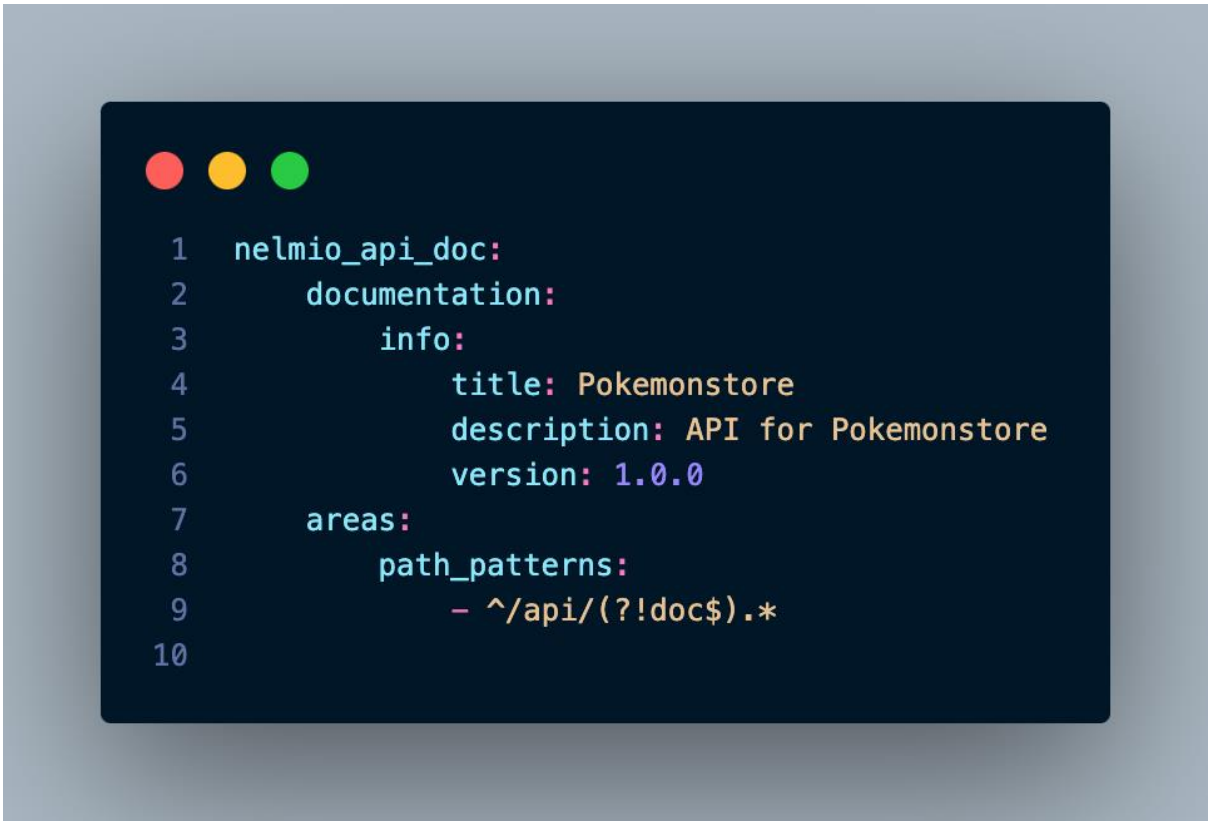
3.2.B04 - Automate generation of documentation

Pour montrer ma capacité à générer automatiquement de la documentation, j'ai utilisé un projet personnel sur lequel je suis en train de travailler. Ce projet a pour but de fournir un site de e-commerce sur le thème de Pokémon, avec la possibilité d'acheter tous types de produits que l'on pourrait retrouver dans une boutique du jeu.

Pour ce projet, j'ai une stack Symfony pour le back-end, Postgresql pour la gestion des données et Next.js pour la partie front-end, accompagné de Tailwind CSS pour le framework CSS.

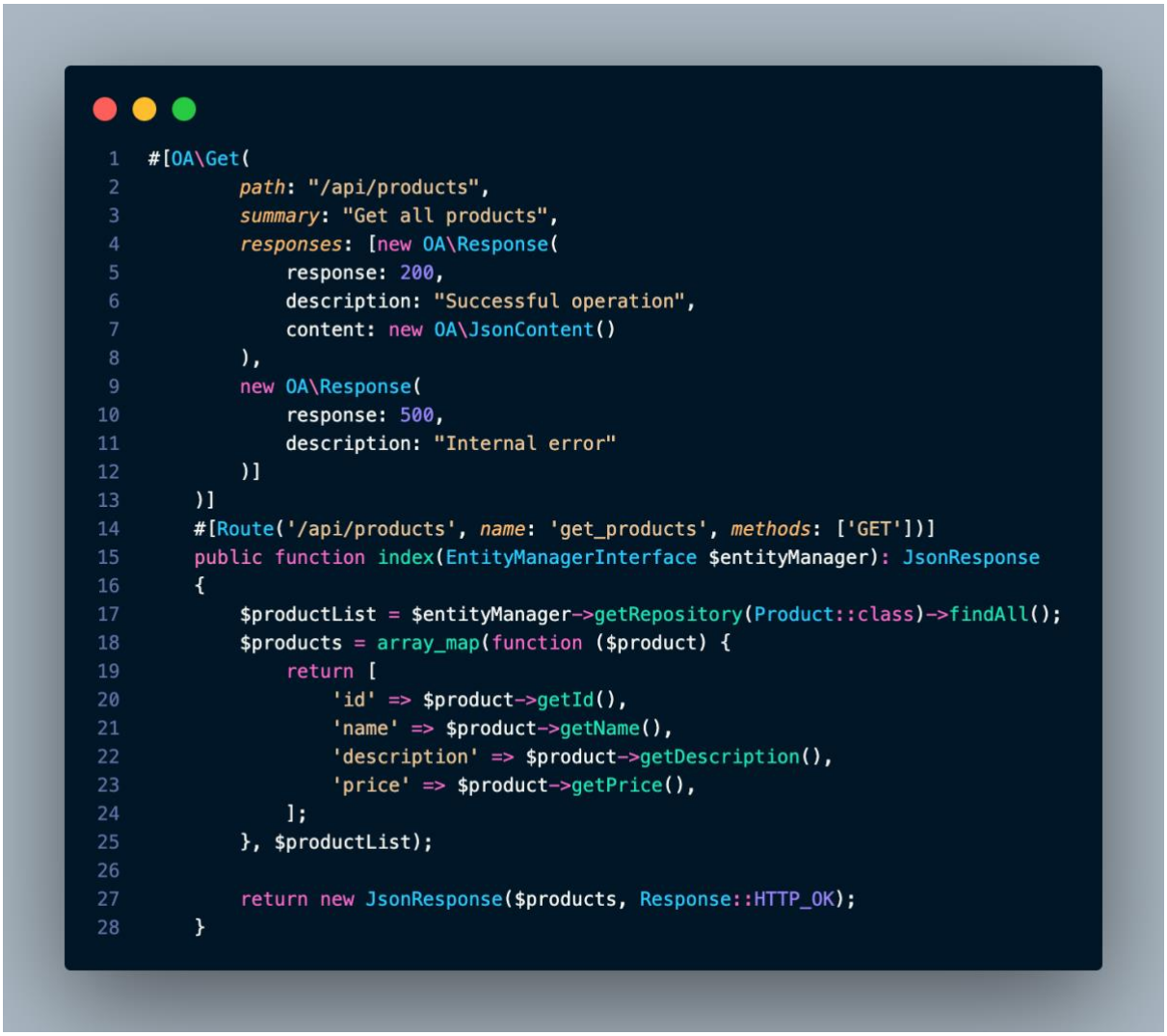
Pour générer la documentation de mon API Symfony, j'ai utilisé le bundle NelmioApiDocBundle qui permet de générer une documentation Swagger.



```yaml
nelmio_api_doc:
    documentation:
        info:
            title: Pokemonstore
            description: API for Pokemonstore
            version: 1.0.0
    areas:
        path_patterns:
            - ^/api/(?!doc$).*

```

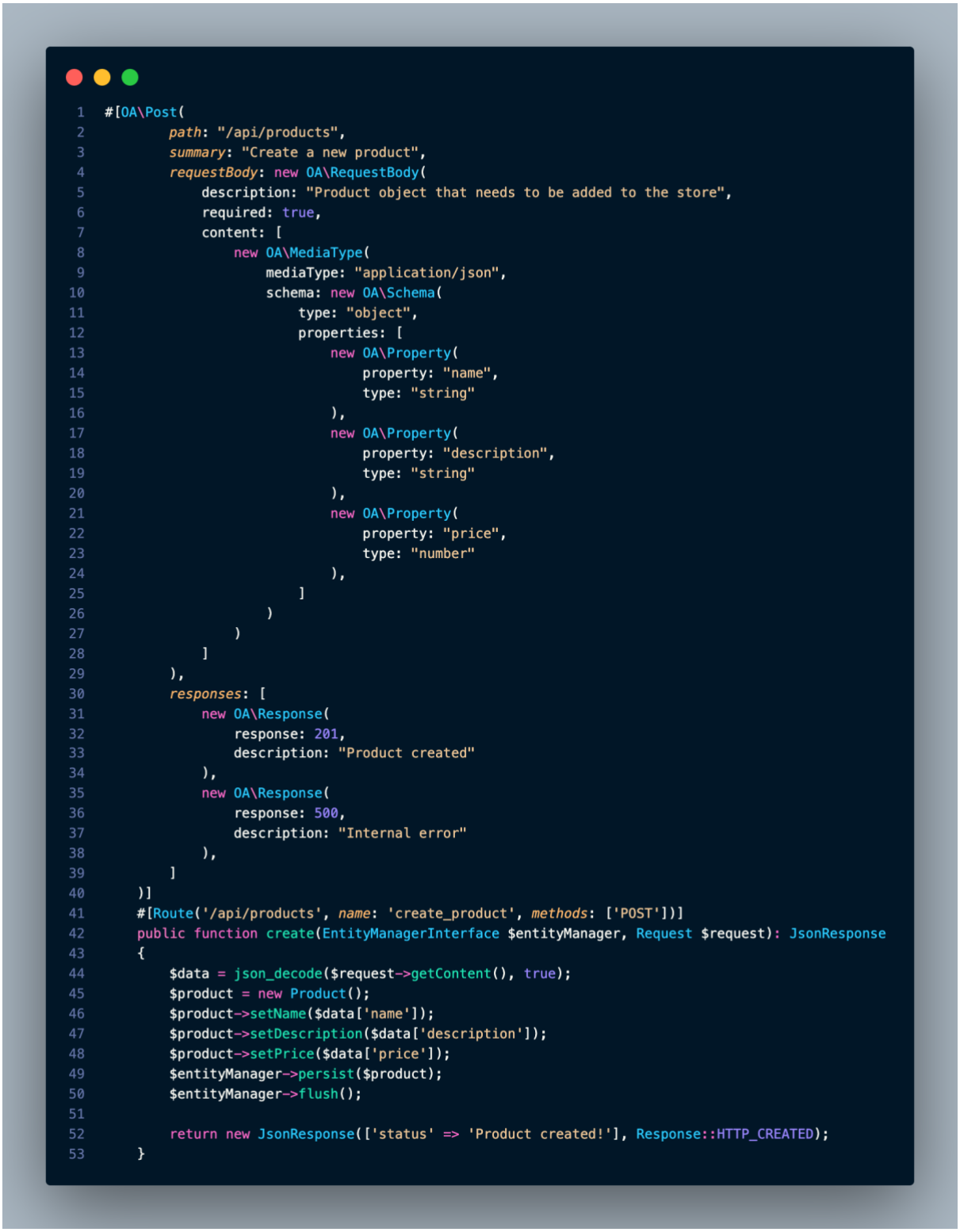
En ce qui concerne la configuration du bundle, j'ai changé la propriété path_patterns en utilisant du regex pour que la documentation se fasse sur toutes mes routes comprenant le terme "/api" tout en excluant la route "/api/doc". Cela me permet d'éviter d'avoir la documentation pour les routes automatiquement générées par Symfony (la route "/_error" par exemple) et pour la route GET qui affiche la documentation.



```php
#[OA\Get(
        path: "/api/products",
        summary: "Get all products",
        responses: [new OA\Response(
            response: 200,
            description: "Successful operation",
            content: new OA\JsonContent()
        ),
        new OA\Response(
            response: 500,
            description: "Internal error"
        )]
    )]
    #[Route('/api/products', name: 'get_products', methods: ['GET'])]
    public function index(EntityManagerInterface $entityManager): JsonResponse
    {
        $productList = $entityManager->getRepository(Product::class)->findAll();
        $products = array_map(function ($product) {
            return [
                'id' => $product->getId(),
                'name' => $product->getName(),
                'description' => $product->getDescription(),
                'price' => $product->getPrice(),
            ];
        }, $productList);

        return new JsonResponse($products, Response::HTTP_OK);
    }
```

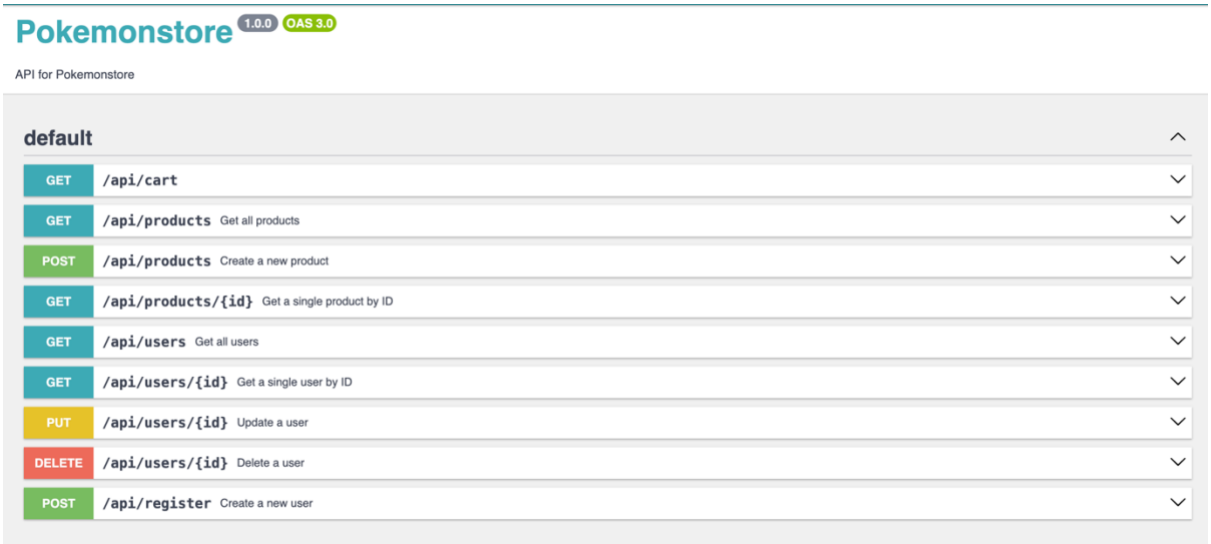
Pour générer la documentation de façon détaillée pour chacune des routes, le bundle utilise la syntaxe des attributs OpenAPI (Swagger). Pour cette route GET, je précise le path correspondant et les différents cas de réponses.



```php
#[OA\Post(
        path: "/api/products",
        summary: "Create a new product",
        requestBody: new OA\RequestBody(
            description: "Product object that needs to be added to the store",
            required: true,
            content: [
                new OA\MediaType(
                    mediaType: "application/json",
                    schema: new OA\Schema(
                        type: "object",
                        properties: [
                            new OA\Property(
                                property: "name",
                                type: "string"
                            ),
                            new OA\Property(
                                property: "description",
                                type: "string"
                            ),
                            new OA\Property(
                                property: "price",
                                type: "number"
                            ),
                        ]
                    )
                )
            ]
        ),
        responses: [
            new OA\Response(
                response: 201,
                description: "Product created"
            ),
            new OA\Response(
                response: 500,
                description: "Internal error"
            ),
        ]
    )]
    #[Route('/api/products', name: 'create_product', methods: ['POST'])]
    public function create(EntityManagerInterface $entityManager, Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true);
        $product = new Product();
        $product->setName($data['name']);
        $product->setDescription($data['description']);
        $product->setPrice($data['price']);
        $entityManager->persist($product);
        $entityManager->flush();

        return new JsonResponse(['status' => 'Product created!'], Response::HTTP_CREATED);
    }
```

Il en va de même pour la route POST, qui contient également les paramètres à ajouter dans le body de la requête HTTP pour permettre de créer un produit.



La documentation de mon API est donc disponible sur localhost:8000/api/doc et possède bien les différentes réponses et les requests bodies pour les requêtes POST et PUT.