int(11)	NO	PRI	<null>	auto_increment
firstname	varchar(100)	NO		<null>	
lastname	varchar(250)	NO		<null>	
mark	decimal(4,2)	NO		<null>	

Para crear la tabla, he utilizado el fichero para crear el schema de la tabla en etc/db_schema.xml. En él se crea la tabla y las columnas que la componen, en este caso siguiendo la estructura detallada en la captura de el enunciado.

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

Para crear el Service Contracts y ORM, hay que crear varios ficheros. Por un lado hay que crear las interfaces que utilizarán los modelos y el repositorio, en Api/CarreroRepositoryInterface.php y en Api/Data/CarreroInterface.
Y por otro he creado los modelos en Model/ResourceModel/Carrerob.php , Model/Carrerob.php y el repositorio en Model/CarreroRepository.php. De esta forma se pueden crear los objetos de los alumnos y leer y escribir de la tabla hiberus_exam.
Por otra parte en el fichero etc/di.xml hay que asociar cada interfaz con su modelo con los bloques <preference>. Y también doy de alta los metadata con la información de la base de datos: el nombre de la tabla y el id primario.

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.

* 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.**

En este ejercicio ya tenía creado el db_schema ya que en el anterior interpreté que debía hacerse ahí para crear la tabla. Por lo que me centré en realizar el Data Patch. Para ello hay que crear la siguiente estructura de directorios y fichero: Setup/Patch/Data/AddData.php.
Como quería hacer la alternativa opcional, me creé un fichero csv con datos de 5 alumnos con nombre y apellido.
Desde el método apply en AddData.php, se lee el fichero csv para obtener los datos y añadirlos a la tabla, junto con un número aleatorio para la nota, el cual genera un float entre 0 y 10.

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.

Creo un controlador en Controller/Index/Index el cual contiene la clase que responderá cuando se introduzca la url carrero/index/index. Para indicar que la url carrero corresponde al controlador que hay en este módulo, hay que indicarlo también en el fichero etc/frontend/routes.xml.
En este paso, solo se muestra un "Hola" en la página, ya que aún no hay ningún template ni layout asociados a este controlador.

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".**
2. En el listado (**ul**) debe mostrarse el **nombre, apellido y la nota.**
3. Debajo del listado, añadir un texto "Total number of students XX." donde XX debe corresponder al total de alumnos almacenados.
4. Añadir una traducción al español a nivel de módulo para el literal anterior, de modo que el texto mostrado sea: "Total: XX alumnos."

Primero creo el layout y el template. Para ello, creo los directorios view/frontend y dentro de ellos uno layout y otro templates. En layout creo el layout para el modulo que será el que responda al controlador por lo que el nombre del fichero xml es carrero_index_index.xml.
En este fichero asocio el bloque (que aún no está creado) con el template pthml.

Dentro de templates, creo un fichero index.phtml, el cual muestra el contenido de la página al usuario al entrar en la ruta del controlador. Aquí añado la estructura detallada en el enunciado.

Ahora creo también el bloque, que será el que proveerá a esta plantilla los métodos y los datos a mostrar desde la base de datos. Por lo que creo un método que devuelve todo el listado de alumnos para rellenar la lista de alumnos.

Para crear la traducción, añado al texto a traducir en la plantilla phtml __('texto') para indicar que ese literal tiene una traducción. Y por otra parte creo una carpeta i18n que es donde se ponen los diferentes archivos csv con la traducción a cada idioma que se quiera traducir. En este caso a es_ES de español. Y se da de alta la cadena a traducir y su traducción.

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

Los ficheros .less hay que meterlos dentro del directorio view/frontend/web/css/source. En mi caso creo un fichero _module.less, el cual le dará los estilos al módulo. Ya añado los estilos para que hagan lo que pide el ejercicio.

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)*

Lo primero que hay que hacer es obtener la nota más alta. Para ello creo otro método en el bloque que me calcula la nota más alta. Y llamo a este método desde la vista index.phtml y la guardo en un input hidden.
Ahora con js, recogeré ese valor del input y lo mostraré en un alert. Para inyectar el js hay que crear varias cosas:

Por un lado el requirejs-config.js donde se mapean los js que se añadirán al modulo. Y en view/frontend/web/js, se crean los js que indicamos en el paso anterior. Para la nota más alta, he creado un maxnote.js, el cual recoge el valor del input y lo muestra con el alert. Todo esto realizado con jQuery.

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

Al igual que en el ejercicio anterior, para extraer los datos, me creo un método en el bloque index.php el cual me calcula la media de las notas y las imprime en otra fila.

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

El plugin hay que darlo de alta en el fichero di.xml. Se indica el nombre del plugin y la ruta en la que se encuentra.
Después en el directorio Plugin/Carrerob creo el CarrerobPlugin.php el cual ejecutará su método después ejecutarse el método del bloque que devuelve el listado de los alumnos. Aquí comprobará los alumnos que tengan una nota inferior a 5 y les cambiará la nota a 4.9. De esta forma no se guarda en la base de datos.

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)*

Primero compruebo en la vista si la nota del alumno a mostrar es inferior o superior a 5. Si es superior o igual, añado una clase al listado llamada pass y si no una llamada fail.
Doy de alta estos estilos en el _extend.less para que los aprobados sean de color verde y los suspensos de color rojo.
Para hacerlo con mixin, creo un fichero _mixin.less junto con el _module.less, el cual contendrá el mixin. Este mixin solo aplica el color que tendrá el texto de la lista. Importo el mixin desde _module y llamo al mixin desde la clase pass y fail, pasándole como variable el color.

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

Añado otro método al bloque hace una consulta a la base de datos ordenada descendentemente por la nota. Llamo a este método desde la vista y a la hora crear la lista comparo si el id del alumno está en ese top 3. Si está incluyo otra clase que pone el borde de esa línea de la lista más grande y en azul.

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).**

Para crear un CLI, hay que crear el directorio Console dentro del proyecto. Y a su vez, una clase php que llamo ExamCommand.php, en la cual se registra el comando y la funcionalidad del mismo. Por lo que en el método configure() se genera el nombre del método: hiberus:carrero y en el método execute, la acción que ejecuta, en este caso devolver un listado con todos los alumnos y su nota.
Hay que registrarlo también en el di.xml como tipo CommandListInterface, para que se reconozca como comando.

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

- Permitir ver todos los datos de la tabla de exámenes.
- Crear otro endpoint que permita borrar alumnos por id.
- Crear otro que permita guardar un nuevo alumno y su nota.

Para crear los endpoint de Api, los he creado en webapi.xml dentro de etc. Aquí he especificado la ruta para cada ejercicio. rest/V1/carrerob/exam, rest/V1/carrerob/exam/removebyid/id y rest/V1/carrerob/exam/create/firstname/lastname/mark. Y cada ruta apunta a su método correspondiente de CarreroRepository. Para cada opción he creado un método. Y cada método, para practicar he hecho la consulta/creación/eliminación de datos de una forma distinta.
Eliminar lo he hecho utilizando el InterfaceFactory. Crear con el resoruce getconnection, ejecutando un sql directamente. Y de leer todos los registros a través de un Collection.

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

Para crear la sección de configuración hay que crear una carpeta adminhtml dentro de etc. Y un fichero system.xml dentro de esta carpeta. En este fichero se introducen las secciones, tabs y variables para dar la estructura a la página de configuración.
Se crean dos fields de integer, para recoger los datos que pide el ejercicio. Y por otro lado se llama y recoge esta configuración desde los métodos que van a utilizarla (el método para listar los alumnos en la vista del controlador y el método Plugin para cambiar la nota minima).

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

No me ha dado a tiempo a terminarlo, porque en mi portátil no se crean los logs, tengo que realizarlo desde la máquina de labs.pue y a las horas que puedo hacer el examen, no están disponibles.
Para realizarlo sin crear clases hay que introducir los virtualtypes en di.xml. Hay 3 bloques principales. El primero donde se indica el fichero log a utilizar, otros dos que relacionan en handler y el logger, con los virtualtypes.

Por otra parte, hay que inyectar el loggerinterface en la clase que vaya a utilzarse el log. Y escribir en el log. Esta parte es la que creo que no he terminado, ya que el log me lo sigue escribiendo en los ficheros predeterminados de magento en vez de en el mío.