

Proyecto-Sistema Penitenciario

Manuel Auqui, Leoni Guambo, Mayerli Méndez, Jorge Ortiz, Luis Valencia, Escuela Politécnica Nacional (EPN), Quito - Ecuador

C. Vite

Resumen – En este documento se muestra la resolución del proyecto final el cual consiste en un sistema penitenciario, en la cual se utilizó diferentes herramientas React, Heroku, Alwaysdata y Swagger. Donde estos componentes ayudaran en la creación de una aplicación sólida que estará estructurada con el uso del BackEnd y el FrontEnd. Los componentes que se usaran de preferencia será Laravel y React por ser una librería y framework que nos ayudara con la formulación de carpetas y asignaciones de procesos de una manera más fácil, intuitiva y dinámica en la formulación de nuestro código, con el objetivo de cumplir la funcionalidad y la lógica del programa. Swagger será utilizado con el propósito de llevar una documentación precisa para el usuario donde tendrá conocimientos claros en el uso de la funcionalidad y los parámetros que se usará como ejemplo en el desarrollo de la API.

I. INTRODUCCIÓN

A continuación, se realiza la parte teórica de como fue el proceso práctico en el desarrollo del sistema penitenciario, basándose tanto en el laboratorio del desarrollo de los sprints del backend y frontend , usando nuestras bases de datos always data como almacenamiento de los datos . Después de ello usaremos una base de datos que alojará nuestros datos de forma online que siempre estará activa en las funciones que realicemos en la página web del sistema penitenciario. Se usó react para trabajar con el frontend y dar las funcionalidades correspondientes a cada CRUD . En heroku haremos el despliegue de la página, pero siempre con las configuraciones de claves que contiene la aplicación que creemos en ese sistema. Desplegando así la API del backend y frontend.

II. CONOCIMIENTO TEÓRICO

A. React

React es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. En el proyecto hay más de mil desarrolladores libres.

B. Jsx

Es una extensión de React para la sintaxis del lenguaje JavaScript que proporciona una forma de estructurar la representación de componentes utilizando una sintaxis familiar para muchos desarrolladores. Es similar en apariencia a HTM.

D. Always data

La API de alwaysdata.net permite que aplicaciones externas consuman servicios automáticos de resumen, almacenamiento y análisis cuantitativo y cualitativo, las aplicaciones externas realizan llamadas de tipo GET y POST, con las cuales obtienen y guardan datos en las bases de datos.

E. Swagger

Es una documentación que muestra una documentación de una API para que sea entendible para el usuario donde se crearan métodos como un ejemplo de compilación y formulación de la lógica del Bac-End- Utiliza formato JSON o YAML para que el uso de su interfaz sea interactivo, donde esta aplicación utilizará los métodos HTTP como GET, POST, PUT o DELETE para crear ejemplos de los parámetros que se requiere para el funcionamiento de la API y el usuario sabrá el sentido de la arquitectura del algoritmo.

Los métodos que se utilizarán para la documentación de la API son:

- @OA: Anotación abierta de la API.
- @OA|Post: Anotación de la API con su método HTTP.
- Path: dirección URL
- Tags: Conjunto que se almacenara los métodos
- @OA Request: Anotaciones de campo donde asignara parámetros dentro de un JSON Content.
- @OA Response: Posibles respuesta de ejecución con su código HTTP.

Swagger se utiliza dentro de controladores de una aplicación Backend de Laravel donde un comentario o documentación se enlazará con una función para su definición y validación de la instrucción.

III. DESARROLLO

A. Organización y trabajo en equipo.

Product owner: Ing Byron Loarte

Scrum Máster: Jorge Ortiz

Dev Team: Auqui- Guambo- Méndez-Valencia

B. Sprint de backend.

1. Documentación de la API con Swagger.

En la Fig.1 vamos a comenzar con la instalación de los paquetes de Swagger-php y Swagger UI con el siguiente comando de instalación.

```
Regenerating docs default
PS C:\Users\diana\OneDrive\Desktop\CuartoSemestreESFOT\PracticasPreprofesionales\proyectobackend>> 1
>> composer require "darkaonline/l5-swagger"
```

Fig.1. Instalación de paquetes.

En la Fig.2 se instalará los paquetes de L5 de Swagger para su configuración de Token, OAuth y Sactum para la documentación de nuestra API.

```
php artisan vendor:publish --provider "L5Swagger\L5SwaggerServiceProvider"
```

Fig.2. Instalacion de paquetes de Swagger UI y Sactum.

Comenzaremos con la configuración de L5Swagger para admitir las autorizaciones del token de tipo Bearer, esto nos ayudara para implantar nuestro token al momento de utilizar métodos que necesiten una autenticación de nuestros perfiles y además se comentara las propiedades de Sactum para no tener conflictos con la asignación de los tokens como se visualiza en la Fig.3.

```
// Open API 3.0 support
'passport' => [
    'type' => 'apiKey', // The type of the security scheme. Valid values are "basic", "httpBasic", "apiToken", "laravel passport oauth2 security",
    'in' => 'header',
    'name' => 'Authorization',
    'scheme' => 'https',
    'flows' => [
        'password' => [
            'authorizationUrl' => config('app.url') . '/oauth/authorize',
            'tokenUrl' => config('app.url') . '/oauth/token',
            'refreshUrl' => config('app.url') . '/token/refresh',
            'scopes' => []
        ],
        ...
    ],
    ...
],
// 'sanctum' => [ // Unique name of security
    'type' => 'bearer', // Valid values are "Basic", "apiKey" or "oauth2",
    'description' => 'Enter token in format (Bearer <token>)',
    'name' => 'Authorization', // The name of the header or query parameter to be used
    'in' => 'header', // The location of the API key. Valid values are "query" or "header"
    ...
],
'security' => [
    ...
]
```

Fig.3. Configuración de L5-Swagger de permisos OAuth

Podremos visualizar Swagger por medio del inicio de php artisan serve y poner en el local host <http://127.0.0.1:8000/api/documentation> esto nos abrirá una interfaz gráfica de Swagger donde podremos ver un botón de authorize como se visualiza en la Fig.4.

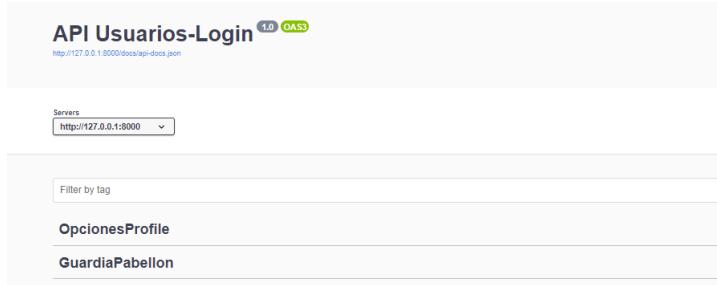


Fig.4. Interfaz Principal de Swagger

Para poder realizar una documentación debemos recordar que por cada documentación debe enlazarse a una función de los controladores para que puedan ser válidos.

Podremos visualizar en la Fig.5 y Fig.6 la primera documentación del AuthController donde se agregará un método POST y la asignación de parámetros que podremos insertar datos que puedan validar su funcionamiento donde se visualizará el primer comentario de la función login.

```
/*
 * @OA\Server(url="http://127.0.0.1:8000")
 */
class AuthController extends Controller
{
    /**
     * @OA\Post(
     *     path="/api/v1/login",
     *     operationId="authLogin",
     *     tags={"Login"},
     *     summary="User Login",
     *     description="Login User Here",
     *     @OA\RequestBody(
     *         @OA\JsonContent(),
     *         @OA\MediaType(
     *             mediaType="multipart/form-data",
     *             @OA\Schema(
     *                 type="object",
     *                 required={"email", "password"},
     *                 @OA\Property(property="email", type="email"),
     *                 @OA\Property(property="password", type="password")
     *             ),
     *         ),
     *         @OA\Response(
     *             response=201,
     *             description="Login Successfully",
     *             @OA\JsonContent()
     *         ),
     *         @OA\Response(
     *             response=200,
     *             description="Login Successfully",
     *         )
     *     )
     */
    public function login(Request $request)
    {
        ...
    }
}
```

Fig.5. Método POST de ingreso de elementos login.

```
        ...
        description="Login Successfully",
        @OA\JsonContent()
    ),
    @OA\Response(
        response=200,
        description="Login Successfully",
        @OA\JsonContent()
    ),
    @OA\Response(
        response=422,
        description="Unprocessable Entity",
        @OA\JsonContent()
    ),
    @OA\Response(response=400, description="Bad request"),
    @OA\Response(response=404, description="Resource Not Found"),
    ...
)
// Creación de un array con los roles que se pueden descartar
private $discarded_role_names = ['prisoner'];

// Función para el manejo del inicio de sesión
public function login(Request $request)
{
    ...
    // Validación de los datos de entrada
    $request->validate([
        'email' => ['required', 'string', 'email'],
        'password' => ['required', 'string'],
    ]);

    ...
    // Obtener un usuario
    $user = User::where('email', $request['email'])->first();
    ...
}
```

Fig.6. Función login para el comentario Swagger.

En la Fig.7 podremos visualizar una interfaz donde se da la accesibilidad de autorización de token Bearer que será indispensable para los métodos que lo requieran y esto es obtenido cuando obtenemos ese token de login.

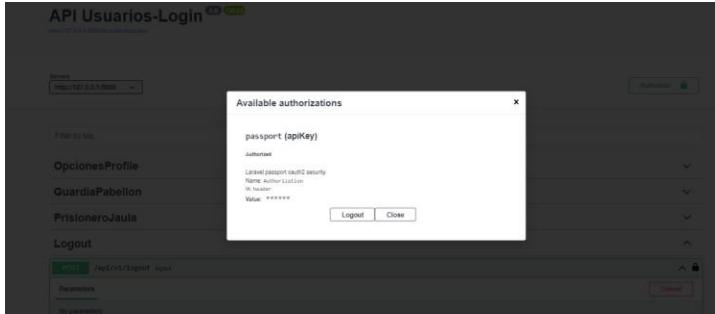


Fig.7. Autorización del código Bearer OAuth.

Ahora en la Fig.8 dispondremos de igual forma en el comentario de instrucciones para la función logout con una característica especial en donde se utilizará el valor del token cuando se asignó en el inicio de sesión de login.

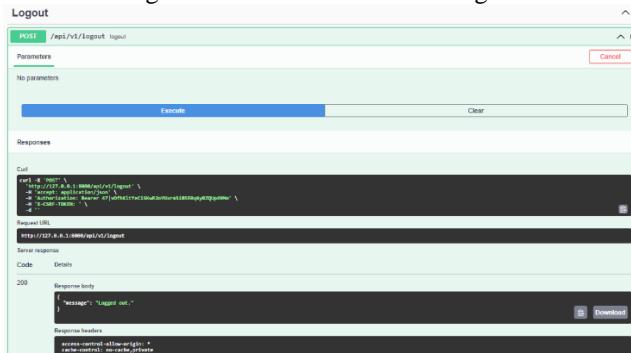


Fig.8. Funcionamiento de login.

Ahora procederemos con el reseteo de nuestra contraseña como un trámite de recuperación de contraseña donde nos iremos al controlador PasswordController y empezaremos a crear el siguiente comentario para la función de resendlink como se visualiza en la Fig.9 y Fig.10.

```

AuthController.php M PasswordController.php X ProfileController.php I, M
app > Http > Controllers > Auth > PasswordController.php > PasswordController > update
  9
10  use App\Http\Controllers\Controller;
11  use Illuminate\Http\Request;
12
13 class PasswordController extends Controller
14 {
15
16     /**
17      * @OA\Post(
18      * path="/api/v1/forgot-password/",
19      * tags={"Login"},
20      * summary="Login",
21      * operationId="forgot-password",
22      *
23      * @OA\Parameter(
24      * name="email",
25      * in="query",
26      * required=true,
27      * @OA\Schema(
28      * type="string"
29      * )
30      *
31      * @OA\Response(
32      * response=200,
33      * description="Success",
34      * @OA\MediaType(
35      * mediaType="application/json",
36      * )
37      * ),
38      * @OA\Response(
39      * response=401,
40      * description="Unauthenticated"
41      * )
42      * ),
43      * @OA\Response(
44      * response=401,
45      * description="Bad Request"
46      * ),
47      * @OA\Response(
48      * response=404,
49      * description="not found"
50      * ),
51      * @OA\Response(
52      * response=403,
53      * description="Forbidden"
54      * )
55      */
56     // Función para el manejo del reseteo de contraseña
57     public function resendLink(Request $request)
58     {
59         // Validación de los datos de entrada
60         $request->validate([
61             'email' => ['required', 'email'],
62         ]);
63
64         // enviar el link de restablecimiento de contraseña al mail
65         // https://laravel.com/docs/9.x/passwords#requesting-the-password-reset-link
66         $status = Password::sendResetLink(
67             $request->only('email')
68         );
69
70         if ($status) {
71             return response()->json(['Message' => 'Logout out.']);
72         }
73     }
74 }

```

```

36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

```

Fig.10. Función resendLink que validara el comentario Swagger

Ahora procederemos con la creación de una nueva contraseña después de enviar el mensaje de reseteo de contraseña donde este comentario se ligará a la función restore como se visualiza en la Fig.11, Fig.12 y Fig.13.

```

    /**
     * @OA\Post(
     * path="/api/v1/reset-password/",
     * tags={"Login"},
     * summary="Login",
     * operationId="reset-password",
     *
     * @OA\Parameter(
     * name="token",
     * in="query",
     * required=true,
     * @OA\Schema(
     * type="string"
     * )
     * ),
     * @OA\Parameter(
     * name="email",
     * in="query",
     * required=true,
     * @OA\Schema(
     * type="string"
     * )
     * ),
     * @OA\Parameter(
     * name="password",
     * in="query",
     * required=true,
     * @OA\Schema(
     * type="string"
     * )
     * ),
     * @OA\Parameter(
     * name="password_confirmation"
     * )
     */

```

Fig.11. Creación de una nueva contraseña con método POST.

Fig.9. Método POST con sus parámetros de resetear contraseña.

```

112     *      name="password_confirmation",
113     *      in="query",
114     *      required=true,
115     *      @OA\Schema(
116     *          type="string"
117     *      ),
118     *  ),
119     *  @OA\Response(
120     *      response=200,
121     *      description="Success",
122     *      @OA\MediaType(
123     *          mediaType="application/json",
124     *      )
125   ),
126   @OA\Response(
127     *      response=401,
128     *      description="Unauthenticated"
129   ),
130   @OA\Response(
131     *      response=400,
132     *      description="Bad Request"
133   ),
134   @OA\Response(
135     *      response=404,
136     *      description="not found"
137   ),
138   @OA\Response(
139     *      response=403,
140     *      description="Forbidden"
141   ),
142   )
143 )

```

Fig.12. Parámetros necesarios de crear contraseña.

```

140     *      description="Forbidden"
141   },
142   */
143 /**
144 // Función para enviar el redirect del formulario para restablecer la contraseña
145 public function redirectReset(Request $request)
146 {
147     $frontend_url = env('APP_FRONTEND_URL');
148     $token = $request->route('token');
149     $email = $request->email;
150     $url = $frontend_url.'/'.$token.$email;
151     return $this->sendResponse(message: 'Successful redirection', result: ['url' => $url]);
152 }
153
154 // Función para la actualización del password
155 public function restore(Request $request)
156 {
157     // Validación de los datos de entrada
158     $validated = $request -> validate([
159         'token' => ['required', 'string'],
160         'email' => ['required', 'string', 'email'],
161         // https://laravel.com/docs/9.x/validation#rule-confirmed
162         'password' => [
163             'required',
164             'string',
165             'confirmed',
166             PasswordValidator::defaults()->mixedCase()->numbers()->symbols(),
167         ],
168     ]);
169
170     // Función para cambiar el password
171     $status = Password::reset($validated, function ($user, $password)
172     {

```

Fig.13. Función restore para la creación de nueva contraseña Swagger

Al momento de adquirir nuestro token y de completar los parámetros para el reseteo podremos visualizar en la Fig.14 y Fig.15 como nos devuelve una confirmación de crear una nueva contraseña por medio del reseteo.

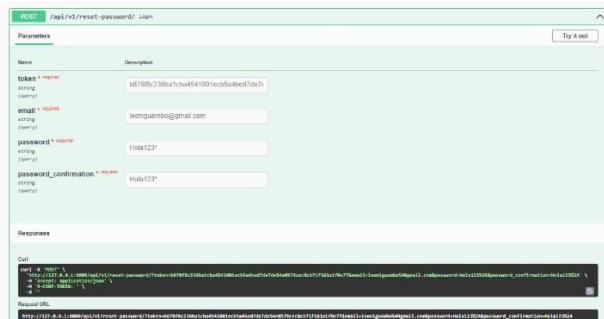


Fig.14. Interfaz de autenticacion de Swagger.

Fig.15. Comprobación de reseteo de contraseña

Podemos actualizar nuestra contraseña por medio de la función update que se enlazara con nuestro comentario de Swagger para obtener parámetros de cambiar nuestro password como se visualiza en la Fig.16.

```

188     }
189
190 /**
191  *  * @OA\Post(
192  *  * path="/api/v1/update-password",
193  *  * tags={"Logout"},
194  *  * summary="update password",
195  *  * operationId="actualizarkey",
196  *  * security:[
197  *      {"passport": {}},
198  *  ],
199  *  * @OA\Parameter(
200  *      name="password",
201  *      in="query",
202  *      required=true,
203  *      @OA\Schema(
204  *          type="string"
205  *      )
206  *  ),
207  *  * @OA\Parameter(
208  *      name="password_confirmation",
209  *      in="query",
210  *      required=true,
211  *      @OA\Schema(
212  *          type="string"
213  *      )
214  *  ),
215  *  * @OA\Response(
216  *      response=200,
217  *      description="Success",
218  *      @OA\MediaType(
219  *          mediaType="application/json",
220  *      )
221  *  ),
222  *  * @OA\Response(
223  *      response=401,
224  *      description="Unauthenticated"
225  *  ),
226  *  * @OA\Response(
227  *      response=400,
228  *      description="Bad Request"
229  *  ),
230  *  * @OA\Response(
231  *      response=404,
232  *      description="not found"
233  *  ),
234  *  * @OA\Response(
235  *      response=403,
236  *      description="Forbidden"
237  *  )
238  *  */
239
// Función para actualizar el password del usuario
public function update(Request $request)
{
    // Validación de los datos de entrada
    $validated = $request -> validate([
        'password' => ['required', 'string', 'confirmed'],
        PasswordValidator::defaults()->mixedCase()->numbers()->symbols()
    ]);
    $user = $request->user();
}

```

Fig.16. Método POST para la actualización de contraseña.

```

217     *      response=200,
218     *      description="Success",
219     *      @OA\MediaType(
220     *          mediaType="application/json",
221     *      )
222   ),
223   @OA\Response(
224     *      response=401,
225     *      description="Unauthenticated"
226   ),
227   @OA\Response(
228     *      response=400,
229     *      description="Bad Request"
230   ),
231   @OA\Response(
232     *      response=404,
233     *      description="not found"
234   ),
235   @OA\Response(
236     *      response=403,
237     *      description="Forbidden"
238   )
239  */
240
// Función para actualizar el password del usuario
public function update(Request $request)
{
    // Validación de los datos de entrada
    $validated = $request -> validate([
        'password' => ['required', 'string', 'confirmed'],
        PasswordValidator::defaults()->mixedCase()->numbers()->symbols()
    ]);
    $user = $request->user();
}

```

Fig.16. Parámetros necesarios y función update.

Podremos verificarlo en Swagger su actualización de password como se visualiza en la Fig.17.

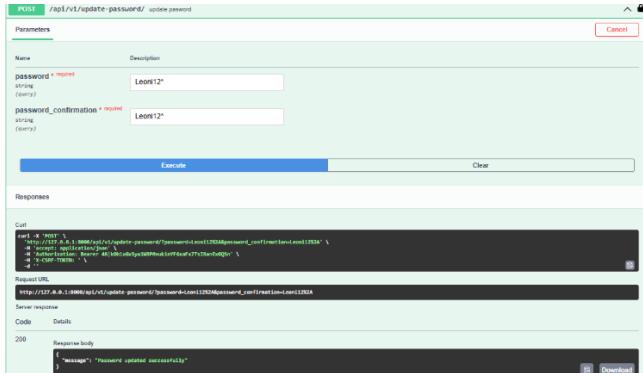


Fig.17. Actualización de contraseña.

Ahora comenzaremos con la visualización de los datos del perfil del usuario por medio del método GET que se usara en el controlador ProfileController como se visualiza en la Fig.18 .

```
app > Http > Controllers > Account > ProfileController.php > ProfileController
14 /**
15  * @OA\Get(
16  *     path="/api/v1/profile/",
17  *     tags={"OpcionesProfile"},
18  *     summary="Ver perfil",
19  *     operationId="profile-show",
20  *     security={
21  *         {"passport": {}},
22  *     },
23  *
24  *     @OA\Response(
25  *         response=200,
26  *         description="Success",
27  *         @OA\MediaType(
28  *             mediaType="application/json",
29  *         )
30  *     ),
31  *     @OA\Response(
32  *         response=401,
33  *         description="Unauthenticated"
34  *     ),
35  *     @OA\Response(
36  *         response=400,
37  *         description="Bad Request"
38  *     ),
39  *     @OA\Response(
40  *         response=404,
41  *         description="not found"
42  *     )
43  */
44 /**
45  * @OA\Parameter(
46  *     name="last_name",
47  *     in="query",
48  *     required=true,
49  *     @OA\Schema(
50  *         type="string"
51  *     )
52  */
53 // función para mostrar los datos de perfil del usuario
54 public function show()
55 {
56     // Se obtiene el usuario autenticado
57     // https://laravel.com/docs/9.x/authentication#retrieving-the-authenticated-user
58     $user = Auth::user();
59     // Se invoca a la función padre
60     return $this->sendResponse(message: "User's profile returned successfully", result: [
61         'user' => new ProfileResource($user),
62         'avatar' => $user->getAvatarPath()
63     ]);
64 }
```

Fig.18. Método GET para la visualización de Perfil.

```
35 /**
36  * @OA\Response(
37  *     response=401,
38  *     description="Unauthenticated"
39  * ),
40  * @OA\Response(
41  *     response=400,
42  *     description="Bad Request"
43  * ),
44  * @OA\Response(
45  *     response=404,
46  *     description="not found"
47  * ),
48  * @OA\Response(
49  *     response=403,
50  *     description="Forbidden"
51  * )
52 */
53 // función para mostrar los datos de perfil del usuario
54 public function show()
55 {
56     // Se obtiene el usuario autenticado
57     // https://laravel.com/docs/9.x/authentication#retrieving-the-authenticated-user
58     $user = Auth::user();
59     // Se invoca a la función padre
60     return $this->sendResponse(message: "User's profile returned successfully", result: [
61         'user' => new ProfileResource($user),
62         'avatar' => $user->getAvatarPath()
63     ]);
64 }
```

Fig.18. Función show para ver su perfil

Con ello podremos visualizar en Swagger como podremos nosotros introducir nuestros datos y ver el resultado del perfil de usuario como se visualiza en la Fig.19 de la función show.

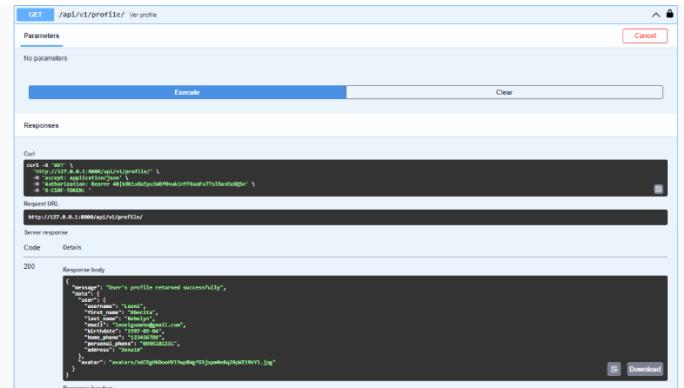


Fig.19. Perfil en Swagger

Como siguiente paso del ProfileController vamos a actualizar los datos por medio de POST en el comentario que se pondrá para que Swagger sepa que elementos se va a introducir como se visualiza en la Fig.20, Fig.21de la función store

```
91 /**
92  * @OA\Parameter(
93  *     name="last_name",
94  *     in="query",
95  *     required=true,
96  *     @OA\Schema(
97  *         type="string"
98  *     )
99  * ),
100 * @OA\Parameter(
101 *     name="email",
102 *     in="query",
103 *     required=true,
104 *     @OA\Schema(
105 *         type="string"
106 *     )
107 * ),
108 * @OA\Parameter(
109 *     name="birthdate",
110 *     in="query",
111 *     required=true,
112 *     @OA\Schema(
113 *         type="string"
114 *     )
115 * ),
116 * @OA\Parameter(
117 *     name="home_phone",
118 *     in="query",
119 *     required=true,
120 *     @OA\Schema(
121 *         type="string"
122 *     )
123 * ),
124 */
125 /**
126 * @OA\Post(
127 *     path="/api/v1/profile/",
128 *     tags={"OpcionesProfile"},
129 *     summary="Actualiza los datos del perfil",
130 *     operationId="profile-store",
131 *     security={
132  *         {"passport": {}},
133  *     },
134  *     @OA\Parameter(
135  *         name="username",
136  *         in="query",
137  *         required=true,
138  *         @OA\Schema(
139  *             type="string"
140  *         )
141  *     ),
142  *     @OA\Parameter(
143  *         name="first_name",
144  *         in="query",
145  *         required=true,
146  *         @OA\Schema(
147  *             type="string"
148  *         )
149  *     ),
150  *     @OA\Parameter(
151  *         name="last_name",
152  *         in="query",
153  *         required=true,
154  *         @OA\Schema(
155  *             type="string"
156  *         )
157  *     )
158  */
159 
```

Fig.20. Método POST y actualización de perfil.

```

127
128    * @OA\Parameter(
129        * name="personal_phone",
130        * in="query",
131        * required=true,
132        * @OA\Schema(
133            * type="string"
134        )
135    );
136
137    * @OA\Parameter(
138        * name="address",
139        * in="query",
140        * required=true,
141        * @OA\Schema(
142            * type="string"
143        )
144    );
145
146    * @OA\Response(
147        * response=200,
148        * description="Success",
149        * @OA\MediaType(
150            * mediaType="application/json",
151        )
152    );
153    * @OA\Response(
154        * response=401,
155        * description="Unauthenticated"
156    );
157    * @OA\Response(
158        * response=400,
159        * description="Bad Request"
160    );
161
162    * @OA\Response(
163        * response=400,
164        * description="Bad Request"
165    );
166
167    * @OA\Response(
168        * response=404,
169        * description="not found"
170    );
171
172    * @OA\Response(
173        * response=403,
174        * description="Forbidden"
175    );
176
177    * }
178
179
180    // función para actualizar los datos del usuario
181    public function store(Request $request)
182    {
183        // Validar que el usuario sea mayor de edad
184        $allowed_date_range = [
185            'max' => date('Y-m-d', strtotime('-70 years')),
186            'min' => date('Y-m-d', strtotime('-18 years')),
187        ];
188
189        // Validación de los datos de entrada
190        $request->validate([
191            'first_name' => ['required', 'string', 'min:3', 'max:35'],
192            'last_name' => ['required', 'string', 'min:3', 'max:35'],
193            'email' => ['required', 'string', 'email'],
194            'birthdate' => ['nullable', 'string', 'date_format:Y-m-d', 'after_or_equal:'.$allowed_date_range['max']],
195            'birthday' => ['nullable', 'string', 'date_format:Y-m-d', 'before_or_equal:'.$allowed_date_range['min']],
196            'password' => ['nullable', 'string', 'min:6', 'max:20', 'rule:unique[users]>ignore[$request->user()->id]'],
197        ]);
198
199        // https://laravel.com/docs/9.x/validation#rule-alternative
200
201        $user = User::find($request->user());
202        $user->fill($request->only(['first_name', 'last_name', 'email', 'password']));
203        $user->save();
204    }

```

Fig.21. Parámetros de actualización y función store.

Y se puede visualizar la diferencia entre la cuenta antigua y la actual como se visualiza en la Fig.22, Fig.23, Fig.24 y Fig.25.

Fig.22. Perfil anterior

de_id	first_name	last_name	personal_phone	address	password	state	email
4	Wyatt	Dubugue	0937529036	772 Leonora Motorway Apt. 270	\$2y\$10\$mfoc8VTY1tbP3420QRFXwO.FOHmMZl93VmAjyGWRwK...	1	woodrow0@example.com
2	maria	otero	0980955854	Distrito metropolitano	\$2y\$10\$J3ejBoedh7we3Z1H3L.er22QQ3W198N0.UWwTPQ...	0	maria@mail.com
2	Juan	Yorobush	0995102822	Tumbaco	\$2y\$10\$uCFilgyPrin2Ulc26X6eTsc8VUf1xuyYieldHQXK...	1	juanito@gmail.com
3	Bebetylta	Bebetylta	0995101231	Zona10	\$2y\$10\$18RofEz/25nTgls1UmN9qPlfHgnzcpZTbsalRml...	1	leonguambo@gmail.com
4	Manuel	Aquil	0999999999	Sur Quito	\$2y\$10\$9QvhJEEmffan9O.pYgX.N3ZG/GUIBdQuCISBB2S...	0	manu@hotmail.com

Fig.23. Base de datos anterior

Fig.24. Perfil Actual

de_id	first_name	last_name	personal_phone	address	password	state	email
4	Wyatt	Dubugue	0937529036	772 Leonora Motorway Apt. 270	\$2y\$10\$mfoc8VTY1tbP3420QRFXwO.FOHmMZl93VmAjyGWRwK...	1	woodrow0@example.com
2	maria	otero	0980955854	Distrito metropolitano	\$2y\$10\$J3ejBoedh7we3Z1H3L.er22QQ3W198N0.UWwTPQ...	0	maria@mail.com
2	Juan	Yorobush	0995102822	Tumbaco	\$2y\$10\$uCFilgyPrin2Ulc26X6eTsc8VUf1xuyYieldHQXK...	1	juanito@gmail.com
3	Leonista	Guambo	0999999999	Quito	\$2y\$10\$18RofEz/25nTgls1UmN9qPlfHgnzcpZTbsalRml...	1	leonguambo@gmail.com
4	Manuel	Aquil	0999999999	Sur Quito	\$2y\$10\$9QvhJEEmffan9O.pYgX.N3ZG/GUIBdQuCISBB2S...	0	manu@hotmail.com

Fig.25. Base de datos Actual

Ahora se comenzará con la creación de los directores, Prisioneros y Guardias, pero sabemos que nuestra API esta desarrollado por medio de la herencia, entonces recordemos que Swagger tiene una lógica de comentario por función para que sean plasmadas, entonces sabremos que existirá una sobreescritura por conflicto de funciones. Entonces vamos a comenzar en desarrollar funciones vacías esto con la funcionalidad de crear funciones que puedan complementar los comentarios Swagger para su respectiva definición y evitar la sobreescritura.

Comenzaremos con DirectorController donde nuestro primer comentario será un GET para visualizar los directores de la base de datos por medio de la función index del CRUD como se visualiza en la Fig.26.

```

12    public function __construct()
13    {
14        // Se procede a establecer el gate
15        // https://laravel.com/docs/9.x/authorization#via-middleware
16        $this->middleware('can:manage-directors');
17        // Se establece el rol para este usuario
18        $role_slug = "director";
19        // Se establece que si puede recibir notificaciones
20        $can_receive_notifications = true;
21        // Se hace uso del controlador padre
22        parent::__construct($role_slug,$can_receive_notifications);
23    }
24
25    /**
26     * @OA\Get(
27     * path="/api/v1/director/",
28     * tags={"OpcionesDirector"},
29     * summary="Listar directores registrados en la base de datos",
30     * operationId="director",
31     * security={
32     * {"passport": {}},
33     * },
34     *
35     * @OA\Response(
36     * response=200,
37     * description="Success",
38     * @OA\MediaType(
39     * mediaType="application/json",
40     * ),
41     * ),
42    */

```

```

    *      description="Success",
    *      @OA\MediaType(
    *          mediaType="application/json",
    *      )
    * ),
    * @OA\Response(
    *     response=401,
    *     description="Unauthenticated"
    * ),
    * @OA\Response(
    *     response=400,
    *     description="Bad Request"
    * ),
    * @OA\Response(
    *     response=404,
    *     description="not found"
    * ),
    * @OA\Response(
    *     response=403,
    *     description="Forbidden"
    * )
    */

    public function indexDir()
    {

    }

    /**
     * * @OA\Post(
     | ** path="/api/v1/director/create",

```

Fig.26. Alistamiento de todos los directores.

Con ello podemos visualizar el funcionamiento de poder ver a los directores como se visualiza en la Fig.27.

The screenshot shows the Swagger UI interface for the `/api/v1/director` endpoint. It includes sections for 'Responses' (containing a 'Curl' command and a 'Request URL' field with the value `http://127.0.0.1:8000/api/v1/director/`) and 'Server response' (showing a 200 status code with a JSON response body). The response body is a list of directors:

```

        [
            {
                "id": 1,
                "username": "Gokuu",
                "first_name": "Kakaroto",
                "last_name": "Son",
                "email": "goku@gmail.com",
                "birthdate": "1999-05-15",
                "home_phone": "123456789",
                "personal_phone": "0995102824",
                "address": "1000 Rosalia Fort. Salta 202"
            }
        ]
    
```

Fig.27. Resultado de Swagger con su visualización.

Ahora dispondremos de crear un comentario Swagger para especificar los parámetros de creación de un director de la función store como se visualiza en la Fig.28.

```

    /**
     * * @OA\Post(
     *     path="/api/v1/director/create",
     *     tags={"OpcionesDirector"},
     *     summary="Crear un nuevo director en la base de datos",
     *     operationId="director-create",
     *     security={
     *         {"passport": {}}
     *     },
     *     @OA\Parameter(
     *         name="username",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="first_name",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="last_name",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="last_name",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="email",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="birthdate",
     *         in="query",
     *         required=true,
     *         @OA\Schema(
     *             type="string"
     *         )
     *     ),
     *     @OA\Parameter(
     *         name="home_phone",
     *         in="query",
     *         required=true,

```

Fig.28. Método POST para la creación de un director en la función store.

Podremos visualizarlo por medio de Swagger como se creó un director en la Fig.29, Fig.30, Fig.31

The screenshot shows the Swagger UI interface for the `/api/v1/director/create` endpoint. It displays a form with the following fields and their values:

- username**: Gokuu
- first_name**: Kakaroto
- last_name**: Son
- email**: goku@gmail.com
- birthdate**: 1999-05-15
- home_phone**: 123456789
- personal_phone**: 0995102824
- address**: Montaña Poco

Fig.29. Interfaz de creación de director.

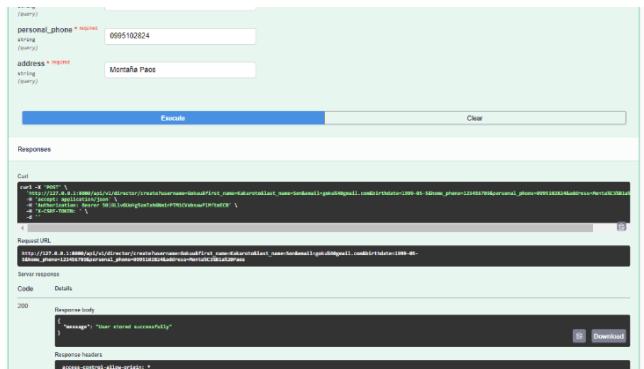


Fig.30. Verificación de creación Swagger.

		Leonista	Guambo	0999999999	Quito	\$2y\$10\$QhobpXrEBYHORyfQ071eyTbOacjvCAhFwpdVKA \$2y\$10\$gQrhJEMmri6an9IO/p1yQXN3ZG/GUIkhdQuCSB82
		Manuel	Auqui	0999999999	Sur Quito	\$2y\$10\$gQrhJEMmri6an9IO/p1yQXN3ZG/GUIkhdQuCSB82
		Kakaroto	Son	0995102824	Montaña	\$2y\$10\$gQrhJEMmri6an9IO/p1yQXN3ZG/GUIkhdQuCSB82 Pass

Fig.31. Director creado en la base de datos.

Crearemos un comentario para decir al Swagger que parámetros se necesitará para obtener un director de la función show en específico como se visualiza en la Fig.32.

```

    /**
     * @OA\Response(
     *      response=200,
     *      description="Success",
     *      @OA\MediaType(
     *          mediaType="application/json",
     *      )
     * ),
     * @OA\Response(
     *      response=401,
     *      description="Unauthenticated"
     * ),
     * @OA\Response(
     *      response=400,
     *      description="Bad Request"
     * ),
     * @OA\Response(
     *      response=404,
     *      description="not found"
     * ),
     * @OA\Response(
     *      response=403,
     *      description="Forbidden"
     * )
    */

    public function showDir()
    {
    }

    /**
     * @OA\Get(
     *      path="/api/v1/director/{user}",
     *      tags={"OpcionesDirector"},
     *      summary="Visualiza a un director en específico",
     *      operationId="director-user-show",
     *      security={
     *          {"passport": {}},
     *      },
     *      @OA\Parameter(
     *          name="user",
     *          in="path",
     *          required=true,
     *          @OA\Schema(
     *              type="integer"
     *          )
     *      ),
     *
     *      @OA\Response(
     *          response=200,
     *          description="Success",
     *          @OA\MediaType(
     *              mediaType="application/json",
     *          )
     *      ),
     *      @OA\Response(
     *          response=401,
     *          description="Unauthenticated"
     *      )
    )

```

Fig.32. Método GET para obtener un director en la función show.

Podemos visualizar su funcionamiento en la Fig.33.

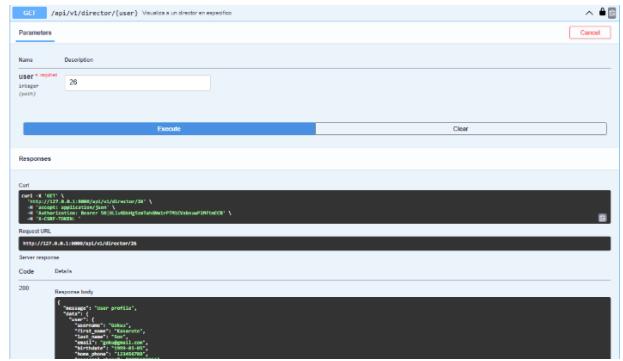


Fig.33. Resultado en la interfaz de Swagger.

Comenzaremos con la Actualización de los directores por medio de múltiples parámetros que le dictaremos a Swagger por medio de nuestros comentarios en la función update como se visualiza en la Fig.34.

```

    /**
     * @OA\Post(
     *      path="/api/v1/director/{user}/update",
     *      tags={"OpcionesDirector"},
     *      summary="Actualiza los datos del director en específico",
     *      operationId="director-user-update",
     *      security={
     *          {"passport": {}},
     *      },
     *      @OA\Parameter(
     *          name="user",
     *          in="path",
     *          required=true,
     *          @OA\Schema(
     *              type="integer"
     *          )
     *      ),
     *
     *      @OA\Parameter(
     *          name="username",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="first_name",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="last_name",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="email",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      )
    )

```

```

    *      in="query",
    *      required=true,
    *      @OA\Schema(
    *          type="string"
    *      )
    * ),
    * @OA\Parameter(
    *     name="email",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="birthdate",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="home_phone",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="personal_phone",
    *     description="Success",
    *     @OAMimeType(
    *         mediaType="application/json",
    *     )
    * ),
    * @OA\Response(
    *     response=401,
    *     description="Unauthenticated"
    * ),
    * @OA\Response(
    *     response=400,
    *     description="Bad Request"
    * ),
    * @OA\Response(
    *     response=404,
    *     description="not found"
    * ),
    * @OA\Response(
    *     response=403,
    *     description="Forbidden"
    * )
*/
public function updateDir()
{
}

```

Fig.34. método POST para la actualización de director en la función update.

Podremos visualizar el éxito del proceso en la Fig.35 y Fig.36.

Fig.35. Interfaz de actualización de Swagger.

	Edt	Copy	Delete	22	2. maria	otero	0980955854	Motonavia	2023-07-11T10:24:02.000Z	2023-07-11T10:24:02.000Z	Apt. 270
<input type="checkbox"/>				23	2. Juan	Yorobush	0995102822	Tumbaco	\$2y\$10\$uCFJgypPrn2ULcb26X6eTsc8VU61v0uyYnUdHC		Districto metropolitano
<input type="checkbox"/>				24	3. Leonisita	Guambo	0999999999	Quito	\$2y\$10\$QubgeXKEBjYH0RyfQ0T1eyTbOaqCAGFwpc		
<input type="checkbox"/>				25	4. Manuel	Auréa	0999999999	Sur Quito	\$2y\$10\$uQvhJEMmfan9Q0py9qXN3ZGjGUBkQjQicfS		
<input type="checkbox"/>				26	2. Goku	Goku	091345678	Kamabouze	\$2y\$10\$uPjijqSyE57QVPMBVQ2en0REW78zKw48MR		

Fig.36. Actualización de directo en base de datos.

Ahora comenzaremos en dar de baja a un director de la función destroy por medio de los comentarios Swagger como se visualiza en la Fig.37.

```

/**
 *  * @OA\Get(
 *      path="/api/v1/director/{user}/destroy",
 *      tags={"OpcionesDirector"},
 *      summary="Da de baja a un director",
 *      operationId="director-user-destroy",
 *      security={
 *          {"passport": {}},
 *      },
 *      @OA\Parameter(
 *          name="user",
 *          in="path",
 *          required=true,
 *          @OA\Schema(
 *              type="integer"
 *          )
 *      ),
 *
 *      @OA\Response(
 *          response=200,
 *          description="Success",
 *          @OAMimeType(
 *              mediaType="application/json",
 *          )
 *      ),
 *      @OA\Response(
 *          response=401,
 *          description="Unauthenticated"
 *      ),
 *      @OA\Response(
 *          response=400,
 *          description="Bad Request"
 *      ),
 *      @OA\Response(
 *          response=404,
 *          description="not found"
 *      ),
 *      @OA\Response(
 *          response=403,
 *          description="Forbidden"
 *      )
 */
public function destDir()
{
}

```

Fig.37. Método GET en la función destroy para inactivar director.

Podemos ver el funcionamiento del método de dar de baja en la Fig.38 y Fig.39.

Fig.38. Comprobaciones Swagger de inactividad.

Graham					
	user	name	birthdate	home_phone	status
y\$105J3nB0d7We3Z1H13LerZQG3W198N0tWwTPQ...	0	maria@gmail.com	023121284	NULL	NULL
y\$105aCfPhyPiH2U2z2X6x7scGV86tXuyYnUJdHQ9X...	1	juanito@gmail.com	Juanito	123456789	1997-05-20 NULL
y\$105QhdqbrXcEBYHQRyfGZ1leyTxDaqj-CMFrwpdvKAh2...	1	leoni@gmail.com	Leoni	123456789	1999-10-10 15:45:58
y\$105QdjhneMhs6ar90.y2g6XK32zG1JBdQjCISB2S...	0	manu@hotmail.com	Manu123	123456789	2001-10-12 NULL
y\$105pfhQdyvE97QVPMBVX2h0QERyTBz2wKABMPJzgPS...	0	gaku@gmail.com	Gaku	123456788	1999-05-05 NULL

Fig.39. Base de datos de inactivar a un director.

Ahora comenzaremos con la creación del Swagger para Guardias por lo que nos enfocaremos en el controlador GuardController y en la función index dispondremos a poner nuestro comentario como se visualiza en la Fig.40 y Fig.41.

```

25 /**
26  *  * @OA\Get(
27  *  ** path="/api/v1/guard/",
28  *  *  tags={"OpcionesGuard"}, 
29  *  *  summary="Lista de Guardias en la base de datos",
30  *  *  operationId="guard",
31  *  * security={
32  *  *   {"passport": {}},
33  *  * },
34  *  *
35  *  * @OA\Response(
36  *  *  response=200,
37  *  *  description="Success",
38  *  *  @OA\MediaType(
39  *  *   mediaType="application/json",
40  *  * )
41  *  ),
42  *  @OA\Response(
43  *  *  response=401,
44  *  *  description="Unauthenticated"
45  *  ),
46  *  @OA\Response(
47  *  *  response=400,
48  *  *  description="Bad Request"
49  *  ),
50  *  @OA\Response(
51  *  *  response=404,
52  *  *  description="not found"
53  *  ),
54  *  @OA\Response(
55  *  *  response=403,
56  *  *  description="Forbidden"
57  *  ),
58  *  @OA\Response(
59  *  *  response=400,
60  *  *  description="Bad Request"
61  *  ),
62  *  @OA\Response(
63  *  *  response=404,
64  *  *  description="not found"
65  *  ),
66  *  @OA\Response(
67  *  *  response=403,
68  *  *  description="Forbidden"
69  *  )
70  */
71
72 public function indexGuard()
73 {
74 }

```

Fig.40. Alistamiento de Guardias con método GET de la función index.

Fig.41. Resultados en Swagger

Ahora comenzaremos con la creación de un Guardia por medio de la función store para el almacenamiento a la base de datos con la ayuda de los comentarios de Swagger de la función store como se visualiza en la Fig.42 .

```

65 /**
66  *  * @OA\Post(
67  *  ** path="/api/v1/guard/create",
68  *  *  tags={"OpcionesGuard"}, 
69  *  *  summary="Crear un nuevo guardia en la base de datos",
70  *  *  operationId="guard-create",
71  *  * security={
72  *  *   {"passport": {}},
73  *  * },
74  *  * @OA\Parameter(
75  *  *   name="username",
76  *  *   in="query",
77  *  *   required=true,
78  *  *   @OA\Schema(
79  *  *     type="string"
80  *  *   )
81  *  * ),
82  *  * @OA\Parameter(
83  *  *   name="first_name",
84  *  *   in="query",
85  *  *   required=true,
86  *  *   @OA\Schema(
87  *  *     type="string"
88  *  *   )
89  *  * ),
90  *  * @OA\Parameter(
91  *  *   name="last_name",
92  *  *   in="query",
93  *  *   required=true,
94  *  *   @OA\Schema(
95  *  *     type="string"
96  *  *   )
97  *  * ),
98  *  * @OA\Parameter(
99  *  *   name="email",
100 *  *   in="query",
101 *  *   required=true,
102 *  *   @OA\Schema(
103 *  *     type="string"
104 *  *   )
105 *  * ),
106 *  * ),
107 *  * @OA\Parameter(
108 *  *   name="birthdate",
109 *  *   in="query",
110 *  *   required=true,
111 *  *   @OA\Schema(
112 *  *     type="string"
113 *  *   )
114 *  * ),
115 *  * @OA\Parameter(
116 *  *   name="home_phone",
117 *  *   in="query",
118 *  *   required=true,
119 *  *   @OA\Schema(

```

```

    * @OA\Parameter(
    *   name="personal_phone",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Parameter(
    *   name="address",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    *
    * @OA\Response(
    *   response=200,
    *   description="Success",
    *   @OA\MediaType(
    *     mediaType="application/json",
    *   )
    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request",
    *   response=200,
    *   description="Success",
    *   @OA\MediaType(
    *     mediaType="application/json",
    *   )
    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request"
    * ),
    * @OA\Response(
    *   response=404,
    *   description="not found"
    * ),
    * @OA\Response(
    *   response=403,
    *   description="Forbidden"
    * )
  */
}

public function storeGuard()
{
}

```

Fig.42. Método POST en creación de Guardia de la función store.

Podremos visualizar su creación en la Fig.43 y Fig.44.

Fig.43. Interfaz de Swagger en la creación de guardia.

Fig.44.

Podremos visualizar un Guardia en específico por medio de la función show y los comentarios de Swagger para definir sus parametros como se visualiza en la Fig.45.

```

  * * @OA\Get(
  *   path="/api/v1/guard/{user}",
  *   tags={"OpcionesGuard"},
  *   summary="Visualiza a un guardia en específico",
  *   operationId="guard-user-show",
  *   security={
  *     {"passport": {}},
  *   },
  *   @OA\Parameter(
  *     name="user",
  *     in="path",
  *     required=true,
  *     @OA\Schema(
  *       type="integer"
  *     )
  *   ),
  *
  *   @OA\Response(
  *     response=200,
  *     description="Success",
  *     @OA\MediaType(
  *       mediaType="application/json",
  *     )
  *   ),
  *   @OA\Response(
  *     response=401,
  *     description="Unauthenticated"
  *   ),
  *   @OA\Response(
  *     response=200,
  *     description="Success",
  *     @OA\MediaType(
  *       mediaType="application/json",
  *     )
  *   ),
  *   @OA\Response(
  *     response=401,
  *     description="Unauthenticated"
  *   ),
  *   @OA\Response(
  *     response=400,
  *     description="Bad Request"
  *   ),
  *   @OA\Response(
  *     response=404,
  *     description="not found"
  *   ),
  *   @OA\Response(
  *     response=403,
  *     description="Forbidden"
  *   )
  */

  public function showGuard()
  {
  }

```

Fig.45. Método GET en la función show para visualizar a un guardia.

Podremos verificar su funcionalidad por medio de Swagger como se visualiza en la Fig.46.

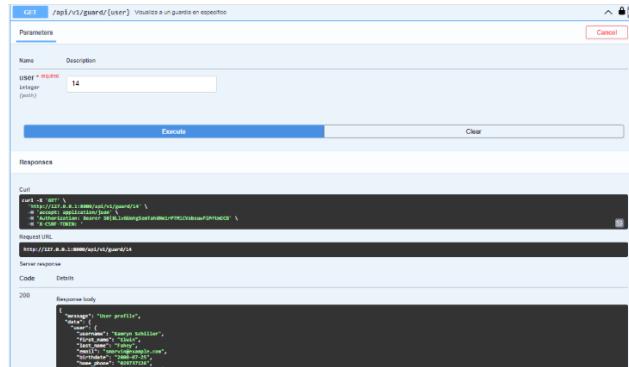


Fig.46. Comprobación de resultados en Swagger.

Podremos actualizar nuestro guardia por medio de la función update y los comentarios Swagger para especificar el método que debe realizarse como se visualiza en la Fig.47.

```
/*
 *  * @OA\Post(
 *  *   path="/api/v1/guard/{user}/update",
 *  *   tags={"OpcionesGuard"}, 
 *  *   summary="Actualiza un guardia en específico",
 *  *   operationId="guard-user-update",
 *  * security={
 *  *     {"passport": {}},
 *  *   },
 *  *   @OA\Parameter(
 *  *     name="user",
 *  *     in="path",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="integer"
 *  *     )
 *  *   ),
 *  *
 *  *   @OA\Parameter(
 *  *     name="username",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="first_name",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="last_name",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="email",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="birthdate",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="home_phone",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="personal_phone",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   )
 *  * )
 *  */

```

```
  *   ),
 *  *   @OA\Parameter(
 *  *     name="username",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="first_name",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="last_name",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="email",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="birthdate",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="home_phone",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   ),
 *  *   @OA\Parameter(
 *  *     name="personal_phone",
 *  *     in="query",
 *  *     required=true,
 *  *     @OA\Schema(
 *  *       type="string"
 *  *     )
 *  *   )
 *  * )
 *  */

```

```

    * @OA\Response(
    *   response=200,
    *   description="Success",
    *   @OA\MediaType(
    *     mediaType="application/json",
    *   )
    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request"
    * ),
    * @OA\Response(
    *   response=404,
    *   description="not found"
    * ),
    * @OA\Response(
    *   response=403,
    *   description="Forbidden"
    * )
  */
}

public function updateGuard()
{
}
*/

```

Fig.47. Función update con Método POST para la actualización de guardia

Podremos verificarlo por medio de Swagger su actualización y comprobarlo en la base de datos como se visualiza en la Fig.48, Fig.49 y Fig.50.

Fig.48. Verificación de actualización en Swagger

Fig.49. Actualización exitosa Swagger.

Copy	Delete	z.	mana	otero	U9607bb5b04	metropolitano	2zy310sJ3npoedh7tre3c1H1fSLenLzzLJUJRWV195WUJUWw1PM2...	0	r
Copy	Delete	23	2	Juan	Yorobusto	0995102622	Tumbaco \$2y\$10\$eCFlgyPi/n2UJLch2lXGeTscBVU61v0uyYnJdHQ9X...	1	j
Copy	Delete	24	3	Marcela	Guambo	0999998741	Zona10 \$2y\$10\$QsbubrgxZEBHYQR/Q071nyTnOqvCAGFwpdKKAHZ...	1	h
Copy	Delete	25	4	Manuel	Augúl	0999999999	Sur Quito \$2y\$10\$gQnJEMmGanGOpjy9hN3Z2GUlBkjduqCISBBZS...	0	r
Copy	Delete	26	2	Goku	Goku	0912345678	Kamehouse \$2y\$10\$PjfqSywE07GVPMII/X2hC9RE/WTB/c2eKABMPJrzgPS...	0	t

Fig.50. Actualización en la base de datos.

Como siguiente paso podremos visualizar el siguiente comentario de Swagger donde vamos a listar todos los presos por medio de la función index como se visualiza en la Fig.51.

```

/**
 *  * @OA\Get(
 *   path="/api/v1/prisoner/",
 *   tags={"OpcionesPrisionero"},
 *   summary="Lista de prisioneros existentes",
 *   operationId="prisioneros",
 *   security={
 *     {"passport": {}},
 *   },
 *   *
 *   @OA\Response(
 *     response=200,
 *     description="Success",
 *     @OA\MediaType(
 *       mediaType="application/json",
 *     )
 *   ),
 *   @OA\Response(
 *     response=401,
 *     description="Unauthenticated"
 *   ),
 *   @OA\Response(
 *     response=400,
 *     description="Bad Request"
 *   ),
 *   @OA\Response(
 *     response=404,
 *     description="not found"
 *   ),
 *   @OA\MediaType(
 *     mediaType="application/json",
 *   )
 * ),
 * @OA\Response(
 *   response=401,
 *   description="Unauthenticated"
 * ),
 * @OA\Response(
 *   response=400,
 *   description="Bad Request"
 * ),
 * @OA\Response(
 *   response=404,
 *   description="not found"
 * ),
 * @OA\Response(
 *   response=403,
 *   description="Forbidden"
 * )
 */
public function indexPri()
{
}

```

Fig.51. Lista de todos los presos con método GET en la función index.

Podemos visualizar su despliegue por Swagger como se visualiza en la Fig.52.

The screenshot shows the Swagger UI interface for the `/api/v1/prisoner` endpoint. It includes fields for 'Parameters' (None), 'Responses' (200), and a 'Code Details' section showing the generated code for the response.

```

GET /api/v1/prisoner/ Lista de prisioneros existentes
Parameters
No parameters
Responses
200
Response body
[{"id": 1, "name": "Pédro Salchpapa", "first_name": "Pedro", "last_name": "Loano", "email": "pepe@gmail.com", "birthdate": "1990-12-1", "home_phone": "123456789", "personal_phone": "09123456781", "address": "Sauces"}, {"id": 2, "name": "Miguelito", "first_name": "Miguelito", "last_name": "Bolívar", "email": "miguelito@gmail.com", "birthdate": "1995-05-20", "home_phone": "123456789", "personal_phone": "09123456781", "address": "Sauces"}]
  
```

Fig.52. Resultado index de Swagger.

En las Fig.53 podremos visualizar por medio de los comentarios Swagger y la función store la creación de un prisionero.

```

/**
 *  * @OA\Post(
 *  ** path="/api/v1/prisoner/create",
 *  *  tags={"OpcionesPrisionero"},
 *  *  summary="Crear un nuevo prisionero en la base de datos",
 *  *  operationId="prisionero-create",
 *  *security={
 *  *  {"passport": {}},
 *  *  },
 *  * @OA\Parameter(
 *  *  name="username",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="first_name",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="last_name",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="last_name",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="email",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="birthdate",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="home_phone",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="personal_phone",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Parameter(
 *  *  name="address",
 *  *  in="query",
 *  *  required=true,
 *  *  @OA\Schema(
 *  *    type="string"
 *  *  )
 *  *  ),
 *  * @OA\Response(
 *  *  response=200,
 *  *  description="Success",
 *  *  @OA\MediaType(
 *  *    mediaType="application/json",
 *  *  )
 *  *  ),
 *  *  description= Success ,
 *  *  @OA\MediaType(
 *  *    mediaType="application/json",
 *  *  )
 *  *  ),
 *  * @OA\Response(
 *  *  response=401,
 *  *  description="Unauthenticated"
 *  *  ),
 *  * @OA\Response(
 *  *  response=400,
 *  *  description="Bad Request"
 *  *  ),
 *  * @OA\Response(
 *  *  response=404,
 *  *  description="not found"
 *  *  ),
 *  * @OA\Response(
 *  *  response=403,
 *  *  description="Forbidden"
 *  *  )
 *  * )
 */
public function storePri()
{
}
  
```

The screenshot shows the Swagger UI interface for the `/api/v1/prisoner/create` endpoint. It includes fields for 'Parameters' (username, first_name, last_name, email, birthdate, home_phone, personal_phone, address) and a 'Code Details' section showing the generated code for the response.

```

POST /api/v1/prisoner/create Crear un nuevo prisionero en la base de datos
Parameters
Name Description
username * required Pédro Salchpapa
first_name * required Pedro
last_name * required Loano
email * required pepe@gmail.com
birthdate * required 1990-12-1
home_phone * required 123456789
personal_phone * required 09123456781
address * required Sauces
  
```

Fig.53. Método POST en la creación de un prisionero en la función store.

Podemos comprobar la creación del prisionero en Swagger por medio de la Fig.54 y Fig.55.

The screenshot shows the Swagger UI interface for the `/api/v1/prisoner/create` endpoint. It includes fields for 'Parameters' (username, first_name, last_name, email, birthdate, home_phone, personal_phone, address) and a 'Code Details' section showing the generated code for the response.

```

POST /api/v1/prisoner/create Crear un nuevo prisionero en la base de datos
Parameters
Name Description
username * required Pédro Salchpapa
first_name * required Pedro
last_name * required Loano
email * required pepe@gmail.com
birthdate * required 1990-12-1
home_phone * required 123456789
personal_phone * required 09123456781
address * required Sauces
  
```

Fig.54. Comprobación en Swagger de actualizacion.

	id	role_id	first_name	last_name	personal_phone	address	password
Delete	27	3	Valencia	Valencia	0123456789	Sur de Quito	\$2y\$10\$RIRWxy7SGoASPuWFg:
Delete	28	4	Pedro	Loano	0912358741	Sauces	\$2y\$10\$2EMtCupjXT6MRWY58SS

With selected: Edit Conv Delete Export

Fig.55. Actualización de base de datos de prisioneros.

En esta función show podremos visualizar un prisionero en específico como se muestra en la Fig.56.

```

167     /**
168      * * @OA\Get(
169      *  path="/api/v1/prisoner/{user}",
170      *  tags={"OpcionesPrisionero"},
171      *  summary="Visualiza al prisionero de la base de datos",
172      *  operationId="prisionero-user-show",
173      *  security={
174      *    {"passport": {}},
175      *  },
176      *  @OA\Parameter(
177      *    name="user",
178      *    in="path",
179      *    required=true,
180      *    @OA\Schema(
181      *      type="integer"
182      *    )
183      *  ),
184      *
185      *  @OA\Response(
186      *    response=200,
187      *    description="Success",
188      *    @OA\MediaType(
189      *      mediaType="application/json",
190      *    )
191      *  ),
192      *  @OA\Response(
193      *    response=401,
194      *    description="Unauthenticated"
195      *  ),
196      *  @OA\Response(
197      *    response=200,
198      *    description="Success",
199      *    @OA\MediaType(
200      *      mediaType="application/json",
201      *    )
202      *  ),
203      *  @OA\Response(
204      *    response=401,
205      *    description="Unauthenticated"
206      *  ),
207      *  @OA\Response(
208      *    response=400,
209      *    description="Bad Request"
210      *  ),
211      *  @OA\Response(
212      *    response=404,
213      *    description="not found"
214      *  ),
215      *  @OA\Response(
216      *    response=403,
217      *    description="Forbidden"
218      *  )
219      * )
220  */
221
222  public function showPri()
223  {
224
225  }

```

Fig.56. Método GET en la visualización de un prisionero.

Podremos comprobar su funcionalidad en la Fig.57.

The screenshot shows the Swagger UI interface for the 'GET /api/v1/prisoner/{user}' endpoint. The URL in the address bar is `http://127.0.0.1:5000/api/v1/prisoner/28`. The response status is 200, and the response body is a JSON object representing a user profile:

```

{
  "username": "User profile",
  "first_name": "Pedro",
  "last_name": "Loano salazar",
  "last_name_2": "Loano",
  "middle_name": "Salazar",
  "middle_name_2": "Salazar",
  "personal_phone": "+519 33 12 33 33"
}

```

Fig.57. Funcionalidad en Swagger.

En la función update podemos visualizar como se introduce un comentario Swagger para los parámetros de actualización como se visualiza en la Fig.58.

```

/***
 *  * @OA\Post(
 *  path="/api/v1/prisoner/{user}/update",
 *  tags={"OpcionesPrisionero"},
 *  summary="Actualiza los datos del prisionero",
 *  operationId="prisionero-user-update",
 *  security={
 *    {"passport": {}},
 *  },
 *  @OA\Parameter(
 *    name="user",
 *    in="path",
 *    required=true,
 *    @OA\Schema(
 *      type="integer"
 *    )
 *  ),
 *  @OA\Parameter(
 *    name="username",
 *    in="query",
 *    required=true,
 *    @OA\Schema(
 *      type="string"
 *    )
 *  ),
 *  @OA\Parameter(
 *    name="first_name",
 *    in="query",
 *    required=true,
 *    @OA\Schema(
 *      type="string"
 *    )
 *  ),
 *  @OA\Parameter(
 *    name="first_name",
 *    in="query",
 *    required=true,
 *    @OA\Schema(
 *      type="string"
 *    )
 *  ),
 *  @OA\Parameter(
 *    name="last_name",
 *    in="query",
 *    required=true,
 *    @OA\Schema(
 *      type="string"
 *    )
 *  ),
 *  @OA\Parameter(
 *    name="email",
 *    in="query",
 *    required=true,
 *    @OA\Schema(
 *      type="string"
 *    )
 *  )
*/

```

```

    *      required=true,
    *      @OA\Schema(
    *          type="string"
    *      )
    * ),
    * @OA\Parameter(
    *     name="email",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="birthdate",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="home_phone",
    *     in="query",
    *     required=true,
    *     @OA\Schema(
    *         type="string"
    *     )
    * ),
    * @OA\Parameter(
    *     name="personal_phone",
    *     in="query",
    *     @OA\Response(
    *         response=200,
    *         description="Success",
    *         @OA\MediaType(
    *             mediaType="application/json",
    *         )
    *     ),
    *     @OA\Response(
    *         response=401,
    *         description="Unauthenticated"
    *     ),
    *     @OA\Response(
    *         response=400,
    *         description="Bad Request"
    *     ),
    *     @OA\Response(
    *         response=404,
    *         description="not found"
    *     ),
    *     @OA\Response(
    *         response=403,
    *         description="Forbidden"
    *     )
    * )
*/
    public function updatepri()
    {
    }

```

Fig.58. Método POST en la actualización de un preso en la función update.

Podemos comprobarlo por Swagger su función de actualización en la Fig.59, Fig.60 y Fig.61.

Fig.59. Interfaz Swagger en la actualización de preso.

Fig.60. Confirmación de preso en Swagger.

Fig.61. Actualización en la base de datos.

En la función destroy, el comentario Swagger dará la orden de documentar la inactividad de un preso como se visualiza en la Fig.62.

```

    * * @OA\Get(
    ** path="/api/v1/prisoner/{user}/destroy",
    *  tags={"OpcionesPrisionero"},
    *  summary="Da de baja a un prisionero",
    *  operationId="prisionero-user-destroy",
    * security={
    *   {"passport": {}},
    * },
    * @OA\Parameter(
    *   name="user",
    *   in="path",
    *   required=true,
    *   @OA\Schema(
    *     type="integer"
    *   )
    * ),
    *
    * @OA\Response(
    *   response=200,
    *   description="Success",
    *   @OA\MediaType(
    *     mediaType="application/json",
    *   )
    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(

```

```

    *      mediaType="application/json",
    *  ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request"
    * ),
    * @OA\Response(
    *   response=404,
    *   description="not found"
    * ),
    * @OA\Response(
    *   response=403,
    *   description="Forbidden"
    * )
  */
  public function destroy()
  {
  }
}

```

Fig.62. Método GET en la inactividad de preso de la función destroy.

Podemos ver la funcionalidad de inactividad en la Fig.63 y Fig.64.

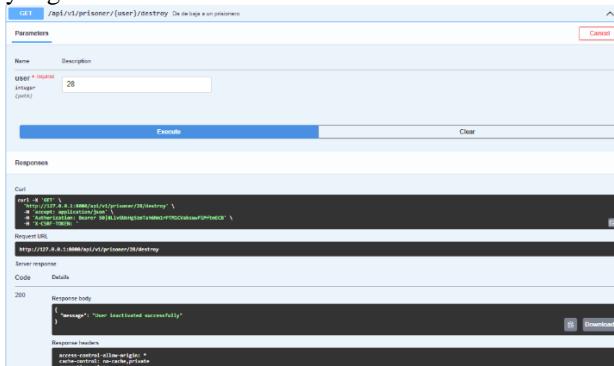


Fig.63. Confirmación de inactividad Swagger.

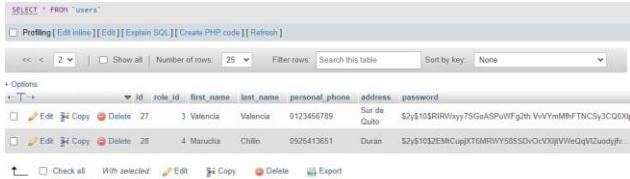


Fig.64. Base de datos de inactividad

Ahora dispondremos de crear un comentario Swagger en la función index con la funcionalidad de poder desplegar las cárceles de la base de datos en el controlador JailController. Como se visualiza en la Fig.65.

```

20 // Métodos del Controlador
21 // Listar las cárceles
22
23 /**
24  *  * @OA\Get(
25  *   path="/api/v1/jail/",
26  *   tags={"OpcionesJail"},
27  *   summary="Listar cárceles totales",
28  *   operationId="jail",
29  *   security={
30  *     {"passport": {}},
31  *   },
32  *
33  *   @OA\Response(
34  *     response=200,
35  *     description="Success",
36  *     @OA\MediaType(
37  *       mediaType="application/json",
38  *     )
39  *   ),
40  *   @OA\Response(
41  *     response=401,
42  *     description="Unauthenticated"
43  *   ),
44  *   @OA\Response(
45  *     response=400,
46  *     description="Bad Request"
47  *   ),
48  *   @OA\Response(
49  *     response=404,
50  *     description="not found"
51  *   ),
52  *   @OA\Response(
53  *     response=403,
54  *     description="Forbidden"
55  *   )
56  */
57 /**
58  public function index()
59  {
60    // Obtener la colección de cárceles
61    $jails = Jail::orderBy('name', 'asc')->get();
62
63    // Invoca el controlador padre para la respuesta json
64    // El moldeo de la información por el Resource
65    return $this->sendResponse($message: 'Jail list generated successfully', $result: [
66      'jails' => SpaceResource::collection($jails)
67    ]);
68  }
69 // Crear una nueva cárcel
70

```

Fig.65. Método GET para visualizar la lista de jails de la función index.

En la función store vamos a crear un comentario Swagger para que la plataforma sepa los parámetros de ejecución como se visualiza en la Fig.66.

```

// Crear una nueva cárcel
/**
*  * @OA\Post(
*   path="/api/v1/jail/create",
*   tags={"OpcionesJail"},
*   summary="Crear una nueva cárcel para el preso",
*   operationId="Jail-create",
*   security={
*     {"passport": {}},
*   },
*   @OA\Parameter(
*     name="name",
*     in="query",
*     required=true,
*     @OA\Schema(
*       type="string"
*     )
*   ),
*   @OA\Parameter(
*     name="code",
*     in="query",
*     required=true,
*     @OA\Schema(
*       type="string"
*     )
*   )

```

```

    *      type="string"
    *    ),
    *  ),
    * @OA\Parameter(
    *   name="type",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Parameter(
    *   name="capacity",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Parameter(
    *   name="ward_id",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Parameter(
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Parameter(
    *   name="description",
    *   in="query",
    *   required=true,
    *   @OA\Schema(
    *     type="string"
    *   )
    * ),
    * @OA\Response(
    *   response=200,
    *   description="Success",
    *   @OA\MediaType(
    *     mediaType="application/json"
    *   )
    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request"
    * ),
    * @OA\Response(
    *   response=404,
    *   description="not found"
    * ),
    * @OA\Response(
    *   response=403,
    *   description="Forbidden"
    * )
  */
}

public function store(Request $request)
{
  // Validación de los datos de entrada
  // Crear un array asociativo de clave y valor
  $jail_data = $request->validate([
    'name' => ['required', 'string', 'min:3', 'max:45'],
    // https://laravel.com/docs/9.x/validationrule-alpha-dash
    'code' => ['required', 'string', 'alpha_dash', 'min:5', 'max:45'],
    'type' => ['required', 'string'],
    'capacity' => ['required', 'string', 'numeric', 'digits:1', 'min:2', 'max:5'],
    // https://laravel.com/docs/9.x/validationrule-exists
    'ward_id' => ['required', 'string', 'numeric', 'exists:wards,id'],
    'description' => ['nullable', 'string', 'min:5', 'max:255'],
  ]);
}

```

Fig.66. Creación de jail con método POST de la función store.

En la función show vamos a crear un comentario Swagger para la ejecución de visualizar una cárcel en específico como se visualiza en la Fig.67

```

177  // Mostrar la información de la cárcel
178
179 /**
180  *  * @OA\Get(
181  *   path="/api/v1/jail/{jail}",
182  *   tags={"OpcionesJail"},
183  *   summary="Informacion de la carcel en visitar",
184  *   operationId="Jail-show",
185  *   security={
186  *     {"passport": {}},
187  *   },
188  *   @OA\Parameter(
189  *     name="jail",
190  *     in="path",
191  *     required=true,
192  *     @OA\Schema(
193  *       type="integer"
194  *     )
195  *   ),
196  *
197  *
198  *   @OA\Response(
199  *     response=200,
200  *     description="Success",
201  *     @OA\MediaType(
202  *       mediaType="application/json",
203  *     )
204  *   ),
205  *   @OA\Response(
206  *     response=401,
207  *     description="Unauthenticated"
208  *   ),
209  *   @OA\Response(
210  *     response=400,
211  *     description="Bad Request"
212  *   ),
213  *   @OA\Response(
214  *     response=404,
215  *     description="not found"
216  *   ),
217  *   @OA\Response(
218  *     response=403,
219  *     description="Forbidden"
220  *   )
221  */
222
223 public function show(Jail $jail)
224 {
225   // Invoca el controlador padre para la respuesta json
226   // El moldeo de la información por el Resource
227   return $this->sendResponse(message: 'Jail details', result: [
228     'jail' => new JailResource($jail)
229   ]);
230 }

```

Fig.67. Visualizar un Jail específico con método GET de la función show.

Vamos a crear un comentario Swagger para que la función update de los parámetros y se pueda ejecutar en la plataforma como se visualiza en la Fig.68

```

// Actualizar la información de la cárcel
/*
 *  * @OA\Post(
 *    path="/api/v1/jail/{jail}/update",
 *    tags={"OpcionesJail"},
 *    summary="La cárcel sera actualizada en específico",
 *    operationId="jail-user-update",
 *    security={
 *      {"passport": {}},
 *      },
 *      @OA\Parameter(
 *        name="jail",
 *        in="path",
 *        required=true,
 *        @OA\Schema(
 *          type="integer"
 *        )
 *      ),
 *
 *      @OA\Parameter(
 *        name="code",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *      @OA\Parameter(
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *      @OA\Parameter(
 *        name="type",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *      @OA\Parameter(
 *        name="capacity",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *      @OA\Parameter(
 *        name="ward_id",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *      @OA\Parameter(
 *        name="description",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *          type="string"
 *        )
 *      ),
 *
 *      @OA\Response(
 *        response=200,
 *        description="Success",
 *        @OA\MediaType(
 *          mediaType="application/json",
 *        )
 *      ),
 *      @OA\Response(
 *        response=401,
 *        description="Unauthenticated"
 *      ),
 *      @OA\Response(
 *        response=400,
 *        description="Bad Request"
 *      ),

```

```

 *      response=404,
 *      description="not found"
 *    ),
 *    @OA\Response(
 *      response=403,
 *      description="Forbidden"
 *    )
 *  )
 */
}

public function update(Request $request, Jail $jail)
{
    // Validación de los datos de entrada
    // Crear un array asociativo de clave y valor
    $jail_data = $request->validate([
        'name' => ['required', 'string', 'min:3', 'max:45'],
        // https://laravel.com/docs/9.x/validation#rule-alpha-dash
        'code' => ['required', 'string', 'alpha_dash', 'min:5', 'max:45'],
        // https://laravel.com/docs/9.x/validation#rule-digits
        'capacity' => ['required', 'string', 'numeric', 'digits:1', 'min:2', 'max:5'],
        // https://laravel.com/docs/9.x/validation#rule-exists
        'ward_id' => ['required', 'string', 'numeric', 'exists:wards,id'],
        'description' => ['nullable', 'string', 'min:5', 'max:255'],
    ]);
}

```

Fig.68 Actualización de método POST para actualizar el Jail de la función update

Vamos a comenzar con la creación de un comentario Swagger para la ejecución de dar de baja una cárcel de la función destroy como se visualiza en la Fig.69.

```

/**
 *  * @OA\Get(
 *    path="/api/v1/jail/{jail}/destroy",
 *    tags={"OpcionesJail"},
 *    summary="Da de baja una carcel de la base de datos",
 *    operationId="jail-user-destroy",
 *    security={
 *      {"passport": {}},
 *      },
 *      @OA\Parameter(
 *        name="jail",
 *        in="path",
 *        required=true,
 *        @OA\Schema(
 *          type="integer"
 *        )
 *      ),
 *
 *      @OA\Response(
 *        response=200,
 *        description="Success",
 *        @OA\MediaType(
 *          mediaType="application/json",
 *        )
 *      ),
 *
 *      @OA\Response(
 *        response=404,
 *        description="not found"
 *      ),
 *      @OA\Response(
 *        response=403,
 *        description="Forbidden"
 *      )
 */
}

public function destroy(Jail $jail)
{
    // Obtener el estado de la cárcel
    $jail_state = $jail->state;

    // almacenar un string con el mensaje
    $message = $jail_state ? 'inactivated' : 'activated';

    // Verifica que la cárcel tiene prisioneros
    if ($jail->users->isNotEmpty())
    {
        // Invoca el controlador padre para la respuesta json
        return $this->sendResponse($message, code: 403);
    }
}

```

Fig.69 Inactividad de Jail con método GET en la función destroy.

En la sección Ward de cárceles vamos a crear un comentario Swagger para introducir parámetros de visualizar las cárceles totales de la base de datos por medio de la función index como se visualiza en la Fig.70

```

    /**
     * @OA\Get(
     *      path="/api/v1/ward/",
     *      tags={"OpcionesWard"},
     *      summary="Listar las cárceles del sistema",
     *      operationId="ward",
     *      security={
     *          {"passport": {}},
     *      },
     *
     *      @OA\Response(
     *          response=200,
     *          description="Success",
     *          @OA\MediaType(
     *              mediaType="application/json",
     *          )
     *      ),
     *      @OA\Response(
     *          response=401,
     *          description="Unauthenticated"
     *      ),
     *      @OA\Response(
     *          response=400,
     *          description="Bad Request"
     *      ),
     *      @OA\Response(
     *          response=404,
     *          description="not found"
     *      ),
     *      @OA\Response(
     *          response=403,
     *          description="Forbidden"
     *      )
     * )
    */
    public function index()
    {
        // Obtener la colección de pabellones
        $wards = Ward::orderBy('name', 'asc')->get();
        // Invoca el controlador padre para la respuesta json
        // El molde de la información por el Resource
        return $this->sendResponse(message: 'Ward list generated successfully', result: [
            'wards' => SpaceResource::collection($wards)
        ]);
    }
}

```

Fig.70. Función index con el método GET para visualizar Ward.

Se puede visualizar en la Fig.71 su funcionalidad en Swagger

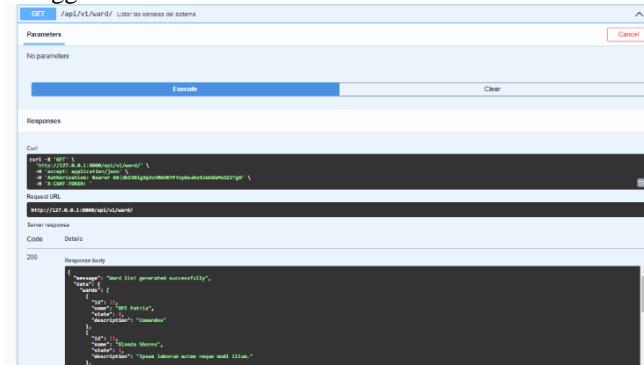


Fig.71 Funcionalidad de Swagger en ward

Vamos a crear el comentario Swagger para la creación de una cárcel en la función store para enviar las instrucciones a la aplicación como se visualiza en la Fig.72.

```

    /**
     * @OA\Post(
     *      path="/api/v1/ward/create",
     *      tags={"OpcionesWard"},
     *      summary="Crear un nuevo registro de cárcel",
     *      operationId="ward-create",
     *      security={
     *          {"passport": {}},
     *      },
     *      @OA\Parameter(
     *          name="name",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="location",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="description",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="location",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      ),
     *      @OA\Parameter(
     *          name="description",
     *          in="query",
     *          required=true,
     *          @OA\Schema(
     *              type="string"
     *          )
     *      )
     * )
    */
    @OA\Response(
        response=200,
        description="Success",
        @OA\MediaType(
            mediaType="application/json",
        )
    ),
    @OA\Response(
        response=401,
        description="Unauthenticated"
    ),
    @OA\Response(
        response=400,
        description="Bad Request"
    )
}

```

```

    * ),
    * @OA\Response(
    *   response=401,
    *   description="Unauthenticated"
    * ),
    * @OA\Response(
    *   response=400,
    *   description="Bad Request"
    * ),
    * @OA\Response(
    *   response=404,
    *   description="not found"
    * ),
    * @OA\Response(
    *   response=403,
    *   description="Forbidden"
    * )
  */
}

// Crear un nuevo pabellon
public function store(Request $request)
{
    // Validación de los datos de entrada
    // Crear un array asociativo de clave y valor
    $ward_data = $request->validate([
        'name' => ['required', 'string', 'min:3', 'max:45'],
        'location' => ['required', 'string', 'min:3', 'max:45'],
        'description' => ['nullable', 'string', 'min:5', 'max:255'],
    ]);

    // https://laravel.com/docs/9.x/eloquent#inserts
    Ward::create($ward_data);
}

```

Fig.72. Método POST en la creación de un ward.

Se puede visualizar en Swagger su creación de ward como se visualiza en la Fig.73.

Fig.73. Confirmación del método crear ward en Swagger

En las Fig.74. vamos a crear un comentario en la función show poder visualizar una cárcel específica de la base de datos por medio de la creación de un comentario Swagger.

```

    /**
     * @OA\Get(
     *   path="/api/v1/ward/{user}",
     *   tags={"OpcionesWard"},
     *   summary="Visualiza el contenido de un registro específico de la cárcel",
     *   operationId="ward-user-show",
     *   security={
     *     {"passport": {}},
     *   },
     *   @OA\Parameter(
     *     name="user",
     *     in="path",
     *     required=true,
     *     @OA\Schema(
     *       type="integer"
     *     )
     *   ),
     *
     *   @OA\Response(
     *     response=200,
     *     description="Success",
     *     @OA\MediaType(
     *       mediaType="application/json",
     *     )
     *   ),
     *   @OA\Response(
     *     response=401,
     *     description="Unauthenticated"
     *   ),
     *   @OA\Response(
     *     response=404,
     *     description="not found"
     *   ),
     *   @OA\Response(
     *     response=403,
     *     description="Forbidden"
     *   )
   */
}

// Mostrar la información del pabellón
public function show(Ward $ward)
{
    // Invoca el controlador padre para la respuesta json
    // El moldeado de la información por el Resource
    return $this->sendResponse(message: 'Ward details', result: [
        'ward' => new WardResource($ward)
    ]);
}

```

Fig.74. Método GET para la visualización de un ward de la función show.

Se puede visualizar su vista de la cárcel específica en la Fig.75.

Fig.75. Funcionalidad en Swagger

En esta sección utilizaremos la función update para crear un comentario Swagger para dar las instrucciones de actualización como se visualiza en la Fig.76.

```

97 /**
98  *  * @OA\Post(
99  *      path="/api/v1/ward/{user}/update",
100 *      tags={"OpcionesWard"},
101 *      summary="Actualiza un registro específico de la cárcel",
102 *      operationId="ward-user-update",
103 *      security={
104  *          {"passport": {}},
105  *      },
106  *      @OA\Parameter(
107  *          name="user",
108  *          in="path",
109  *          required=true,
110  *          @OA\Schema(
111  *              type="integer"
112  *          )
113  *      ),
114  *      @OA\Parameter(
115  *          name="name",
116  *          in="query",
117  *          required=true,
118  *          @OA\Schema(
119  *              type="string"
120  *          )
121  *      ),
122  *      @OA\Parameter(
123  *          name="location",
124  *          in="query",
125  *          required=true,
126  *          @OA\Schema(
127  *              type="string"
128  *          )
129  *      ),
130  *      @OA\Parameter(
131  *          name="location",
132  *          in="query",
133  *          required=true,
134  *          @OA\Schema(
135  *              type="string"
136  *          )
137  *      ),
138  *      @OA\Parameter(
139  *          name="description",
140  *          in="query",
141  *          required=true,
142  *          @OA\Schema(
143  *              type="string"
144  *          )
145  *      ),
146  *      @OA\Response(
147  *          response=200,
148  *          description="Success",
149  *          @OA\MediaType(
150  *              mediaType="application/json",
151  *          )
152  *      ),
153  *      @OA\Response(
154  *          response=401,
155  *          description="Unauthenticated"
156  *      ),
157  *      @OA\Response(
158  *          response=400,
159  *          description="Bad Request"
160  *      ),
161  *      @OA\Response(
162  *          description="Bad Request"
163  *      ),
164  *      @OA\Response(
165  *          response=404,
166  *          description="not found"
167  *      ),
168  *      @OA\Response(
169  *          response=403,
170  *          description="Forbidden"
171  *      )
172  *  )
173 /**
174  * Actualizar la información del pabellón
175 public function update(Request $request, Ward $ward)
176 {
177     // Validación de los datos de entrada
178     // Crear un array asociativo de clave y valor
179     $ward_data = $request->validate([
180         'name' => ['required', 'string', 'min:3', 'max:45'],
181         'location' => ['required', 'string', 'min:3', 'max:45'],
182         'description' => ['nullable', 'string', 'min:5', 'max:255'],
183     ]);
184     // Actualiza los datos del pabellón
185     $ward->fill($ward_data)->save();
186     // Invoca el controlador padre para la respuesta json
187     return $this->sendResponse(message: "Ward updated successfully");
188 }
189 */

```

Fig.76. Actualización de ward con método POST de la función update.

Se puede evidenciar su actualización del ward en Swagger como se visualiza en la Fig.77, Fig.78y Fig.79.

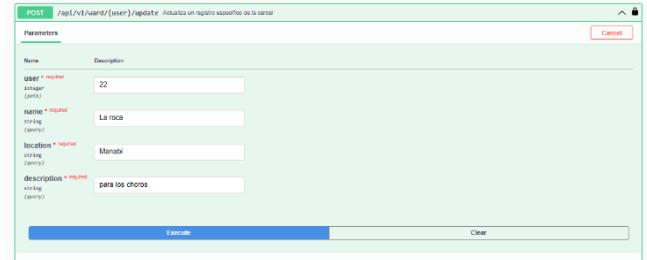


Fig.77. Swagger con la actualización.

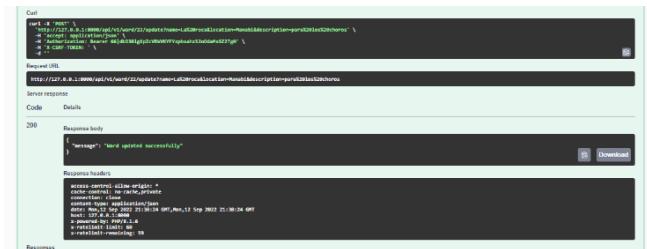


Fig.78. Cofirmacion de Swagger en actualizacion.

✓	Edit	Copy	Delete	19 Turcote Branch - Stokes Gardens
				1 Asperitum non necessitatibus nisi et aut
				2022-08-14 20:41:16 2022-08-14 20:41:16
✓	Edit	Copy	Delete	20 Labadie Junctions Luevillitz Isle
				1 Quam repellendus voluptatem totam
				2022-08-14 20:41:17 2022-08-14 20:41:17
✓	Edit	Copy	Delete	21 BFE Patria Latacunga
				0 Comandos
				2022-09-10 20:02:09 2022-09-10 20:02:09
✓	Edit	Copy	Delete	22 La roca Manabi
				1 para los choros
				2022-09-12 21:35:31 2022-09-12 21:38:24

Fig.79. Base de datos de actualizacion

Ahora crearemos un comentario para dar de baja una cárcel por medio de la función destroy y estas instrucciones se generará por medio de un comentario Swagger como se visualiza en la Fig.80 y Fig81.

```

    /**
     *  * @OA\Get(
     *      path="/api/v1/ward/{user}/destroy",
     *      tags={"OpcionesWard"},
     *      summary="Deshabilita un registro específico",
     *      operationId="ward-user-destroy",
     *      security={
     *          {"passport": {}},
     *      },
     *      @OA\Parameter(
     *          name="user",
     *          in="path",
     *          required=true,
     *          @OA\Schema(
     *              type="integer"
     *          )
     *      ),
     *      @OA\Response(
     *          response=200,
     *          description="Success",
     *          @OA\MediaType(
     *              mediaType="application/json",
     *          )
     *      ),
     *      @OA\Response(
     *          response=401,
     *          description="Unauthenticated"
     *      ),
     *      @OA\Response(
     *          response=400,
     *          description="Bad Request"
     *      ),
     *      @OA\Response(
     *          description="Bad Request"
     *      ),
     *      @OA\Response(
     *          response=404,
     *          description="not found"
     *      ),
     *      @OA\Response(
     *          response=403,
     *          description="Forbidden"
     *      )
     *  )
     */

```

Fig.78. Parámetros deshabilitar

```

    *     description="not found"
    *   ),
    *   @OA\Response(
    *     response=403,
    *     description="Forbidden"
    *   )
    */
    // Dar de baja a un pabellon
    public function destroy(Ward $ward)
    {
        // Obtener el estado del pabellon
        $ward_state = $ward->state;

        // Almacenar un string con el mensaje
        $message = $ward_state ? 'inactivated' : 'activated';

        // Verifica que el pabellon tiene guardias
        if ($ward->users->isNotEmpty())
        {
            return $this->sendResponse(message: 'This ward has assigned guard(s)', code: 403);
        }
        // Cambia el estado del pabellon
        $ward->state = !$ward_state;

        // Guardar en la BDD
        $ward->save();

        // Invoca el controlador padre para la respuesta json

```

Fig.78. Función deshabilitar

Vamos a crear un comentario Swagger en la función index para visualizar los reportes del usuario guardia como se visualiza en la Fig.79 y Fig.80.

```

22
23     // Métodos del Controlador
24     // Listar los reportes
25 /**
26  *  * @OA\Get(
27  *  *  path="/api/v1/report/",
28  *  *  tags={"OpcionesReport"},
29  *  *  summary="Listar todos los reportes del guardia existente",
30  *  *  operationId="report",
31  *  *  security={
32  *  *      {"passport": {}},
33  *  *  },
34  *  *
35  *  * @OA\Response(
36  *  *  response=200,
37  *  *  description="Success",
38  *  *  @OA\MediaType(
39  *  *      mediaType="application/json",
40  *  *  )
41  *  ),
42  *  * @OA\Response(
43  *  *  response=401,
44  *  *  description="Unauthenticated"
45  *  ),
46  *  * @OA\Response(
47  *  *  response=400,
48  *  *  description="Bad Request"
49  *  ),
50  *  * @OA\Response(
51  *  *  response=404,
52  *  *  description="not found"
53  *  )

```

Fig.79. Parámetros listar

```

46  *  * @OA\Response(
47  *  *  response=400,
48  *  *  description="Bad Request"
49  *  ),
50  *  * @OA\Response(
51  *  *  response=404,
52  *  *  description="not found"
53  *  ),
54  *  * @OA\Response(
55  *  *  response=403,
56  *  *  description="Forbidden"
57  *  )
58 */
59
60
61
62
63
64 public function index()
65 {
66     // Se obtiene el usuario autenticado
67     $user = Auth::user();
68     // Del usuario se obtiene los reportes
69     $reports = $user->reports;
70     // Invoca el controlador padre para la respuesta json
71     // El moldeo de la información por el Resource
72     return $this->sendResponse(message: 'Report list generated successfully',
73     | 'reports' => ReportResource::collection($reports)
74 );

```

Fig.80. Función index

Podemos visualizar su funcionamiento de jaulas con la siguiente Fig. 81.

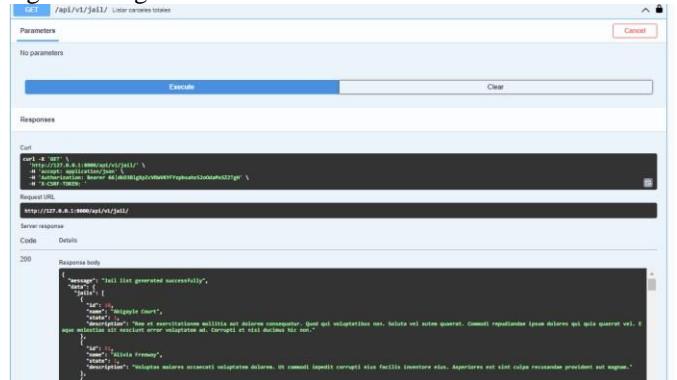


Fig.81. Ejecución Swagger

Vamos a crear en la función store los comentarios Swagger para poder instruir a la aplicación los parámetros de crear un nuevo reporte como se visualiza en la Fig. 82 y Fig. 83.

```

    /**
     * @OA\Post(
     * path="/api/v1/report/create",
     * tags={"OpcionesReport"},
     * summary="Crear un nuevo reporte del guardia",
     * operationId="report-create",
     * security={
     * {"passport": {}},
     * },
     * @OA\Parameter(
     * name="title",
     * in="query",
     * required=true,
     * @OA\Schema(
     * type="string"
     * )
     * ),
     * @OA\Parameter(
     * name="description",
     * in="query",
     * required=true,
     * @OA\Schema(
     * type="string"
     * )
     * ),
     * @OA\RequestBody(
     * @OA\JsonContent(),
     * @OA\MediaType(
     * mediaType="multipart/form-data",
     * @OA\Schema(

```

Fig.82. Parámetros Report

```

        * ),
        * @OA\Response(
        * response=401,
        * description="Unauthenticated"
        * ),
        * @OA\Response(
        * response=400,
        * description="Bad Request"
        * ),
        * @OA\Response(
        * response=404,
        * description="not found"
        * ),
        * @OA\Response(
        * response=403,
        * description="Forbidden"
        * )
    */

public function store(Request $request)
{
    // Validación de los datos de entrada
    $request->validate([
        'title' => ['required', 'string', 'min:5', 'max:45'],
        'description' => ['required', 'string', 'min:5', 'max:255'],
        'image' => ['nullable', 'image', 'mimes:jpg,png,jpeg', 'max:512'], //max image size is 512 kb
    ]);

    // Del request se obtiene únicamente los dos campos
    $report_data = $request->only(['title', 'description']);
    // Se crea una nueva instancia (en memoria)
}

```

Fig.83. Función store

Se puede visualizar la creación de una jaula en la Fig.84 y Fig.85.

The screenshot shows the 'Create' form for a cage. The fields are as follows:

- Name:** Alfa
- Code:** 000FD04
- Type:** high
- Capacity:** 4
- Ward ID:** 21
- Description:** Para los lobos

Fig.84. Función create swagger

The screenshot shows the execution results for the 'Create' operation. It includes:

- Curl:** A command-line interface for sending HTTP requests.
- Request URL:** <http://127.0.0.1:1000/api/v1/jail/create>
- Server response:** A detailed JSON response including headers and body.
- Response body:** A JSON object with the message "jail stored successfully".

Fig.85. Ejecución create swagger

En la función show se creará un comentario Swagger para poder visualizar reportes específicos que quiere ver el usuario como se visualiza en la Fig.85 y Fig.86.

```

    // Mostrar la información del reporte
    /**
     * @OA\Get(
     * path="/api/v1/report/{report}",
     * tags={"OpcionesReport"},
     * summary="Visualiza el contenido de un registro específico que quiere ver",
     * operationId="report-show",
     * security={
     * {"passport": {}},
     * },
     * @OA\Parameter(
     * name="report",
     * in="path",
     * required=true,
     * @OA\Schema(
     * type="integer"
     * )
     * ),
     *
     * @OA\Response(
     * response=200,
     * description="Success",
     * @OA\MediaType(
     * mediaType="application/json",
     * )
     * ),
     * @OA\Response(
     * response=401,
     * description="Unauthenticated"
     * )
    */

```

Fig.86. Parámetros show report

```

    * response=401,
    * description="Unauthenticated"
    * ),
    * @OA\Response(
    * response=400,
    * description="Bad Request"
    * ),
    * @OA\Response(
    * response=404,
    * description="not found"
    * ),
    * @OA\Response(
    * response=403,
    * description="Forbidden"
    * )
    */

public function show(Report $report)
{
    // Invoca el controlador padre para la respuesta json
    // El moldeo de la información por el Resource
    return $this->sendResponse(message: 'Report details', result: [
        'report' => new ReportResource($report),
    ]);
}

```

Fig.87. Parámetros show report

Se puede visualizar el funcionamiento de este Swagger en la Fig.88.

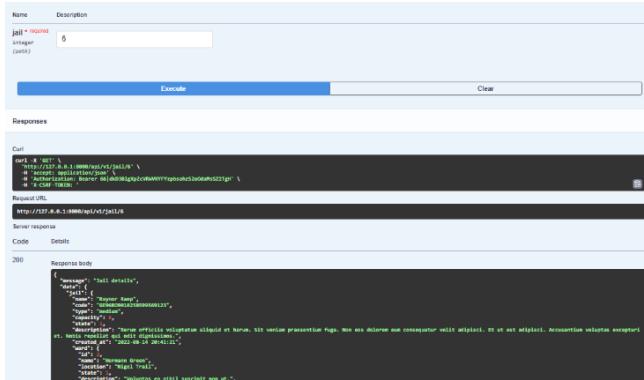


Fig.88. Ejecución show swagger

Se creará en la función update un comentario Swagger para poder actualizar los datos del reporte que el usuario desee modificar como se visualiza en la Fig.88, Fig.89 y Fig.90.

```

103 // Actualizar la información del reporte
104
105 /**
106  * @OA\Post(
107  *   path="/api/v1/report/{report}/update",
108  *   tags={"OpcionesReport"},
109  *   summary="Actualiza un registro específico proveniente del guardia",
110  *   operationId="report-update",
111  * security={
112  *     {"passport": {}},
113  *   },
114  *   @OA\Parameter(
115  *     name="report",
116  *     in="path",
117  *     required=true,
118  *     @OA\Schema(
119  *       type="integer"
120  *     )
121  *   ),
122  *   @OA\Parameter(
123  *     name="title",
124  *     in="query",
125  *     required=true,
126  *     @OA\Schema(
127  *       type="string"
128  *     )
129  *   ),
130  *   @OA\Parameter(
131  *     name="description",
132  *     in="query",
133  *     required=true,
134  *   )
135  */

```

Fig.89. Parámetros update report

```

163   *   ),
164   *   @OA\Parameter(
165   *     name="description",
166   *     in="query",
167   *     required=true,
168   *     @OA\Schema(
169   *       type="string"
170   *     )
171   *   ),
172   *   @OA\RequestBody(
173   *     @OA\JsonContent(),
174   *     @OA\MediaType(
175   *       mediaType="multipart/form-data",
176   *       @OA\Schema(
177   *         type="object",
178   *         required={"image"},
179   *         @OA\Property(property="image", type="file"),
180   *       ),
181   *     ),
182   *   ),
183   *   @OA\Response(
184   *     response=200,
185   *     description="Actualizado"
186   *   ),
187   *   @OA\Response(
188   *     response=401,
189   *     description="Unauthenticated"
190   *   ),
191   *   @OA\Response(

```

Fig.90. Parámetros update report

```

192   *   ),
193   *   @OA\Response(
194   *     response=400,
195   *     description="Bad Request"
196   *   ),
197   *   @OA\Response(
198   *     response=404,
199   *     description="not found"
200   *   ),
201   *   @OA\Response(
202   *     response=403,
203   *     description="Forbidden"
204   *   )
205   */
206
207
208 public function update(Request $request, Report $report)
209 {
210   // Validación de los datos de entrada
211   $request->validate([
212     'title' => ['required', 'string', 'min:5', 'max:45'],
213     'description' => ['required', 'string', 'min:5', 'max:255'],
214     'image' => ['nullable', 'image', 'mimes:jpg,png,jpeg', 'max:512'],
215   ]);
216
217   // Del request se obtiene únicamente los dos campos
218   $report_data = $request->only(['title', 'description']);
219   // Actualiza los datos del reporte
220   $report->fill($report_data)->save();
221 }

```

Fig.91. Función Update

Se puede evidenciar su funcionamiento en las Fig.92 y Fig.93.

The screenshot shows the Swagger UI interface for the 'Update' endpoint. It has a 'Parameters' section with the following fields:

- jail * required integer (path) value: 62
- name * required string (query) value: Alia A
- code * required string (query) value: DSFSDF4
- type * required string (query) value: low
- capacity * required string (query) value: 2
- ward_id * required string (query) value: 5
- description * required string (query) value: para los chortos

Fig.92. Función Update swagger

	Edit	Copy	Delete	99	Clevis Highway	AT27034555639277766	high	0	1	20	laborum. Accusamus amet item.	2022-08-16 20:41:40	2022-08-14 20
				60	Montana Freeway	LB9653934513J0J1R55FE4E5340MS	low	6	1	20	quiabundam dolent nisi rem consequatur Sunt	2022-08-14 20:41:40	2022-08-10 19:21:02
				61	Latacunga	GGDFGFDG34	high	5	1	1	para los chortos	2022-09-10 19:26:54	2022-09-10 19:21:02
				62	Alia A	DSFSDF4	low	2	1	5	para los chortos	2022-09-12 23:54:54	2022-09-12 23:54:54

Fig.93. Alwaysdata

Se va a dar de baja un reporte por medio de los comentarios Swagger y la función destroy como se visualiza en la Fig.94 y Fig.95.

```

34
35     // Dar de baja a un pabellon
36     /**
37      * @OA\Get(
38      * path="/api/v1/report/{report}/destroy",
39      * tags={"OpcionesReport"},
40      * summary="Da de baja un reporte del guardia",
41      * operationId="report-destroy",
42      * security={
43      * {"passport": {}},
44      * },
45      *
46      * @OA\Parameter(
47      * name="report",
48      * in="path",
49      * required=true,
50      * @OA\Schema(
51      * type="integer"
52      * )
53      * ),
54      *
55      *
56      * @OA\Response(
57      * response=200,
58      * description="Success",
59      * @OA\MediaType(
60      * mediaType="application/json",
61      * )
62      * ),
63      *
64    )

```

Fig.94. Parámetro destroy

```

164     * @OA\Response(
165     *      response=401,
166     *      description="Unauthenticated"
167     * ),
168     * @OA\Response(
169     *      response=400,
170     *      description="Bad Request"
171     * ),
172     * @OA\Response(
173     *      response=404,
174     *      description="not found"
175     * ),
176     * @OA\Response(
177     *      response=403,
178     *      description="Forbidden"
179     * )
180   */
181
182
183   public function destroy(Report $report)
184   {
185     // Obtener el estado del reporte
186     $report_state = $report->state;
187     // Almacenar un string con el mensaje
188     $message = $report_state ? 'inactivated' : 'activated';
189     // Cambia el estado del pabellon
190     $report->state = !$report_state;
191     // Guardar en la BD
192     $report->save();
193     // Invoca el controlador padre para la respuesta json
194     return $this->sendResponse($message, "Report $message successfully");
195   }

```

Fig.95. Función destroy

Se puede evidenciar la inactividad de la jaula en la Fig.96.

The screenshot shows the Swagger UI interface. In the 'Responses' section, there is a successful response (200) with the message: "{'message': 'Jaula inactivated successfully'}". Below it, the 'Server response' section shows the raw JSON response: {"message": "Jaula inactivated successfully"}.

Fig.97. Ejecución en swagger

Ahora en el controlador GuardtoWardController podremos crear un comentario de Swagger donde especificaremos los parámetros de listar todas las asignaciones realizadas en la función index como se visualiza en la Fig.98.

```

22     * $this->middleware('verify.ward.assignment')->only('assign');
23   }
24   /**
25    * @OA\Get(
26    * path="/api/v1/assignment/guards-and-wards",
27    * tags={"GuardiaPabellon"},
28    * summary="Listar los guardias asignados a pabellones",
29    * operationId="assignment(guards-wards)",
30    * security={
31    * {"passport": {}},
32    * },
33    *
34    * @OA\Response(
35    * response=200,
36    * description="Success",
37    * @OA\MediaType(
38    * mediaType="application/json",
39    * )
40    * ),
41    * @OA\Response(
42    * response=401,
43    * description="Unauthenticated"
44    * ),
45    * @OA\Response(
46    * response=400,
47    * description="Bad Request"
48    * ),
49    * @OA\Response(
50    * response=404,
51    * description="not found"
52    * ),
53    * @OA\Response(
54    * response=403

```

Fig.98. Parámetros listar

Se podrá visualizar su funcionamiento en Swagger como se ve en la Fig.99.

The screenshot shows the Swagger UI for the 'assignment(guards-wards)' endpoint. It includes sections for 'Parameters' (none), 'Responses' (200 OK), and 'Code'. The 'Code' section shows a successful response with status 200, message 'Assignment between Guards and wards generated successfully', and a JSON body containing a list of assignments. One assignment is highlighted with a blue border: {"id": 1, "guard": "Julian", "ward": "Nataly Rosales", "url": "http://127.0.0.1:8000/api/v1/assignment/guards-and-wards/1", "date": "Mon, 12 Sep 2022 23:51:48 GMT", "date2": "Mon, 12 Sep 2022 23:51:48 GMT", "status": "active", "created_at": "2022-09-12T23:51:48Z", "updated_at": "2022-09-12T23:51:48Z", "deleted_at": null}.

Fig.99. Ejecución swagger

Con el siguiente método se podrá asignar los pabellones con sus guardias como se visualiza en la Fig.100 y Fig.101 con los comentarios Swagger de la función assign.

```

    *      response=400,
    *      description="Bad Request"
    * ),
    * @OA\Response(
    *      response=404,
    *      description="not found"
    * ),
    * @OA\Response(
    *      response=403,
    *      description="Forbidden"
    * )
    * )
// Método para realizar la asignación respectiva
public function assign(User $user, Ward $ward)
{
    // Obtener solo los id de los pabellones de los usuarios
    // https://laravel.com/docs/9.x/eloquent-collections#method-modelKeys
    $guards_wards_id = $user->wards->modelKeys();

    // https://laravel.com/docs/9.x/eloquent-relationships#syncing-associations
    $user->wards()->syncWithPivotValues($guards_wards_id, ['state' => false]);
    $user->wards()->sync([$ward->id]);

    // Invoca el controlador padre para la respuesta json
    // El moldeo de la información por el Resource
    return $this->sendResponse(message: 'Assignment updated successfully', result: [
        'user' => new UserResource($user),
        'ward' => $ward
    ]);
}

```

Fig.100. Función assign

Podemos ver su funcionalidad en Swagger de la siguiente Fig.101.

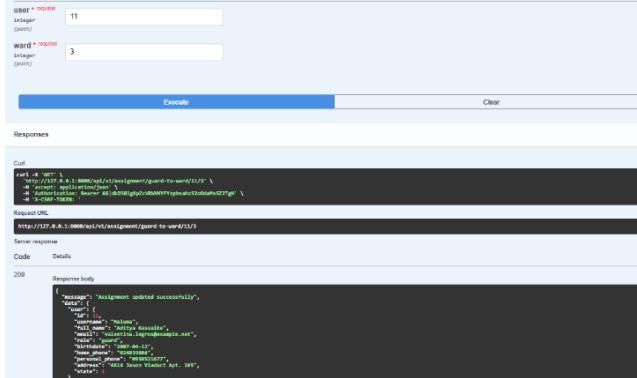


Fig.101. Ejecución swagger

Ahora crearemos en JailandPrisoner Controller un comentario Swagger de la función index donde se enlistará los presos en sus jaulas como se visualiza en la Fig.102.

```

/**
 *  * @OA\Get(
 *      path="/api/v1/assignment/prisoners-and-jails",
 *      tags={"PrisioneroJaula"},
 *      summary="Listar los presos para su jaula",
 *      operationId="assignment(priso-jaula)",
 *      security={
 *          {"passport": {}},
 *      },
 *      @OA\Response(
 *          response=200,
 *          description="Success",
 *          @OA\MediaType(
 *              mediaType="application/json",
 *          )
 *      ),
 *      @OA\Response(
 *          response=401,
 *          description="Unauthenticated"
 *      ),
 *      @OA\Response(
 *          response=400,
 *          description="Bad Request"
 *      ),
 *      @OA\Response(
 *          response=404,
 *          description="not found"
 *      ),
 *      @OA\Response(
 *          response=403,
 *          description="not found"
 *      ),
 *      @OA\Response(
 *          response=403,
 *          description="Forbidden"
 *      )
 * )
// Métodos del Controlador
// Listar los prisioneros y cárceles disponibles
public function index()
{
    // Traer el rol prisionero
    $role = Role::where('slug', 'prisoner')->first();

    // Traer los usuarios a partir de ese rol
    $prisoners = $role->users();

    // https://laravel.com/docs/9.x/eloquent#cursors
    // https://laravel.com/docs/9.x/collections#method-filter
    // Por cada cárcel que se va a iterar
    $jails = Jail::cursor()->filter(function ($jail)
    {
        // Se obtiene el número de usuarios (prisioneros)
        $total_users = $jail->users->count();
    });
}

```

Fig.102. Función listar

Ahora se podrá visualizar en la Fig.103 por medio de la interfaz Swagger de la Fig.104.

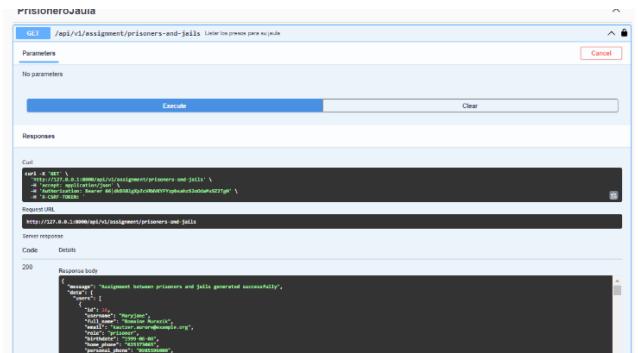


Fig.103. Ejecución swagger

En la función assign de Jail se podrá realizar un comentario Swagger para visualizar el jail con su respectivo preso como se visualiza en la Fig.104.

```
* * @OA\Get(
*   response=404,
*   description="not found"
* ),
*   @OA\Response(
*     response=403,
*     description="Forbidden"
*   )
*)
*/
// Método para realizar la asignación respectiva
public function assign(User $user, Jail $jail)
{
    // Obtener solo los id de las cárceles de los usuarios
    // https://laravel.com/docs/9.x/eloquent-collections#method-modelKeys
    $prisoner_jails_id = $user->jails->modelKeys();

    // https://laravel.com/docs/9.x/eloquent-relationships#syncing-associations
    $user->jails()->syncWithPivotValues($prisoner_jails_id, ['state' => false]);

    $user->jails()->sync([$jail->id]);

    // Invoca el controlador padre para la respuesta json
    // El molde de la información por el Resource
    return $this->sendResponse(message: 'Assignment updated successfully', result: [
        'user' => new UserResource($user),
        'jail' => $jail
    ]);
}

* -----
*   response=200,
*   description="Success",
*   @OA\MediaType(
*     mediaType="application/json",
*   ),
*   @OA\Response(
```

Fig.104. Fucion assign

Finalmente, se podrá realizar la verificación en la interfaz Swagger con los siguientes resultados que se logra visualizar en la Fig.105.

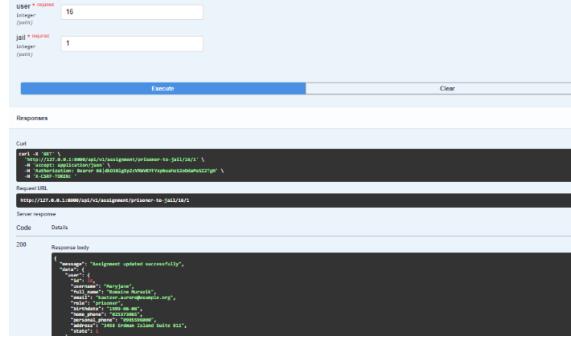


Fig.105. Ejecucion swagger

C. Sprint de frontend.

1. Recuperación de contraseña.

Para la realización de recuperación de contraseña lo primero que se hizo es crear un archivo jsx. Llamado Forgot_password y reset password se trabaja con el API del backend para verificar el usuario, después el usuario ingresa su correo como muestra la Fig.106 .



Fig.106. Ingreso de correo

Mediante Mailtrap le va a enviar un correo al usuario como muestra la Fig.107 y debe dar clic en el botón de reset el cual os envía a otra ventana donde se debeingresar los datos y actualizar la contraseña.

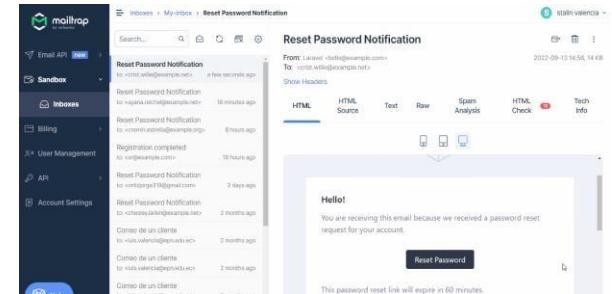


Fig.107. Correo en Mailtrap

Después se pasa a la sección de restablecer contraseña donde se debe ingresar el token enviado a su correo y con ello se puede realizar el restablecimiento de la contraseña Como se ve en la imagen Fig.108

Fig.108. Correo en Mailtrap

2. CRUD de pabellones y celda.

Para realizar la parte del componente front-end se utilizó una plantilla diferente en la parte práctica se observa lo siguiente.

En las actividades realizadas se encuentra el restablecimiento por correo de la cuenta y el crud de pabellones y cárceles

En el perfil de director podemos observar todos los pabellones que existen, lo podemos ver en forma de tabla y además podemos realizar las partes importantes del crud como lo veremos a continuación:

Para realizar este crud primero debemos ingresarse como director y procedemos a observar la pantalla de director comose ve en la imagen Fig.109.



Fig.109. Dashboard director

CRUD pabellones

Una vez establecida la lógica podemos realizar funciones importantes de acuerdo con la función que fue establecida en el api, en este caso pasamos a verlo en la ventana como aparece en la siguiente imagen Fig.110.

Fig.110. Administrar Pabellones

- Crear

Para crear el pabellón nos dirigimos a la sección de crear el pabellón, ahí se nos solicitará el nombre la descripción y dirección como lo vemos en la imagen Fig.111 y lo podemos observar en la base de datos Fig.112.

Fig.111. Crear Pabellón

<input type="checkbox"/>	Edit	Copy	Delete	25	Hells	latacunga c13	1	Es una carcel muy peligroa	2022-09-13 10:45:20
--------------------------	------	------	--------	----	-------	---------------	---	----------------------------	---------------------

Fig.112. Base de datos

- Leer

Podemos observar el pabellón por dirigiéndonos en la parte de mostrar, en este caso observamos la creada anteriormente en la imagen Fig.113.

Fig.117. Comprobación

CRUD celda

Para realizar este crud lo haremos en el perfil de director y realizar el mismo crud, podemos observar cómo se ve por medio de list Fig.118.

#	Nombre de la celda	Capacidad	Modificar	Editar
1	Alta A			
2	Alta Freeway			
3	Axe Roads			
4	Berlin Terrace			
5	Boehm Groove			
6	Braun Overpass			
7	Coldesto Lake			

Fig.118. Crear cárceles

Información de pabellón

Nombre del Pabellón: Hells

Dirección: Iatacunga c13

Descripción: Es una carcel muy peligroa

Estado: Active

Fig.113. Información

- Actualizar

Lo siguiente será actualizar el pabellón ahora nos dirigimos a la sección de actualizar haremos algunos cambios como se lo ve en la imagen Fig.114

Administrador PABELLONES

El objetivo principal de los prisioneros. Es mantener distanciado las organizaciones.

LISTA DE PABELLONES + NUEVO PABELLON

EDITAR Pabellon

Información del Pabellón

Nombre del pabellón *	Dirección *
Hells	Quito c13
Descripción *	Es una carcel no muy peligroa
GUARDAR	

Fig.114. Actualizar

- Borrar

Para borrar una Pabellón lo que se realizara es desactivarla, esto se hará en base al estado que tiene cada una, en el pabellón

anterior creado vemos su estado activo Fig.115, ahora lo desactivaremos y quedará de la siguiente forma Fig.116.

11 Hells

Es una carcel no muy peligroa



Fig.115. Activar

11 Hells

Es una carcel no muy peligroa



Fig.116. Desactivar

Finalmente, procedemos observar en la que en la base de datos contienen todo lo que se realizado en este crud Fig.117.

Edit Copy Delete 25 Hells

Quito c13

0 Es una carcel no muy peligroa

Fig.119. Crear cárceles

- Leer

Procedemos a observar los datos de la cárcel por medio a la lógica realizada en el show jail como vemos en la imagen Fig.120

Fig.120. Crear cárceles

- Actualizar

La siguiente sección es actualizar y se lo puede hacer en la parte de editar y lo vemos en la imagen Fig.121

Fig.121. Actualiza

- Borrar

Para desactivarlo lo hacemos dirigiéndonos a su estado como lo vemos en la imagen Fig.122

Fig.122. Borrar

D. Despliegue a producción.

VI. ENLACE DE HEROKU BACKEND

<https://apiterminado.herokuapp.com/>

<https://apiterminado.herokuapp.com/api/documentation>

VI. CONCLUSIÓN

El uso del BackEnd por parte de Laravel y uso del FrontEnd por parte React se ha combinado estos 2 componentes para tener una mejor visión en el desarrollo de la aplicación web donde se creó una API de servicios que será consumido por parte del programador mediante los algoritmos de interfaces gráficas donde cada elemento será indispensable en los ámbitos de gráfica y funcionalidad.

Se desarrolló esta aplicación con los métodos vistos de HTTP y el uso de librerías como React y el framework Laravel donde nos ayudara a crear un algoritmo limpio y preciso en el desarrollo de este proyecto donde abarcara temas anteriormente vistos de este semestre.

VII. REFERENCIAS

[1] B. Loarte, «Youtube,» 31 Agosto 2022. [En línea].

Available: <https://youtu.be/gR6f3Egk9WI>. [Último acceso: 07 09 2022].

VIII. BIBLIOGRAFÍA

Manuel Auqui, nació en Quito-Ecuador el 17 de febrero de 2002. Realizó sus estudios secundarios en la Unidad Educativa Técnica Fiscal “Arturo Borja”. Actualmente cursa sus estudios en Tecnología Superior en Desarrollo de Software en la Escuela Politécnica Nacional. Además, es miembro de la irrelevant organization del Club de los F.



Áreas de interés: Desarrollo Fronted y Backend, Organización y Liderazgo,

autosaboteo y música.

(manuel.auqui@epn.edu.ec)



(leoni.guambo@epn.edu.ec)

Leoni Guambo, nació en Quito-Ecuador el 1 de septiembre del 2001. Realizó sus estudios secundarios en el Colegio Don Bosco de la Kennedy. Actualmente estudia en la Escuela Politécnica Nacional en la carrera Tecnología Superior en Desarrollo de Software y está cursando el cuarto semestre. Áreas de interés: diseño web y de interfaces, soporte técnico y mantenimiento de equipos.



Mayerli Méndez, nació en Quito-Ecuador el 14 de febrero de 2002. Realizó sus estudios secundarios en la Unidad Educativa “Consejo Provincial de Pichincha”. Actualmente estudia en la Escuela Politécnica Nacional en la carrera Tecnología Superior en Desarrollo de Software y está cursando el cuarto semestre. Áreas de interés: diseño web y de interfaces soporte técnico y mantenimiento de equipos.

(mayerli.mendez@epn.edu.ec)



Jorge Ortiz, nació en Ambato-Ecuador el 23 de mayo del 2000. Realizó sus estudios secundarios en el Colegio Nuestra Señora del Rosario. Esta ejerciendo su carrera en Tecnología Superior en Desarrollo de Software en la Escuela Politécnica nacional desde el 2020 y etsa en el 4to nivel de dicha carrera.

Áreas de interés: programación, bases de datos, deportes, arquitectura y mantenimiento de computadores y ofimática.

(Jorge.ortiz@epn.edu.ec)



Luis Valencia, nació en Quito-Ecuador el 5 de marzo de 1998. Realizó sus estudios secundarios en el Colegio Nacional Vicente Rocafuerte. En la actualidad estudia en La Escuela Politécnica Nacional la carrera tecnológica en Desarrollo de Software en 4to semestre.

Áreas de interés: Desarrollo de páginas web, seguridad informática, testeo de programas, computación y las tecnologías de la información.

(Luis.valencia@epn.edu.ec)