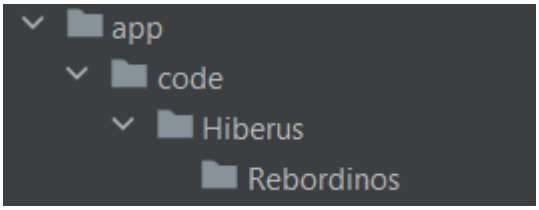
Contenido

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

Para hacer un nuevo módulo primero crearemos las carpetas de nuestro vendor y de nuestro módulo dentro de *app/code*, quedando la estructura de la siguiente manera:



Ahora necesitamos los archivos registration.php y module.xml.

El archivo **registration.php** está ubicado en la raíz del módulo y en este se refleja el nombre del módulo, en este caso Hiberus_Rebordinos.

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Hiberus_Rebordinos',
    __DIR__
);
```

El segundo archivo necesario es **module.xml**, situado en *app/code/Hiberus/Rebordinos/etc*, este contiene los datos del nombre del módulo, la versión y las dependencias.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
"../../../../../lib/internal/Magento/Framework/Module/etc/module.xsd">
    <module name="Hiberus_Rebordinos" setup_version="2.0.1">
    <sequence>
        <module name="Magento_Theme"/>
    </sequence>
    </module>
</config>
```

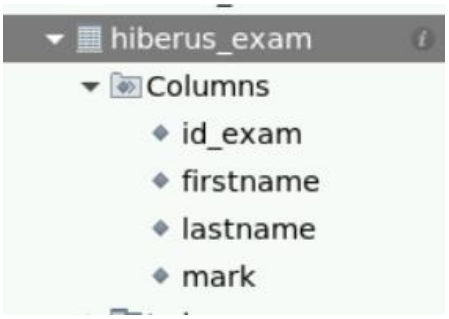
2 - Crear una única tabla llamada hiberus_exam que responda exactamente a la siguiente estructura:

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 una tabla necesitamos meter la estructura de datos en el archivo **db_schema.xml** que se encuentra en app/code/Hiberus/Rebordinos/etc

```xml
<?xml version="1.0"?>
<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="urn:magento:framework:Setup/Declaration/Schema/etc/schema.xsd">
    <table name="hiberus_exam" resource="default" engine="innodb" comment="Tabla Examen Hiberus">
        <column xsi:type="int" name="id_exam" padding="11" unsigned="true" nullable="false" identity="true"/>
        <column xsi:type="varchar" name="firstname" length="100" unsigned="true" nullable="false"/>
        <column xsi:type="varchar" name="lastname" length="250" unsigned="true" nullable="false"/>
        <column xsi:type="decimal" name="mark" scale="2" precision="4" nullable="false"/>
        <constraint xsi:type="primary" referenceId="PRIMARY">
            <column name="id_exam"/>
        </constraint>
    </table>
</schema>
```

Ejecutamos el comando de dockergento magento setup:upgrade y comprobamos en nuestro gesto de bases de datos, en este caso MySQL Workbench, que se ha creado la tabla correctamente:



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

Para crear el Service Contracts necesitamos los crear una interfaz de programación de aplicaciones (API) con el conjunto de interfaces y sus implementaciones que un módulo proporciona a otros módulos.

Primero necesitamos crear la interfaz del repositorio **ExamRepositoryInterface.php** en app\code\Hiberus\Rebordinos\Api que contiene la colección de métodos de getById, save, delete y deleteById.

```php
interface ExamRepositoryInterface
{
    /**
     * @param int $entityId
     * @return ExamInterface
     */
    public function getById(int $entityId)

    /**
     * @param ExamInterface $exam
     * @return ExamInterface
     */
    public function save(ExamInterface $exam)

    /**
     * @param ExamInterface $exam
     * @return void
     */
    public function delete(ExamInterface $exam);

    /**
     * @param int $entityId
     * @return ExamInterface
     */
    public function deleteById(int $entityId)
```

A continuación, creamos la interfaz **ExamInterface.php** dentro de app\code\Hiberus\Rebordinos\Api\Data, esta contiene la colección de *setters y getters* y dos constantes, una con el nombre de la tabla y la otra con el nombre de nuestra columna ID.

```php
interface ExamInterface extends \Magento\Framework\Api\ExtensibleDataInterface
{

    const TABLE_NAME = "hiberus_exam";
    const COLUMN_ID = "id_exam";

    /**
     * @return int
     */
    public function getEntityId();

    /**
     * @param int $entityId
     * @return $this
     */
    public function setEntityId($entityId);

    /**
     * @return string
     */
    public function getFirstname();

    /**
     * @param string $firstname
     * @return $this
     */
    public function setFirstname($firstname);
```

El siguiente paso es crear los modelos para poder gestionar el ORM:

El primero en crearse es **Exam.php** en app\code\Hiberus\Rebordinos\Model\ResourceModel que va a ser nuestro modelo de los recursos que extiende de la clase AbstractDb y contiene los métodos save/load/delete

```php
/**
 * @param AbstractModel $object
 * @return $this|AbstractDb
 * @throws \Exception
 */
public function save(AbstractModel $object) {
    $this->entityManager->save($object);
    return $this;
}

/**
 * @param AbstractModel $object
 * @param mixed $value
 * @param null $field
 * @return AbstractDb|mixed
 */
public function load(AbstractModel $object, $value, $field = null) {
    return $this->entityManager->load($object, $value);
}

/**
 * @param AbstractModel $object
 * @return AbstractDb|void
 * @throws \Exception
 */
public function delete(AbstractModel $object) {
    $this->entityManager->delete($object);
}
```

El segundo archivo es **Exam.php**, pero esta vez se encuentra en app\code\Hiberus\Rebordinos\Model y extiende de AbstractModel y se implementará los métodos que antes se definieron en **ExamInterface.php** con los getters y setters para manejar los datos:

```php
class Exam extends AbstractModel implements ExamInterface
{

    protected function _construct()
    {
        $this->_init(\Hiberus\Rebordinos\Model\ResourceModel\Exam::class );
    }

    /**
     * @inheriDoc
     */
    public function getEntityId()
    {
        return $this->getData( string: 'id_exam');
    }

    /**
     * @inheriDoc
     */
    public function setEntityId($entityId)
    {
        return $this->setData( string: 'id_exam',$entityId);
    }
```

Y por último se creará en app\code\Hiberus\Rebordinos\Model la clase **ExamRepository.php** que implementará los métodos de **ExamRepositoryInterface.php.**

```php
     * @param $entityId
     * @return mixed
     */
    public function getById($entityId)
    {
        try {
            $exam = $this->ExamInterfaceFactory->create();
            $exam->setEntityId($entityId);
            $this->resourceExam->load($exam, $entityId);
        } catch (\Exception $e) {
            $exam = $this->ExamInterfaceFactory->create();
        }

        return $exam;
    }

    /**
     * @param ExamInterface $exam
     * @return bool
     */
    public function delete(ExamInterface $exam)
    {
        try {
            $this->resourceExam->delete($exam);
        } catch (\Exception $e) {
            return false;
        }
        return true;
    }
```

Para los métodos de esta clase se necesitará hacer uso de **ExamInterfaceFactory**, esta clase se crea automáticamente al ejecutar el comando de dockergento magento setup:upgrade si tenemos todos los archivos correctamente creados.

Tras crear todos los archivos necesarios es muy importante añadir app/code/Hiberus/Rebordinos/etc el archivo **di.xml** la implementación de la interfaz con las etiquetas <preferencia/> y el nombre de la tabla y la columna ID.

```xml
<!-- Relacion de la interfaz con el modeloy la interfaz del repositorio con la el repositorio -->
<preference for="Hiberus\Rebordinos\Api\Data\ExamInterface" type="Hiberus\Rebordinos\Model\Exam"/>
<preference for="Hiberus\Rebordinos\Api\ExamRepositoryInterface" type="Hiberus\Rebordinos\Model\ExamRepository"/>

<!--Indica la tabla y la columna id con la que estamos trabajando-->
<type name="Magento\Framework\EntityManager\MetadataPool">
    <arguments>
        <argument name="metadata" xsi:type="array">
            <item name="Hiberus\Rebordinos\Api\Data\ExamInterface" xsi:type="array">
                <item name="entityTableName" xsi:type="const">Hiberus\Rebordinos\Api\Data\ExamInterface::TABLE_NAME</item>
                <item name="identifierField" xsi:type="const">Hiberus\Rebordinos\Api\Data\ExamInterface::COLUMN_ID</item>
            </item>
        </argument>
    </arguments>
</type>
```

Para en un futuro poder manejar todos los datos de la tabla necesitamos crear un **Controller.php** dentro de la ruta app/code/Hiberus/Rebordinos/Model/ResourceModel/Exam donde en el constructor asociamos el modelo con la clase del ResourceModel.

```php
/**
 * Define resource model
 * @return void
 */
protected function _construct()
{
    $this->_init('Hiberus\Rebordinos\Model\Exam', 'Hiberus\Rebordinos\Model\ResourceModel\Exam');
}
```

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.

No hemos dado los como crear Db Schema y Data Patch, la manera que tendríamos de meter datos a nuestra tabla es a través de un controlador y el Factory, introduciendo la información de dos arrays con datos, uno con nombres y otro con apellidos y que el método vaya seleccionando al azar uno de cada.

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.

Para crear un Front controller es necesario tener un archivo de rutas **routes.xml**, en nuestro caso estará definido en app/code/Hiberus/Rebordinos/etc/frontend.

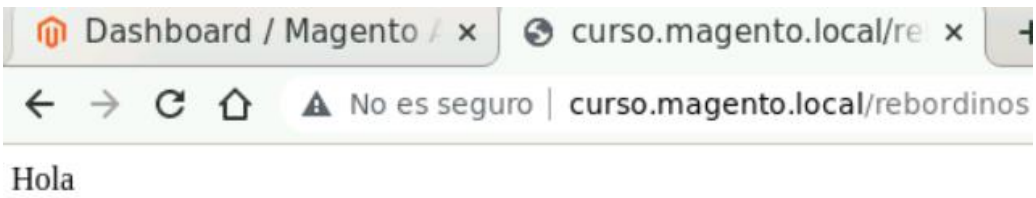
Dentro indicamos el nombre de nuestro módulo y le asignamos tanto el frontName como la id:

```xml
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:App/etc/routes.xsd">
    <router id="standard">
        <route frontName="rebordinos" id="rebordinos">
            <module name="Hiberus_Rebordinos"/>
        </route>
    </router>
</config>
```

El siguiente paso es crear un controlador en app/code/Hiberus/Rebordinos/etc/frontend/Index al que llamaremos **Index.php** que implementará de HttpGetActionInterface. Dentro del método execute() escribiremos el echo “Hola” y pondremos un die() antes del return ya que solo queremos comprobar que se ejecuta porque que aún no hemos creado ningún layouts, ni templates o bloque.

```php
public function execute()
{
    echo "Hola";
    die();
    return $this->pageFactory->create();
}
```

El resultado al poner la ruta /rebordinos en el navegador es que se ejecuta correctamente el controlador y nos muestra el echo “Hola”:



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."

Creamos primero el bloque en un archivo con nombre **Index.php** en app/code/Hiberus/Rebordinos/Block, esta clase extiende de Template y crearemos un método para llamar un array con todos los datos de los exámenes.

```php
/**
 * @return AbstractDb|AbstractCollection|null
 */
public function getExams() {
    $resultPage = $this->examInterfaceFactory->create();
    return $resultPage->getCollection();
}
```

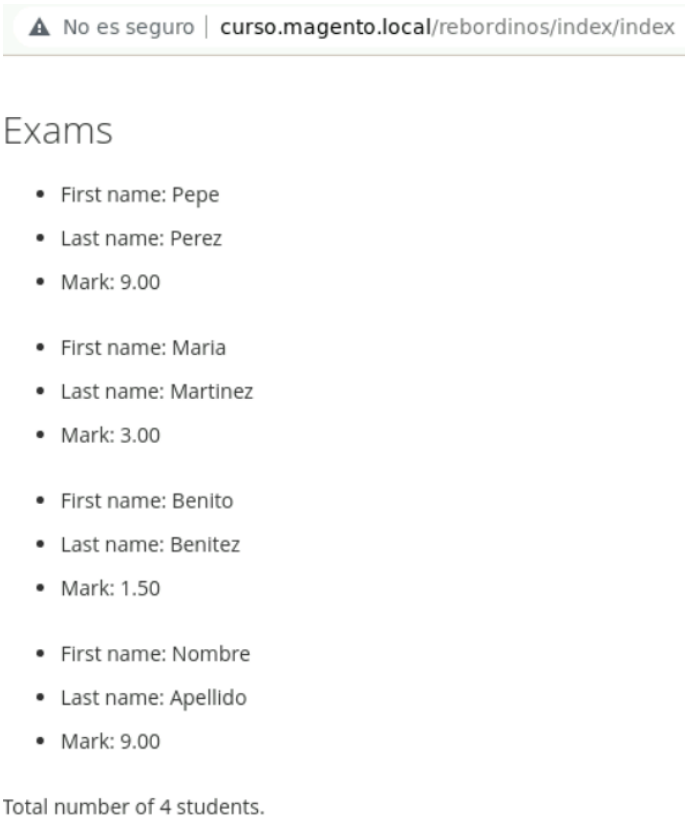
Lo siguiente es crear la plantilla **index.phtml** en app/code/Hiberus/Rebordinos/view/frontend/templates que será el html que mostraremos en el bloque.

```php
<?php

/**
 * @var Index $block
 */

use Hiberus\Rebordinos\Block\Index;

$exams = $block->getExams();

?>

<h2 class="title">Exams</h2>
<?php foreach ($exams as $item) {?>
    <ul>
        <li>First name: <?= $item->getFirstname() ?></li>
        <li>Last name: <?= $item->getLastname() ?></li>
        <li>Mark: <?= $item->getMark() ?></li>
    </ul>
<?php } ?>

<p><?= __('Total number of %1 students', [sizeof($exams)]) ?></p>
```

Y por último creamos el layout en app/code/Hiberus/Rebordinos/view/frontend/layout con el nombre de **rebordinos_index_index.xml** donde definimos el bloque con su clase, nombre y la template utilizada.

```xml
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:View/Layout/etc/page_configuration.xsd">
    <referenceContainer name="content">
        <block class="Hiberus\Rebordinos\Block\Index" name="rebordinos_index_index" template="Hiberus_Rebordinos::index.phtml" />
    </referenceContainer>
</page>
```

Así nos quedaría antes de la traducción:



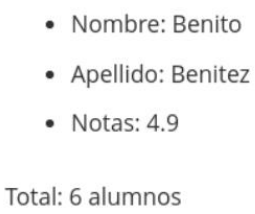
No hemos dado las traducciones, sin embargo, es tan sencillo como crear un archivo **es_ES.csv** la carpeta app/code/Hiberus/Rebordinos/i18n y modificar el archivo **index.phtml** que nos muestra el texto que deseemos traducir.

Este texto va dentro del archivo .csv:

```
"Total number of %1 students", "Total: %1 alumnos"
```

Y para mostrar esta traducción hay que cambiar la configuración del idioma desde el panel del administrador *Stores → Settings → Configuration → General → Locale Options.*

Y así cargaría el texto en español en la vista:



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

No hemos dado js, sin embargo, mirando la documentación se puede añadir de manera sencilla. Primero creamos el archivo de configuración **requirejs-config.js** en *app/code/Hiberus/Rebordinos/view/frontend* con lo siguiente:

```
var config = {
    map: {
        '*': {
            myscript: 'Hiberus_Rebordinos/js/myfile',
        }
    }
};
```

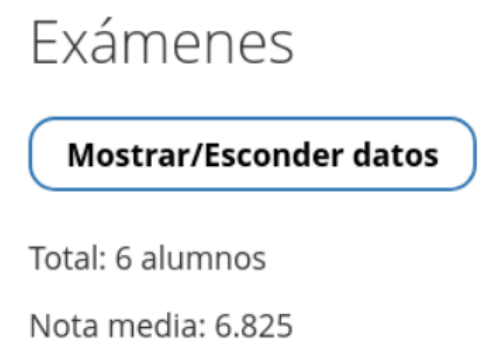
Después creamos nuestro fichero **myfile.js** en *app/code/Hiberus/Rebordinos/view/frontend/web/js/* con la función que queremos realizar, en este caso que oculte o se muestre un div en cuanto se pulse a un botón:

```
define(['jquery'], function($){
    "use strict";
    return function myscript(idBtn,idDiv)
    {
        $(idBtn).click(function(){
            $(idDiv).slideToggle();
        });
    }
});
```

Una vez hecho esto ya podemos dirigirnos a nuestra plantilla **index.phtml**, añadir el botón y el siguiente bloque de script:

```
<script>
    require(['jquery', 'myscript'], function($, myscript) {
        myscript('#open_details','#details');
    });
</script>
```

Así se vería la tabla cuando se pulsa al botón:



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.

Para poder añadir los archivos Less primero hay que asignar un tema en el que se vean reflejados los cambios que personalicemos, en este caso el tema se llamará Alicia y necesitamos crear la siguiente ruta de carpetas: app/desing/frontend/Rebordinos/alicia y necesitamos dos archivos.

El primero a crear, en la ruta anterior, es el **theme.xml**, dentro de él se indica el nombre del tema y la plantilla padre, en este caso Magento/blank.

```xml
<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
    "urn:magento:framework:Config/etc/theme.xsd">
    <title>Alicia</title>
    <parent>Magento/blank</parent>
</theme>
```

En la misma ubicación creamos el archivo **registration.php** con los datos de la dirección del nuevo tema.

```php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */

use \Magento\Framework\Component\ComponentRegistrar;

ComponentRegistrar::register(ComponentRegistrar::THEME, 'frontend/Rebordinos/alicia', __DIR__);
```

El archivo **composer.json** es opcional, dentro van las dependencias necesarias para cargar el tema.

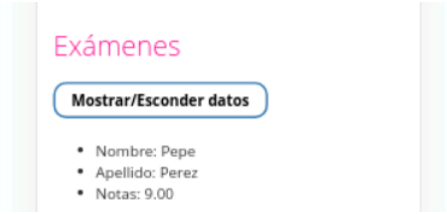
Dentro del panel de administración debemos de seleccionar nuestro tema en *Admin → Content → Design → Configuration*

Como queremos modificar los elementos que están dentro de un Container del tema padre Magento_Theme, será necesario crear el archivo **_extend.less** dentro de la siguiente ruta app/desing/frontend/Rebordinos/alicia/Magento_Theme/web/css/source

```less
& when (@media-common = true) {
    .page-main {
        .title {
            color:deeppink;
        }
    }
}

.media-width(@extremum, @break) when (@extremum = 'min') and (@break = @screen__m) {
    .page-main {
        .title {
            color:dodgerblue;
        }
    }
}
```

El titulo por defecto aparecerá de color azul hasta que la resolución baje de 768 píxeles:

En cuanto se cambie de resolución se aplicará el otro por defecto:



Para crear un mixin (funciones) para aplicar los css con variables, cremamos el archivo **_mixin.less** en app/desing/frontend/Rebordinos/alicia/ /web/css/source

```
.margen(@margin-left-primary){
    margin-left: @margin-left-primary;
}
```

Y para poder reutilizarlo desde cualquier módulo lo importamos en el archivo **_extend.less** dentro de la misma ruta.

```
@import "_mixin.less";
```

Luego volvemos a editar el archivo dónde habíamos cambiado el color del título y ponemos la función y los estilos de la lista:

```
& when (@media-common = true) {
    .page-main {
        .title {
            color:deeppink;
        }
        ul{
            list-style: circle;
        }
        ul:nth-child(even){
            .margin(20px);
        }
    }
}
```

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)*

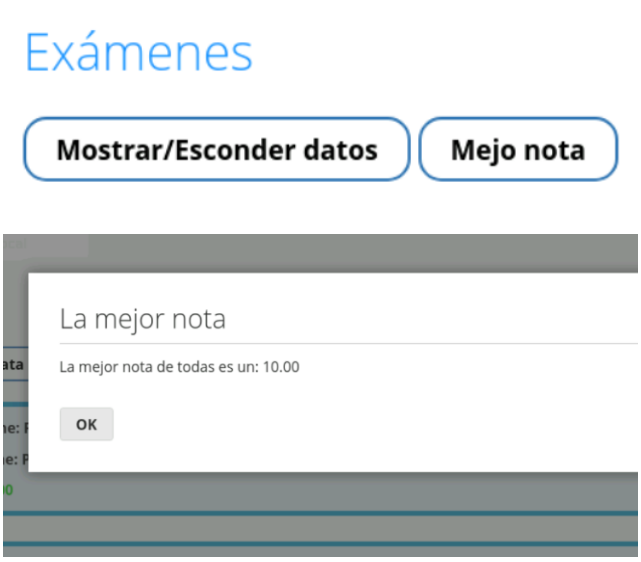
No hemos dado *jQuery widget*, pero siguiendo la documentación hay que crear simplemente un script en nuestro template **index.phtml** donde en el require hay que especificar que el alert.js se encuentra dentro de Magento_Ui/js/alert para que utilice el *jQuery widget.*

```
<script>
    require([
            'jquery',
            'Magento_Ui/js/modal/alert'
        ], function($, alert) {
            $('#open_max').on('click', function(event){
                alert({
                    title: 'La mejor nota',
                    content: $('<div>La mejor nota de todas es un: <?= $topMarks[0] ?></div>'),
                    modalClass: 'alert',

                })
            })
        }
    );
</script>
```

La variable $topMarks es un método del que se mostrará más adelante en el ejercicio 13, el cual recoge las 3 mejores notas ordenadas de mayor a menor, por lo tanto en $topMarks[0] obtenemos la mejor nota de la clase.

También en el mismo archivo añadimos el nuevo botón con la id "open_max" y al pulsar en él se nos despliega el mensaje de alerta en la vista:

```
<button id="open_max" ><?= __('Top Mark') ?></button>
```



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

Para mostrar una nueva fila creamos un nuevo método que saque la nota media de la colección de notas:

```php
public function getAverageMark(){
    $exams = $this->getExams();
    $marks=[];
    foreach ($exams as $item){
        $marks[]=$item->getMark();
    }
    return array_sum($marks)/count($exams);
}
```

Actualizamos el archivo **index.phtml** de app/code/Hiberus/Rebordinos/view/frontend/templates y añadimos la nueva fila:

```php
<p><?= __('The average mark: %1', [$block->getAverageMark()]) ?></p>
```

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

Para definir un plugin creamos una nueva carpeta en app/code/Hiberus/Rebordinos/Plugin/Exams y dentro de esta creamos el archivo **MarkPlugin.php**, dónde estarán los métodos que necesitemos.

El método utilizado es un método de tipo after, que carga el objeto Exam y dependiendo de la nota, a los suspensos se les asignará la nueva nota 4.9.

```
public function afterGetMark(
    Exam $subject,
        $result
)
{
    if ($subject->getData( key: 'mark') < 5.0) {
        return 4.9;
    }
    return $result;
}
```

Debemos declarar el pugin dentro de nuestro archivo **di.xml**, se le asigna un nombre único, el modelo y la ubicación del plugin de la siguiente manera:

```
<!--Necesario para el Plugin-->
<type name="Hiberus\Rebordinos\Model\Exam">
    <plugin name="exam_plugin" type="Hiberus\Rebordinos\Plugin\Exams\MarkPlugin" sortOrder="10"  />
</type>
```

En la vista aparece de la siguiente manera:

- Nombre: Alfredo
- Apellido: Altroz
- Notas: 4.9

- Nombre: Benito
- Apellido: Benitez
- Notas: 4.9

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)

Creamos las funciones de los colores de los alumnos en el archivo en **_mixin.less** que ya teníamos en la ruta app/desing/frontend/Rebordinos/alicia/ /web/css/source

```
.color-pass(@color-pass){
    color: @color-pass;
}

.color-fail(@color-fail){
    color: @color-fail;
}
```

Para comprobar cuales son los registros de alumnos suspensos, en la plantilla **index.phtml** de app/code/Hiberus/Rebordinos/view/frontend/templates comprobamos las notas y dependiendo de si está aprobado o suspenso, añadimos una clase distinta.

```
<?php if($item->getMark() < 5){
    $class="exam-fail";
}else{
    $class="exam-pass";
} ?>
<li class="<?=$class?>"><?=$item->getMark()?></li>
```

Configuramos el archivo con nuestros estilos de **_extend.less:**

```
.exam-pass{
    .color-pass(#36d536);
}
.exam-fail{
    .color-fail(#b00707)
}
```

Así se muestra en la vista:

- Nombre: Carmen
- Apellido: Corona
- Notas: 8.75

    - Nombre: Guillermo
    - Apellido: Gutierrez
    - Notas: 7.25

- Nombre: Alfredo
- Apellido: Altroz
- Notas: 4.9

    - Nombre: Benito
    - Apellido: Benitez
    - Notas: 4.9

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

Creamos un nuevo método que saque todas las notas de los alumnos y lo ordenamos de mayor a menor y se cogen solo las 3 primeras notas.

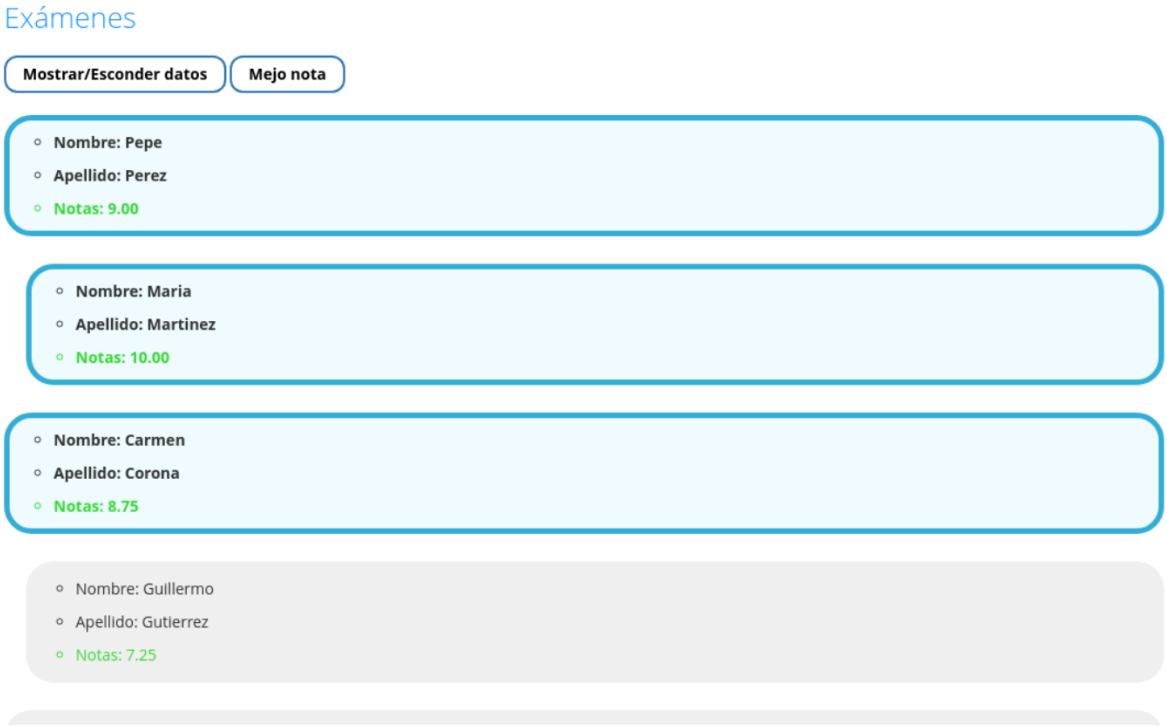
```php
public function getMaxMarks()
{
    $exams = $this->getExams();
    $marks=[];
    foreach ($exams as $item){
        $marks[]=$item->getMark();
    }
    rsort( &array: $marks);
    return array_slice($marks, offset: 0, length: 3);
}
```

Llamamos este método en nuestra template **index.phtml** y comparamos si la nota del alumno está dentro de las 3 mejores y le añadimos una nueva clase a la lista para poder destacarlo con css.

```php
<?php if(in_array($item->getMark(),$topMarks) ){?>
    <ul class="best">
        <?php }else{?>
    <ul>
<?php   } ?>
```

Añadimos estilos para que se destaquen en el fichero **_extend.less** de la ruta de nuestros estilos app/desing/frontend/Rebordinos/alicia/Magento_Theme/web/css/source

Así quedaría finalmente:



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).

Creamos un archivo llamado **ExamsCommand.php** en la ruta *app/code/Hiberus/Rebordinos/Console*, lo configuramos con el nombre de hiberus:rebordinos y le añadimos la descripción.

```
protected function configure()
{
    $this->setName( name: 'hiberus:rebordinos')
        ->setDescription( description: 'Muestra los datos de la tabla de examenes');

    parent::configure();
}
```

En el execute creamos el método que devolverá toda la información:

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    $resultPage = $this->examInterfaceFactory->create();
    $exams = $resultPage->getCollection();

    foreach ($exams as $item){
        $this->exam->setEntityId($item->getEntityId());
        $this->exam->setFirstname($item->getFirstname());
        $this->exam->setLastname($item->getLastname());
        $this->exam->setMark($item->getMark());
        $output->writeln( messages: '<info> ID: ' . $this->exam->getEntityId()
            ' | First name: ' . $this->exam->getFirstname() .
            ' | Last name: ' . $this->exam->getLastname() .
            ' | Mark: ' . $this->exam->getMark() . '</info>');
    }
}
```

Y modificamos el archivo **di.xml** para crear el comando y ejecutamos los comandos dockergento magento setup:di:compile, dockergento magento cache:flush.

```
<!--Necesario para el comando-->
<type name="Magento\Framework\Console\CommandListInterface">
    <arguments>
        <argument name="commands" xsi:type="array">
            <item name="comand_rebordinos" xsi:type="object">Hiberus\Rebordinos\Console\ExamsCommand</item>
        </argument>
    </arguments>
</type>
```

Al ejecutar nuestro nuevo comando dockergento magento hiberus:rebordinos nos muestra lo siguiente:

```
pue@pue-KVM:~/Proyectos/curso-magento$ dockergento magento hiberus:rebordinos
 ID: 1 | Nombre: Pepe | Apellidos: Perez | Nota: 9.00
 ID: 2 | Nombre: Maria | Apellidos: Martinez | Nota: 3.00
 ID: 3 | Nombre: Carmen | Apellidos: Corona | Nota: 2.60
 ID: 4 | Nombre: Guillermo | Apellidos: Gutierrez | Nota: 8.75
 ID: 5 | Nombre: Alfredo | Apellidos: Altroz | Nota: 5.00
 ID: 6 | Nombre: Benito | Apellidos: Benitez | Nota: 1.50
```

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

No hemos dado los endpoint.

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)

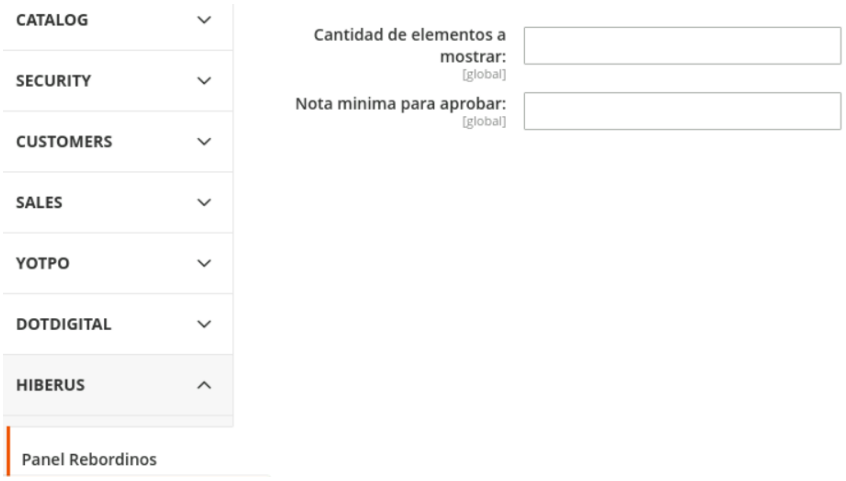
Necesitaremos crear los elementos que necesitemos en el panel un archivo **system.xml** dentro de *app/code/Hiberus/Rebordinos/etc/adminhtml*

```xml
<system>
       <tab id="hiberus" sortOrder="999" translate="label">
         <label>Hiberus</label>
       </tab>
       <section id="hiberus_nombre" showInDefault="1"  showInStore="1" showInWebsite="1" sortOrder="10">
           <label>Panel Rebordinos</label>
           <tab>hiberus</tab>
           <resource>Hiberus_Rebordinos::config</resource>
           <group id="general" showInDefault="1" sortOrder="10">
               <label>General</label>
               <field id="cantidad" type="text" sortOrder="10" showInDefault="1">
                   <label>Cantidad de elementos a mostrar:</label>
                   <validate>integer</validate>
               </field>

               <field id="notaMinima" type="text" sortOrder="10" showInDefault="1">
                   <label>Nota minima para aprobar: </label>
                   <validate>validate-number</validate>
               </field>
           </group>
       </section>
   </system>
```

Modificamos el archivo **di.xml** para que cargue el tema del administrador:

```xml
<!--  Cargamos el tema del administrador  -->
<type name="Magento\Theme\Model\View\Design">
    <arguments>
        <argument name="themes" xsi:type="array">
            <item name="adminhtml" xsi:type="string">Rebordinos/alicia_admin</item>
            <!--  Example: "Magento/backend"  -->
        </argument>
    </arguments>
</type>
```

Tras esto, nos aparecerá en *Stores → Settings → Configuration → Hiberus →Panel Rebordinos* de la siguiente manera:



Teniendo el panel ya configurado, para utilizar estos datos, tenemos que manejarlos con la clase scopeConfg:

```php
public function getPassNote(){
    $data= $this->scopeConfig->getValue( 'hiberus_nombre/general/notaMinima', ScopeInterface::SCOPE_STORE);
    return  $data?:5;
}

public function getSize(){
    $data= $this->scopeConfig->getValue( 'hiberus_nombre/general/cantidad', ScopeInterface::SCOPE_STORE);
    $exams = $this->getExams();
    return  $data?:count($exams);
}
```

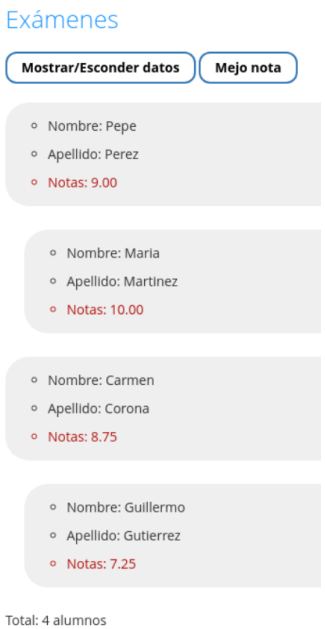
Y modificamos nuestra plantilla **index.phtml** para utilizar esta nueva información y que se muestre en la vista:

```php
$exams = $block->getExams();
$passnote= $block->getPassNote();
$size= $block->getSize();
```

```php
<?php foreach ($exams as $item) {
    if ($i++ == $size) break;?>
    <ul>
        <li><?= __('First name') ?>: <?= $item->getFirstname() ?></li>
        <li><?= __('Last name') ?>: <?= $item->getLastname() ?></li>

        <?php if($item->getMark() < $passnote){
            $class="exam-fail";
        }else{
            $class="exam-pass";
        } ?>
        <li class="<?=$class?>"><?= __('Mark') ?>: <?=$item->getMark()?></li>
    </ul>
```

Vamos a mostrar un ejemplo, en el panel ponemos los siguientes datos:

Y en la vista nos muestra en rojo aquellos que tienen la nota menor a 9,5 y solo aparecen 4 registros.



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.

No lo hemos dado, pero es tán sencillo como crear el método que escriba en el archivo de esta manera y ejecutarlo a cada vez que carge la página.

```php
public function writeLog(){
    $writer = new \Zend\Log\Writer\Stream( string: BP . '/var/log/mylogger.log');
    $logger = new \Zend\Log\Logger();
    $logger->addWriter($writer);
    $logger->info( string: 'Se van a mostrar '.$this->getSize().
        ' alumnos y la nota media de todos es '.$this->getAverageMark());
}
```

El resultado se encuentra en /var/log/mylogger.log y es el siguiente:

mylogger.log

```
2021-09-24T15:47:16+00:00 INFO (6): Se van a mostrar 8 alumnos y la nota media de todos es 6.825
2021-09-24T15:57:47+00:00 INFO (6): Se van a mostrar 1 alumnos y la nota media de todos es 9
```