DaCodes.	**Desafío para puesto de desarrollador BackEnd**	

1.- Documento de desafío.

Descripción

Necesitamos desarrollar una API para que los cursos de e-learning se integren en nuestro sistema. El propósito de esta herramienta es que nosotros, como profesores, administremos la configuración de los cursos y revisiones de desempeño y, para nuestros estudiantes, tomar cursos cuando utilicen nuestra interfaz. Nuestro PM es una persona muy ocupada, por lo que no tenemos tareas detalladas, sino solo las reglas comerciales con las que trabajar. Aquí están:

Instrucciones:

1. Tenemos cursos que contienen lecciones y lecciones que contienen preguntas.
2. Los cursos son correlativos con los anteriores.
3. Las lecciones son correlativas con las anteriores.
4. Las preguntas de cada lección no tienen correlación.
5. Todas las preguntas de una lección son obligatorias.
6. Cada pregunta tiene una puntuación.
7. Cada lección tiene una puntuación de aprobación que debe cumplirse con la suma de las preguntas respondidas correctamente para aprobarla.
8. Se aprueba un curso cuando se aprueban todas las lecciones.
9. No hay restricciones para acceder a los cursos aprobados.
10. Solo los profesores pueden crear y administrar cursos, lecciones y preguntas.
11. Cualquier alumno puede realizar un curso.
12. Inicialmente, necesitaremos responder a este tipo de preguntas:
    - Booleano
    - Opción múltiple donde solo una respuesta es correcta
    - Opción múltiple donde más de una respuesta es correcta
    - Opción múltiple donde más de una respuesta es correcta y todas deben responderse correctamente
13. Los chicos de frontend solicitaron específicamente estos puntos finales para que los estudiantes los usen:
    - Obtenga una lista de todos los cursos, indicando a cuáles puede acceder el estudiante
    - Obtenga lecciones para un curso, indicando a cuáles puede acceder el estudiante
    - Obtenga detalles de la lección para responder sus preguntas
    - Tome una lección (para evitar varias solicitudes, pidieron enviar todas las respuestas de una vez
    - CRUD básico para cursos, lecciones y preguntas

Reglas de la base de código:

1. La API debe desarrollarse utilizando Laravel
2. Debe haber un archivo Léame que documente la instalación y el uso.
3. Puede utilizar los marcos y bibliotecas que desee, pero deben incluirse en el archivo Léame que documente su propósito y una breve explicación con el razonamiento de su elección.

2.- Prerrequisitos

a) Tener instalado PHP y Laravel

b) Desde la terminal instalar Composer

* Comando: composer create-project --prefer-dist laravel/ dacodes-api

c) Tener instalado MySql o alguna base de datos.

d) Configurar el archivo .env con la base de datos a usar

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=auth
DB_USERNAME=root
DB_PASSWORD=
```

e) Instalar el paquete Laravel Passport para la autenticación.

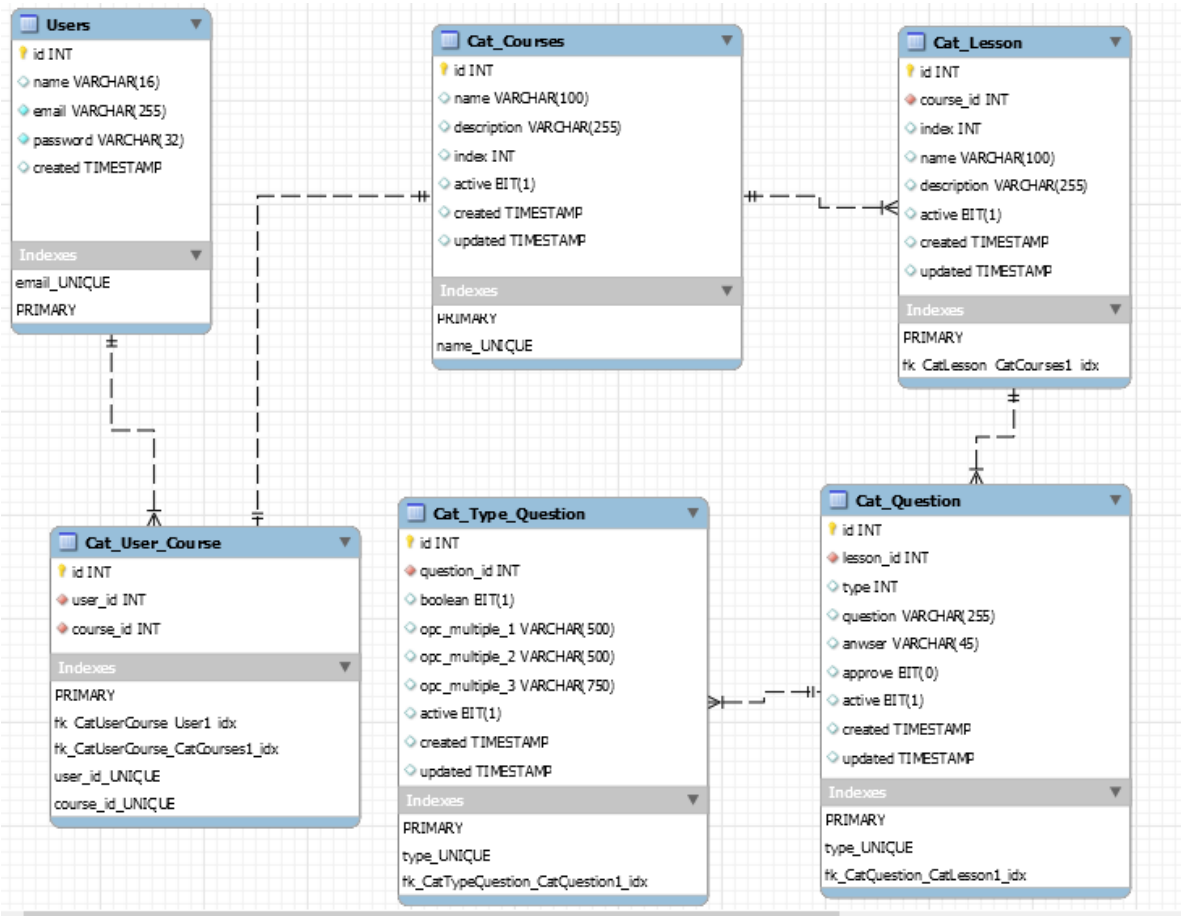
* Comando: composer require laravel/Passport
* Ejecutar las migraciones para que se creen las tablas
* Comando: php artisan migrate
* Generar las llaves, comando: `php artisan passport:install`

f) Instalar el paquete caffeinated shinobi para los roles y permisos

* Comando: composer require caffeinated/shinobi
* Comando: php artisan vendor:publish –
provider="Caffeinated\Shinobi\ShinobiServiceProvider" --tag="config"

3.- Modelo de Base de datos.

Las tablas y las relaciones se crean desde Laravel con las migraciones.



Nota: El modelo fue creado en MySql WorkBench

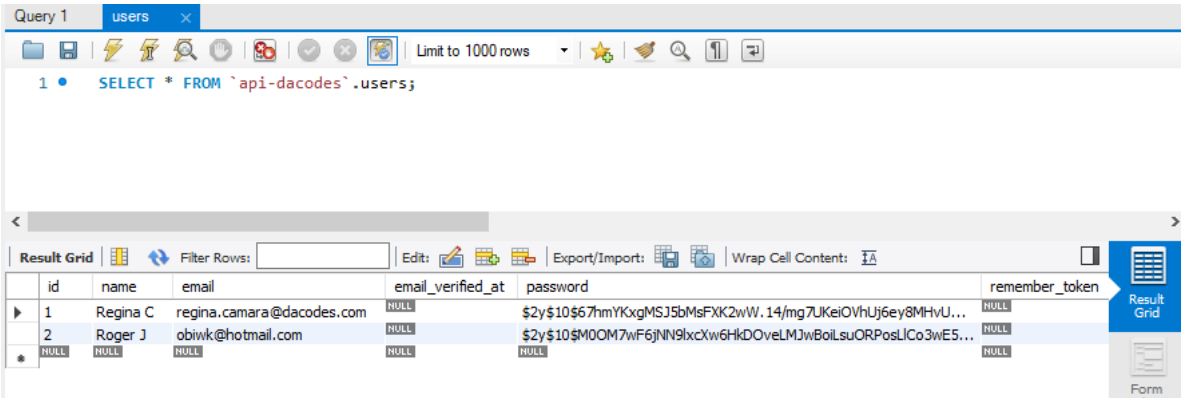
4.- Creación de usuarios y roles y permisos

Para esta prueba se creó un alumno y un estudiante, usando postman y mysql.

Crear usuario	
http://localhost:8000/api/auth/signup	POST

Request	{
	"name": "Regina C",
	"email": "regina.camara@dacodes.com",
	"password": "12345"
	}

Response	Successfully created user!



DaCodes.	**Desafío para puesto de desarrollador BackEnd**	

Login	
http://localhost:8000/api/auth/login	POST

Request	{
	"email": "obiwk@hotmail.com",
	"password": "12345",
	"remember_me": true
	}

Response	{
	"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIxIiwianRpIjoiNjYzOTIwMTkyNTM0ZmIyYTY4YTRiYjY5OGRlN2ZiNmFjNjgwZWQ3MjBjNDUzZjkzZDkwMDYxMjc4MzAwYWY3ZjAyMjQ3OTNhMGFkODJlOWMiLCJpYXQiOjE2MDIzNzM3ODAsIm5iZiI6MTYwMjM3Mzc4MCwiZXhwIjoxNjMzOTA5NzgwLCJzdWIiOiIyIiwic2NvcGVzIjpbXX0.OQV5bwe4K9bTHs91fmhI6DlafQzaUUsM6aM1Xy0DKej9Sw1vuu7CSESEJcog0vLyn8Wwt-xMJfpu6F9MOA9bIMY5BuOjKIWJdSPcZ0W0mM7IGra1FcA-1A4hdAgyUpqRqW5WDDXg1QELGpal0rxkX-AvsFuiXcUvJwad_dA7YGeV2UXRQ4SpQBm9oih-i7kM52B2V3G8KGNsIy3toDOEOETxolPv0MDUo9NA9xJVqZBy9q2DmHOW9ZT7CR8zfgzI0NR5kowZ0RwCBw1KCMW1tj_TaWhLuj_j78qWZETEs7tH4UNpSN6H8Z-QgSiN41-AHOK1gY5tmdVFAYJdwCU0RzLqIfVgRl-p3W96GuKDqHOgfgU2oMZxxFP7rvFV5BKKbiZP6dLaQiS_2gm_bQ5_qJwnUE14bvB-YHHOYY4o8rj9FB7A_45Ya5BCccoY_Sj9L9ASUda35KhCGam7sPsOwzsR-RN3sTha52xfoa4cXXCKvTV64p9nwYgMpgYYueobrM7r8XwNPTOshJDtUIELx2vakCfXRYwPjWScuF2-JOoUUyHQJMLNlVZ2JF7qoHeGLHrDkvesNxZREA_FH8-4lOu4Uvg6dS0S1FB1RF8o8U_xVjUHc4A2-khwGGef7BVzhte3frNTbOIG3SvXKEzDo1LMljnoQZLW6vKN6eF7ICc"
	,
	"token_type": "Bearer",
	"expires_at": "2020-10-17 23:49:41"
	}

Logout	
http://localhost:8000/api/auth/logout	GET
Header -> Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIxIiwianRpIjoiNjYzOTIwMTkyNTM0ZmIyYTY4YTRiYjY5OGRlN2ZiNmFjNjgwZWQ3MjBjNDUzZjkzZDkwMDYxMjc4MzAwYWY3ZjAyMjQ3OTNhMGFkODJlOWMiLCJpYXQiOjE2MDIzNzM3ODAsIm5iZiI6MTYwMjM3Mzc4MCwiZXhwIjoxNjMzOTA5NzgwLCJzdWIiOiIyIiwic2NvcGVzIjpbXX0.OQV5bwe4K9bTHs91fmhI6DlafQzaUUsM6aM1Xy0DKej9Sw1vuu7CSESEJcog0vLyn8Wwt-xMJfpu6F9MOA9bIMY5BuOjKIWJdSPcZ0W0mM7IGra1FcA-1A4hdAgyUpqRqW5WDDXg1QELGpal0rxkX-AvsFuiXcUvJwad_dA7YGeV2UXRQ4SpQBm9oih-i7kM52B2V3G8KGNsIy3toDOEOETxolPv0MDUo9NA9xJVqZBy9q2DmHOW9ZT7CR8zfgzI0NR5kowZ0RwCBw1KCMW1tj_TaWhLuj_j78qWZETEs7tH4UNpSN6H8Z-QgSiN41-AHOK1gY5tmdVFAYJdwCU0RzLqIfVgRl-p3W96GuKDqHOgfgU2oMZxxFP7rvFV5BKKbiZP6dLaQiS_2gm_bQ5_qJwnUE14bvB-YHHOYY4o8rj9FB7A_45Ya5BCccoY_Sj9L9ASUda35KhCGam7sPsOwzsR-RN3sTha52xfoa4cXXCKvTV64p9nwYgMpgYYueobrM7r8XwNPTOshJDtUIELx2vakCfXRYwPjWScuF2-JOoUUyHQJMLNlVZ2JF7qoHeGLHrDkvesNxZREA_FH8-4lOu4Uvg6dS0S1FB1RF8o8U_xVjUHc4A2-khwGGef7BVzhte3frNTbOIG3SvXKEzDo1LMljnoQZLW6vKN6eF7ICc
Response	Successfully logged out!

Nota: Si el usuario no está autenticado y quiere acceder a alguna ruta que necesita autenticación devolverá el response: Unauthorized.

Para la creación de los roles, se hace de forma manual para el primer profesor, se adjunta los Inserts.

```
1 •   SELECT * FROM `api-dacodes`.roles;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	slug	description	created_at	updated_at	special
1	Profesor	teacher	Rol para tener privilegios de profesor	2020-08-31 13:03:50	2020-08-31 13:03:50	NULL
2	Alumno	student	Rol para tener privilegios de alumno	2020-08-31 13:03:50	2020-08-31 13:03:50	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM `api-dacodes`.role_user;
```

Result Grid | Filter Rows: | Edit:

id	role_id	user_id	created_at	updated_at
1	1	1	2020-08-31 13:03:50	2020-08-31 13:03:50
NULL	NULL	NULL	NULL	NULL

Para acceder a las rutas de Profesor el usuario tiene que estar autenticado y tener el rol de profesor.

```
//************************** TEACHER ROUTES ******************************/

//Rutas de cursos
Route::get('/admin/course', 'Admin\CoursesAdminController@index')->middleware('auth:api', 'has.role:teacher')->name('course');
Route::post('/admin/course/create', 'Admin\CoursesAdminController@createCourse')->middleware('auth:api', 'has.role:teacher')->name('course.cre
Route::post('/admin/course/update', 'Admin\CoursesAdminController@updateCourse')->middleware('auth:api', 'has.role:teacher')->name('course.upd
Route::delete('/admin/course/delete', 'Admin\CoursesAdminController@deleteCourse')->middleware('auth:api', 'has.role:teacher')->name('course.d
Route::put('/admin/course/active', 'Admin\CoursesAdminController@updateCourseStatusByColum')->middleware('auth:api', 'has.role:teacher')->name
Route::post('/admin/course/approve', 'Admin\CoursesAdminController@ApproveUserCourse')->middleware('auth:api', 'has.role:teacher')->name('cour
```

Para acceder a las rutas de Alumno el usuario tiene que estar autenticado.

```
//************************** STUDENT ROUTES ******************************/

Route::get('/user/course', 'User\CoursesUserController@index')->middleware('auth:api')->name('courseSt');
Route::get('/user/lesson', 'User\LessonUserController@index')->middleware('auth:api')->name('lessonSt');
Route::get('/user/question', 'User\QuestionUserController@index')->middleware('auth:api')->name('questionSt');

Route::post('/user/typequestion/answer', 'User\QuestionUserController@createAnswer')->middleware('auth:api')->name('typequestion.answer');
```

5.- Rutas de Profesor

Se listan algunas rutas de profesor

Ver cursos	
http://localhost:8000/api/admin/course	GET
Request	{ }

Response	[
	{
	"id": 1,
	"name": "Curso 1",
	"description": "prueba de curso 1",
	"index": 1,
	"active": true,
	"created_at": "2020-09-01T00:08:53.000000Z",
	"updated_at": "2020-09-01T00:08:53.000000Z"
	},
	{
	"id": 2,
	"name": "Curso 2",
	"description": "prueba de curso 2",
	"index": 2,
	"active": true,
	"created_at": "2020-09-18T01:10:28.000000Z",
	"updated_at": "2020-09-29T02:59:22.000000Z"
	}
	]

DaCodes.	**Desafío para puesto de desarrollador BackEnd**	

Crear curso	
http://localhost:8000/api/admin/course/create	POST

Request	{
	"name": "Curso 4",
	"description": "prueba de curso 4 modificado",
	"index": 3,
	"active": true
	}

Response	{
	"success": true,
	"error": null
	}

Actualizar curso	
http://localhost:8000/api/admin/course/update	POST

Request	{
	"id": 4,
	"name": "Curso 4",
	"description": "prueba de curso 4 modificado",
	"index": 3,
	"active": true
	}

Response	{
	"success": true,
	"error": null
	}

Borrar curso	
http://localhost:8000/api/admin/course/delete	DELETE

Request	{
	"id": 4
	}

Response	{
	"success": true,
	"error": null
	}

Nota: todos los CRUD de lecciones, preguntas y tipos de preguntas (respuestas) son similares y solo las puede crear el profesor.

Las rutas donde debe acceder el alumno y el estudiante están en el archivo api.php

6.- Asignar un curso a usuario.

Solo el profesor puede asignar y desasignar un curso a un usuario. Así como de activar o desactivar un curso.

Asignar curso	
http://localhost:8000/api/admin/user/assignUserToCourse	POST

Request	{
	"idUser":2,
	"idCourse": 2
	}

Response	{
	"success": true,
	"user": {
	"id": 2,
	"name": "Roger J",
	"email": "obiwk@hotmail.com",
	"email_verified_at": null,
	"created_at": "2020-09-01T00:09:18.000000Z",
	"updated_at": "2020-09-01T00:09:18.000000Z"
	},
	"course": {
	"id": 2,
	"name": "Curso 2",

	"description": "prueba de curso 2", "index": 2, "active": true, "created_at": "2020-09-18T01:10:28.000000Z", "updated_at": "2020-09-29T02:59:22.000000Z" } }

Desasignar curso	
http://localhost:8000/api/admin/user/unassignUserToCourse	POST

Request	{
	"idUser":2,
	"idCourse": 2
	}

Response	{
	"success": true,
	"user": {
	"id": 2,
	"name": "Roger J",
	"email": "obiwk@hotmail.com",
	"email_verified_at": null,
	"created_at": "2020-09-01T00:09:18.000000Z",
	"updated_at": "2020-09-01T00:09:18.000000Z"
	},
	"course": {
	"id": 2,
	"name": "Curso 2",
	"description": "prueba de curso 2",
	"index": 2,
	"active": true,
	"created_at": "2020-09-18T01:10:28.000000Z",
	"updated_at": "2020-09-29T02:59:22.000000Z"
	}

	}

7.- Lógica sobre de responder las preguntas de las lecciones, de los cursos y aprobar.

Como lo especifica el modelo de base de datos y las migraciones, una o varias lecciones están ligadas a un curso. Una o varias preguntas están ligadas a las lecciones y las respuestas por usuario están ligadas a una pregunta.

Lógica de aprobación:

1.- Los catálogos de cursos y de lecciones tienen un campo index, donde se verifica el consecutivo de los mismos. No se pueden contestar preguntas de una lección o curso, si su anterior no está aprobada.

2.- El catálogo de preguntas tiene un tipo, aunque no hay un catálogo para el tipo se entiende que:

- Tipo 1: Booleano
- Tipo 2: Opción múltiple donde solo una respuesta es correcta
- Tipo 3: Opción múltiple donde más de una respuesta es correcta
- Tipo4: Opción múltiple donde más de una respuesta es correcta y todas deben responderse correctamente

3.- El catálogo de tipo preguntas contiene las respuestas, tiene un campo de aprobación, para que el profesor pueda aprobar la respuesta.

4.- Hay dos tipos de aprobación en las repuestas, el automático donde el api compara por el tipo de pregunta las repuestas y si son iguales a las correctas se aprueban de manera automática, el otro donde el profesor aprueba de manera manual cada respuesta.

5.- Los cursos por usuario solo puede aprobarlos el profesor de forma manual.

8.- Aprobar preguntas y cursos.

Solo los profesores pueden aprobar preguntas y cursos.

Aprobación automático	
http://localhost:8000/api/admin/typequestion/approve	POST

Request	{
	"id":0,
	"user_id":2
	}

Response	{
	"success": true,
	"error": null
	}

Aprobación manual	
http://localhost:8000/api/admin/typequestion/approve	POST

Request	{
	"id":1,
	"user_id":2
	}

Response	{
	"success": true,
	"error": null
	}

Aprobación curso	
http://localhost:8000/api/admin/course/approve	POST

Request	{
	"user_id":2,
	"course_id":1,
	"approve": 1
	}

Response	{
	"success": true,
	"error": null
	}