<u>Tareas que quedaran por finalizar.</u>
Grupo Celulosa / Técnica: Células de aprendizaje.

- **Creación de usuarios**
    - La plataforma debería ser capaz de crear y administrar los usuarios.

- **Clasificador de usuario**
    - El modulo debe clasificar a los usuarios según correspondan los privilegios otorgados, se dividen en:
        - Administrador (Control total de la publicación)
        - Sub-Administrador (Funciones de edición)
        - Estudiante Regular (Solo puede ver la bitácora)

- **Clasificación de materias por usuario**
    - Cada usuario debe tener una clasificación recomendada para ingresar a las bitácoras existentes según el interés y/o materia relacionada con la carrera que este cursando.

    - Se busca clasificar material de interés al usuario atreves de los comentarios y las puntuaciones que designa el usuario.

- **Conexión parcial API**
    - Debido a que la API demoro en ingresar información a la base de datos, implementamos colecciones anexas dentro de la misma BD pero con el mismo sentido que la API señalada al backend.

- **Implementación de opción a calificar un post incompleta debido a:**
    - El modulo necesita verificar los usuarios para limitar que se pueda calificar más de una vez por bitácora.

<u>Recomendaciones</u>

- La implementación de una base de datos no relacional retraso considerablemente el desarrollo del proyecto debido a que dificulta la escala de información base un mismo objeto.

- Crear la guía de API y los datos de manera más expedita podría haber resuelto algunos aspectos de tiempos en el desarrollo del proyecto.

- **Diferencias entre API N°1 v/s N°2**
  - **Versión 1**
  - **Conexión BD (Directa)**
    Conexión directa con base de datos mongo db
    Agregar a database.php

```
'default' => env('DB_CONNECTION', 'mongodb'),
```

    - **Agregar servicio que maneja el controlador de mongodb**

```
     * Package Service Providers...
     */
    Jenssegers\Mongodb\MongodbServiceProvider::class,
```

  - Ajustes de conexión en archivo .env

```
DB_CONNECTION=mongodb
DB_HOST=noestudiosolo.inf.uct.cl
DB_PORT=27017
DB_DATABASE=noestudiosolodb_front
DB_USERNAME=noestudiosolo_front
DB_PASSWORD=noestudiosoloforfrontend
```

Obtención de colecciones

Obtenemos todos los documentos de la colección bitácora de manera directa y los enviamos a la vista.

Utilizamos funciones “all” y “find” para buscar y obtener los datos de los documentos Bitácora.

```
class BitacoraController extends Controller
{
    /**
     * Muestra todas las Bitacoras
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $Bitacora = Bitacora::all();
        return view('Bitacora.index',compact('Bitacora'));
    }
```

```php
public function show(Bitacora $bitacora,$titulo)
{

    $bitacora = Bitacora::find($titulo);
    return view('Bitacora.show',compact('bitacora','titulo'));
}
```

- **Versión 2**

Conexión DB (Utilizando Guzzle)

Se debe llamar un cliente que se conecta la db.

```php
        /**
         * Conexion a la API, base uri es la ruta base
         *  y verify false es para evitar problemas con el certificado.
         * @retorna el objeto cliente con la conexion configurada.
         */
        $this->app->singleton('GuzzleHttp\Client', function(){
            return new Client([
                'base_uri' => 'https://noestudiosolo.inf.uct.cl/',
                'verify' => false
            ]);
        });
```

Obtención de colecciones

A diferencia de la versión 1, esta api necesita que retorne la información al cliente conectado

```php
<?php

namespace App\Repositories;
use GuzzleHttp\Client;

class GuzzleHttpRequest{

    /*
    *   Construccion del objeto cliente que trae la libreria.
    */
    protected $client;

    public function __construct(Client $client){
        $this->client = $client;
    }

    /*
    *   Funcion que recibe como parametro la terminacion de la ruta a la api y hace el get.
    *   @retorna la informacion enviada por el API como un array.
    */
```

También es necesario crear una función “find “ y “all” para obtener los datos de una colección específica y todas las colecciones.

```php
<?php

namespace App\Repositories;

class Tecnicas extends GuzzleHttpRequest{

    /*
    *   Funcion que devuelve un array con todos los documentos de la coleccion Tecnica.
    */

    public function all(){
        return $this->get('tecnica');
    }

    /*
    *   Funcion que devuelve la Tecnica de la id especificada.
    */
    public function find($id){
        return $this->get("tecnica/{$id}");
    }
}

?>
```

Teniendo el cliente y las funciones creadas recién se pueden obtener los datos de técnicas.

```php
    public function __construct(Tecnicas $tecnicas){
        $this->tecnicas = $tecnicas;
    }

    /*
    *   Funcion que maneja la vista de la pagina /tecnicas
    */
    public function index(){

        $tecnicas = $this->tecnicas->all();

        return view('tecnicas.index', compact('tecnicas'));
    }

    /*
    *   Funcion que maneja la vista de la pagina /tecnicas/{id}
    */
    public function show($id){

        $tecnica = $this->tecnicas->find($id);

        return view('tecnicas.show', compact('tecnica'));
    }
}
```

Cabe destacar que la API N°2 **no implemento** las funciones:

Create, Update y Delete del CRUD, entregando solo de esta manera el READ.

A la entrega de la información de la API N°2 el equipo ya tenía el CRUD realizado con la API N°1