WORK DONE

1. Data Retrieval:
   • Fetch data from the provided external dataset using PHP.
   • Caching: To optimize performance, implement a caching mechanism that stores the retrieved data locally (e.g., in a JSON file) for a defined period (e.g., 10 minutes), reducing the need for frequent external API calls.
   • Cache Invalidation: Ensure the cache is invalidated and refreshed if it exceeds the defined period (10 minutes), ensuring up-to-date data retrieval.

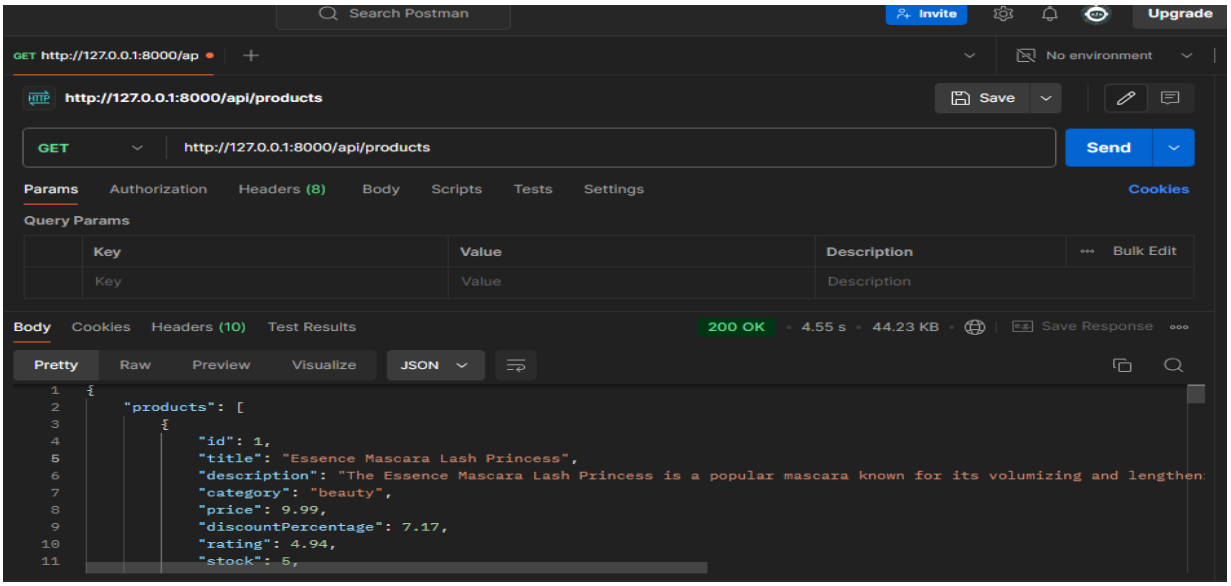
```php
// Defined cache duration (10 minutes)
1 reference
protected $cacheDuration = 600; // 10 minutes in seconds

// Fetch data from external API
7 references
private function fetchProducts(): mixed
{
    // Check if data is cached
    if (Cache::has(key: 'products')) {
        return Cache::get(key: 'products');
    }

    // Fetch data from external API if not cached
    //$response = Http::get('https://dummyjson.com/products');

    //disable the SSL
    $response = Http::withOptions(options: ['verify' => false])->get(url: 'https://dummyjson.com/products');

    // Handle failed requests
    if ($response->failed()) {
        return response()->json(data: ['error' => 'Failed to retrieve data from external API'], status: 500);
    }

    // Cache the data for 10 minutes
    $products = $response->json();
    Cache::put(key: 'products', value: $products, ttl: $this->cacheDuration);

    return $products;
}
```

2. API Endpoints: Develop the following API endpoints to interact with the retrieved data:

   i. List all products: Retrieve and return the full list of products.

```php
// Show the list of all products
1 reference | 0 overrides
public function list(): mixed
{
    $products = $this->fetchProducts();

    // Check for any API fetch errors
    if (!is_array(value: $products)) {
        // Return error response from fetchProducts
        return $products;
    }

    return response()->json(data: $products);      edo2019, 2 hours ago • Created laravel project wit
}
```

OUTPUT



ii. Search products by name: Enable users to search for products by keywords in the product name. Ensure the search functionality is case-insensitive and supports partial matches.

```php
// Search products by name (case-insensitive, partial match)
1 reference | 0 overrides
public function search(Request $request): JsonResponse|mixed
{
    $request->validate(rules: [
        'name' => 'required|string|min:1',
    ]);

    $products = collect(value: $this->fetchProducts()['products']);
    $keyword = strtolower(string: $request->input(key: 'name'));

    $filtered = $products->filter(callback: function (TValue $product) use ($keyword): bool {
        return strpos(haystack: strtolower(string: $product['title']), needle: $keyword) !== false;
    });

    return response()->json(data: $filtered->values(), status: 200);
}
```

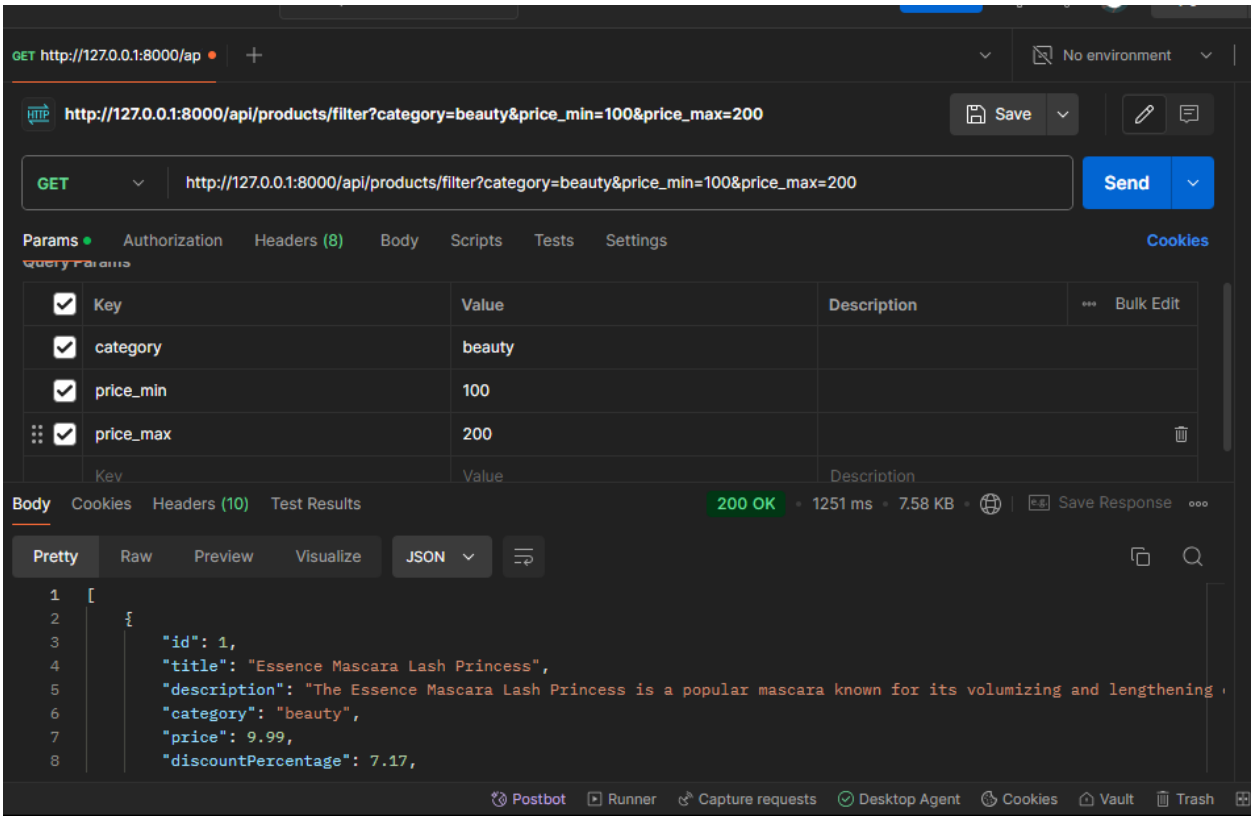
OUTPUT

```
> App\Models\Product::where('title','like','Apple')->get();
= Illuminate\Database\Eloquent\Collection {#6161
    all: [
      App\Models\Product {#6159
        id: 16,
        title: "Apple",
        price: "1.99",
        description: "Fresh and crisp apples, perfect for snacking or incorporating into various recipes.",
        category: "groceries",
        created_at: "2024-09-27 06:19:03",
        updated_at: "2024-09-27 06:19:03",
      },
    ],
  }
```

iii. Filter products by category and price range: Allow users to filter products by category and specify a price range (e.g., minimum and maximum prices) within the selected category.

```
// Filter products by category and price range
1 reference | 0 overrides
public function filter(Request $request): JsonResponse|mixed
{
    $request->validate(rules: [
        'category' => 'required|string|min:1',
        'min_price' => 'numeric|min:0',
        'max_price' => 'numeric|min:0|gte:min_price',
    ]);

    $products = collect(value: $this->fetchProducts()['products']);
    $category = strtolower(string: $request->input(key: 'category'));
    $minPrice = $request->input(key: 'min_price', default: 0);
    $maxPrice = $request->input(key: 'max_price', default: PHP_INT_MAX);

    $filtered = $products->where(key: 'category', operator: $category)
        ->whereBetween(key: 'price', values: [$minPrice, $maxPrice]);

    return response()->json(data: $filtered->values(), status: 200);
}
```

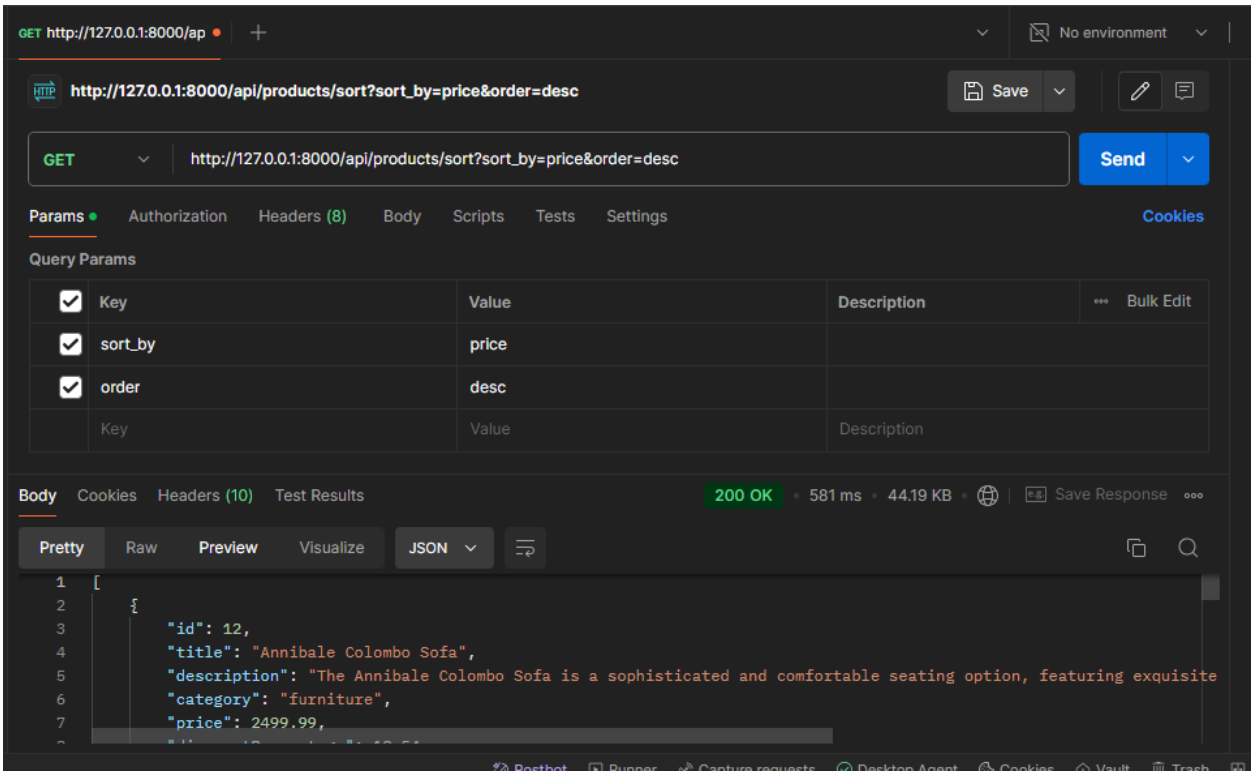
OUTPUT



iv. Sort products: Enable sorting of products by various fields (e.g., price, title) in ascending or descending order.

```php
// Sort products by various fields (ascending or descending)
1 reference | 0 overrides
public function sort(Request $request): JsonResponse|mixed
{
    $request->validate(rules: [
        'sort_by' => 'required|in:price,title',
        'order' => 'required|in:asc,desc',
    ]);

    $products = collect(value: $this->fetchProducts()['products']);
    $sortBy = $request->input(key: 'sort_by', default: 'price');
    $order = $request->input(key: 'order', default: 'asc');

    $sorted = $products->sortBy(callback: $sortBy, options: SORT_REGULAR, descending: $order == 'desc');

    return response()->json(data: $sorted->values(), status: 200);
}
```

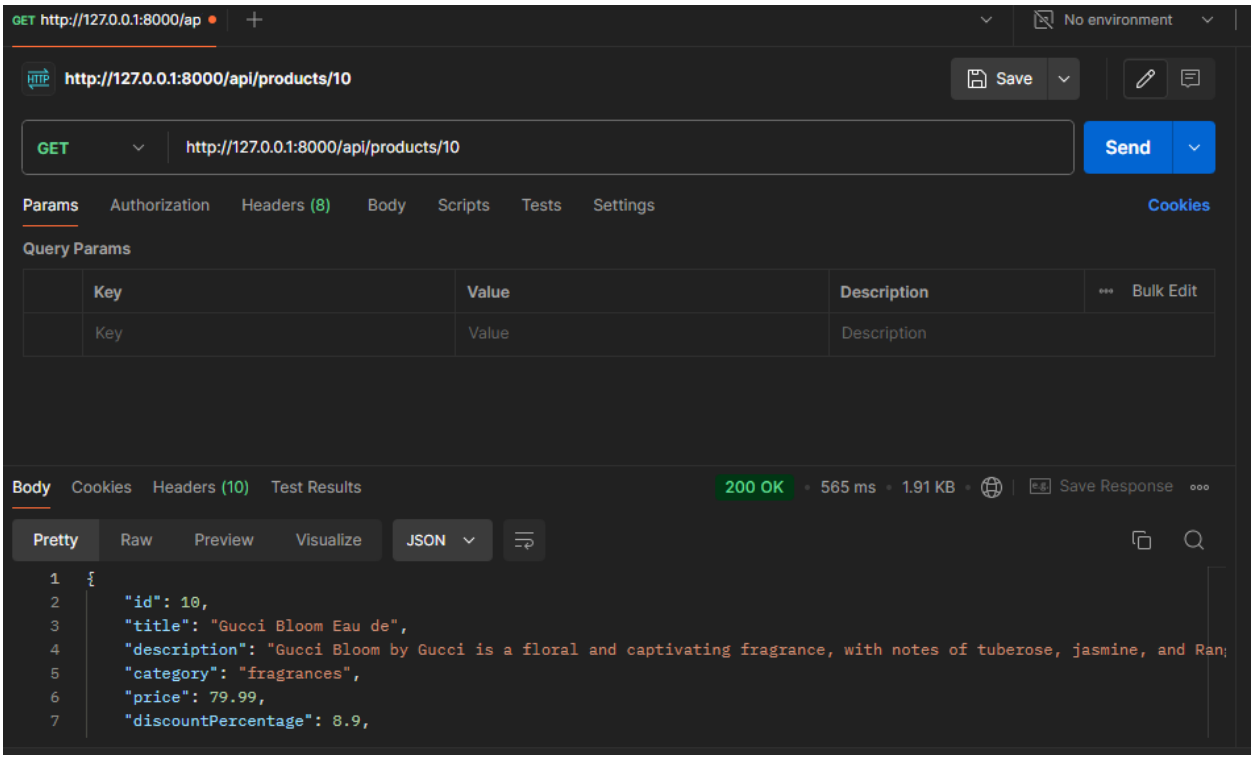
OUTPUT



V. Get product details by ID: Return detailed product information when a valid product ID is provided.

```
// Get product details by ID
1 reference | 0 overrides
public function show($id): JsonResponse|mixed
{
    $products = collect(value: $this->fetchProducts()['products']);
    $product = $products->where(key: 'id', operator: $id)->first();

    if (!$product) {
        return response()->json(data: ['error' => 'Product not found'], status: 404);
    }

    return response()->json(data: $product, status: 200);
}
```

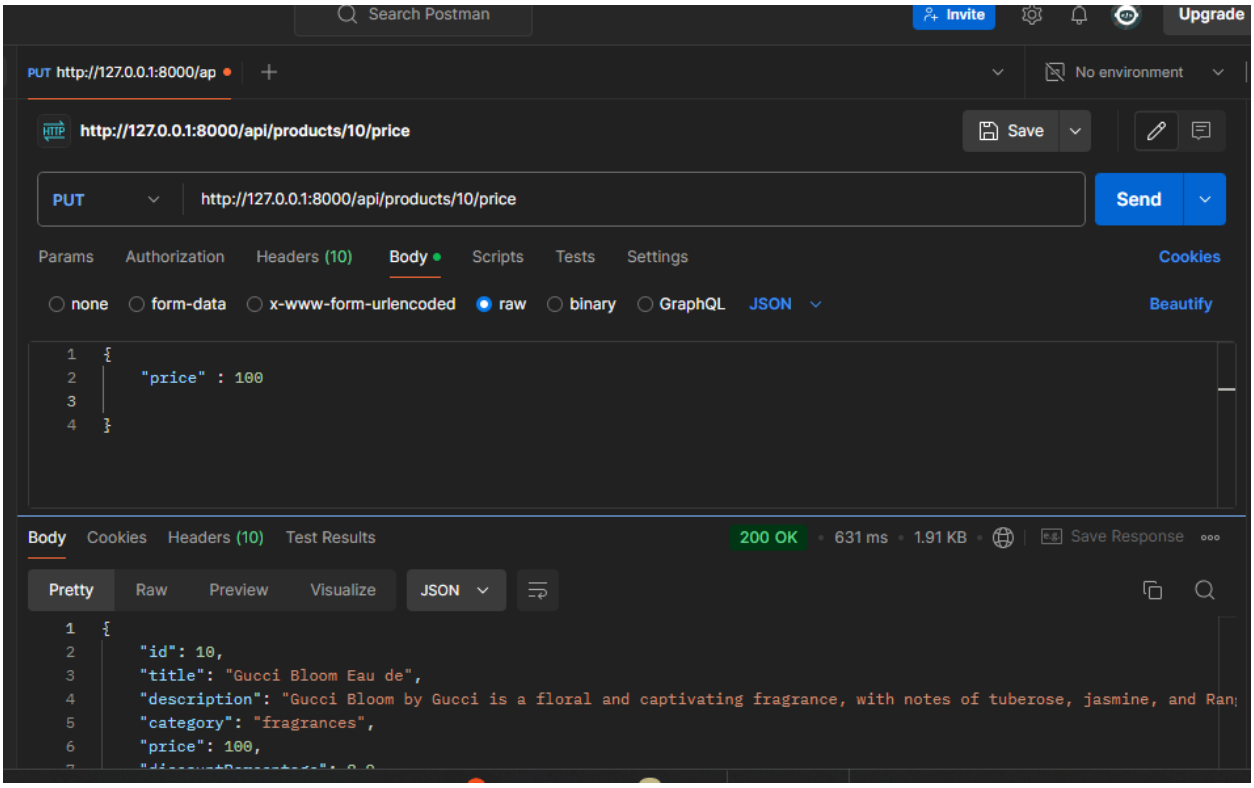
OUTPUT



vi. Update product price: Allow users to update the price of a specific product locally (without modifying the external dataset).

```
// Update product price locally
1 reference | 0 overrides
public function updatePrice(Request $request, $id): JsonResponse|mixed
{
    // Validate price input
    $request->validate(rules: [
        'price' => 'required|numeric|min:0',
    ]);

    $products = collect(value: $this->fetchProducts()['products']);
    $product = $products->where(key: 'id', operator: $id)->first();

    if (!$product) {
        return response()->json(data: ['error' => 'Product not found'], status: 404);
    }

    // Update price locally (not in external API)
    $product['price'] = $request->input(key: 'price');

    return response()->json(data: $product, status: 200);
}
```

OUTPUT



vii. Complex query: Facilitate complex queries allowing users to search by name, filter by category, and sort by price, all within a single request.

```php
// Complex query (search, filter, sort)
0 references | 0 overrides
public function complexQuery(Request $request): JsonResponse|mixed
{
    $request->validate(rules: [
        'name' => 'nullable|string|min:1',
        'category' => 'nullable|string|min:1',
        'min_price' => 'numeric|min:0',
        'max_price' => 'numeric|min:0|gte:min_price',
        'sort_by' => 'nullable|in:price,title',
        'order' => 'nullable|in:asc,desc',
    ]);

    $products = collect(value: $this->fetchProducts()['products']);
    $name = strtolower(string: $request->input(key: 'name'));
    $category = strtolower(string: $request->input(key: 'category'));
    $minPrice = $request->input(key: 'min_price', default: 0);
    $maxPrice = $request->input(key: 'max_price', default: PHP_INT_MAX);
    $sortBy = $request->input(key: 'sort_by', default: 'price');
    $order = $request->input(key: 'order', default: 'asc');

    $filtered = $products->filter(callback: function (TValue $product) use ($name, $category, $minPrice, $maxPrice): bool {
        return (!$name || strpos(haystack: strtolower(string: $product['title']), needle: $name) !== false)
            && (!$category || $product['category'] === $category)
            && $product['price'] >= $minPrice
            && $product['price'] <= $maxPrice;
    });

    $sorted = $filtered->sortBy(callback: $sortBy, options: SORT_REGULAR, descending: $order == 'desc');

    return response()->json(data: $sorted->values(), status: 200);
}
```

3. Rate Limiting: Implement rate limiting to safeguard against potential abuse. Limit each user (based on their IP address) to a maximum of 50 API requests per hour.

   This implemented using the laravel throttle mechanism

```php
Route::get(uri: '/user', action: function (Request $request): mixed {
    return $request->user();
})->middleware(middleware: 'auth:sanctum');

// Group routes with throttle middleware to limit requests
Route::middleware(middleware: 'throttle:50,60')->group(callback: function (): void {

    // List all products (Limited to 50 requests per hour)
    Route::get(uri: '/products', action: [ProductController::class, 'list']);

    // Search products by keyword in the product title (case-insensitive)
    Route::get(uri: '/products/search/{keyword}', action: [ProductController::class, 'search']);

    // Filter products by category and price range using query parameters
    Route::get(uri: '/products/filter', action: [ProductController::class, 'filter']);

    // Sort products by a given field (e.g., price or title) using query parameters
    Route::get(uri: '/products/sort', action: [ProductController::class, 'sort']);

    // Get product details by ID
    Route::get(uri: '/products/{id}', action: [ProductController::class, 'show']);

    // Update product price by ID
    Route::put(uri: '/products/{id}/price', action: [ProductController::class, 'updatePrice']);

    // Additional complex queries or bulk operations can be added as needed
});
```

4. Error Handling and Validation: This is applied in many places on my ProductController like

```php
// Check if data is cached
if (Cache::has(key: 'products')) {
    return Cache::get(key: 'products');
}
```

```php
$request->validate(rules: [
    'category' => 'required|string|min:1',
    'min_price' => 'numeric|min:0',
    'max_price' => 'numeric|min:0|gte:min_price',
]);
```

```php
if (!$product) {
    return response()->json(data: ['error' => 'Product not found'], status: 404);
}

return response()->json(data: $product, status: 200);
```

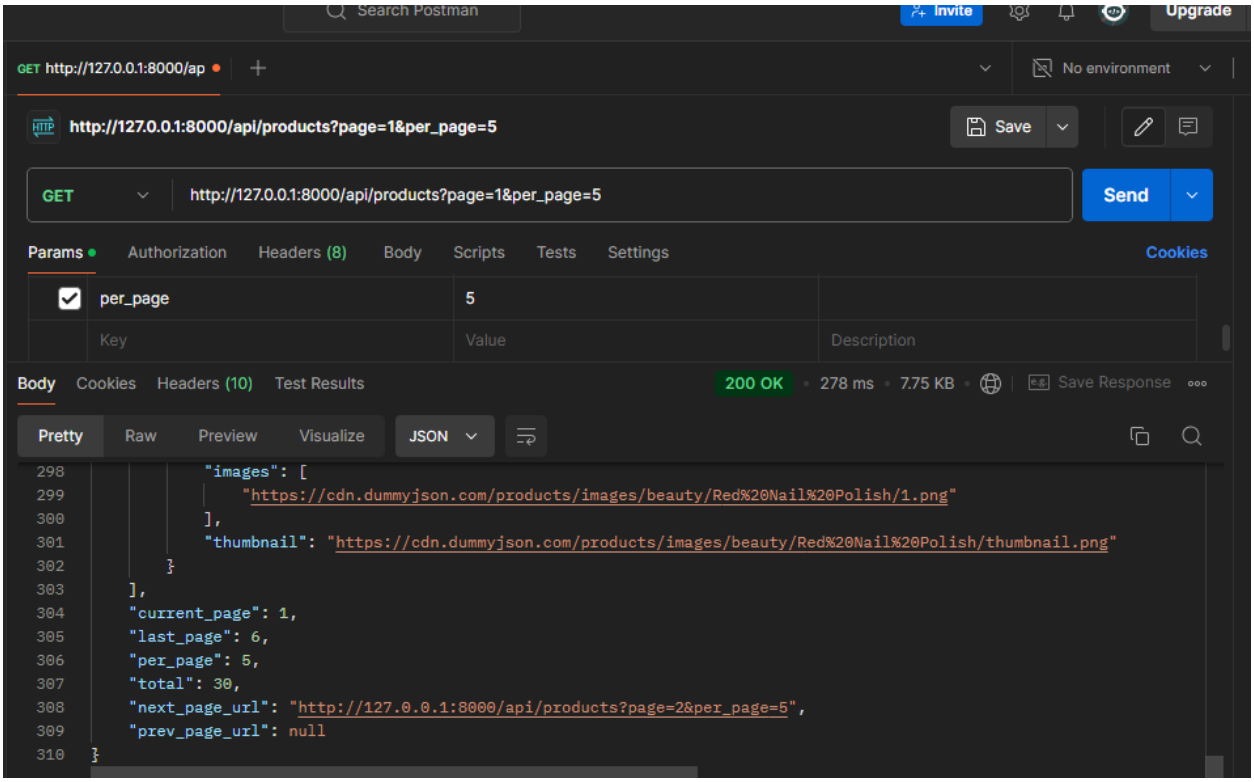
Bulk Operation

```php
//Definition of bulk operation for update
1 reference | 0 overrides
public function bulkUpdate(Request $request): JsonResponse|mixed

$request->validate(rules: [
    'updates' => 'required|array',
    'updates.*.id' => 'required|integer',
    'updates.*.price' => 'nullable|numeric|min:0',
    'updates.*.category' => 'nullable|string|min:1',
]);

$products = collect(value: $this->fetchProducts()['products']);
$updates = $request->input(key: 'updates');

foreach ($updates as $update) {
    $product = $products->where(key: 'id', operator: $update['id'])->first();

    if ($product) {
        if (isset($update['price'])) {
            $product['price'] = $update['price'];
        }
        if (isset($update['category'])) {
            $product['category'] = $update['category'];
        }
    }
}

// Return the updated products
return response()->json(data: $products->values(), status: 200);
```

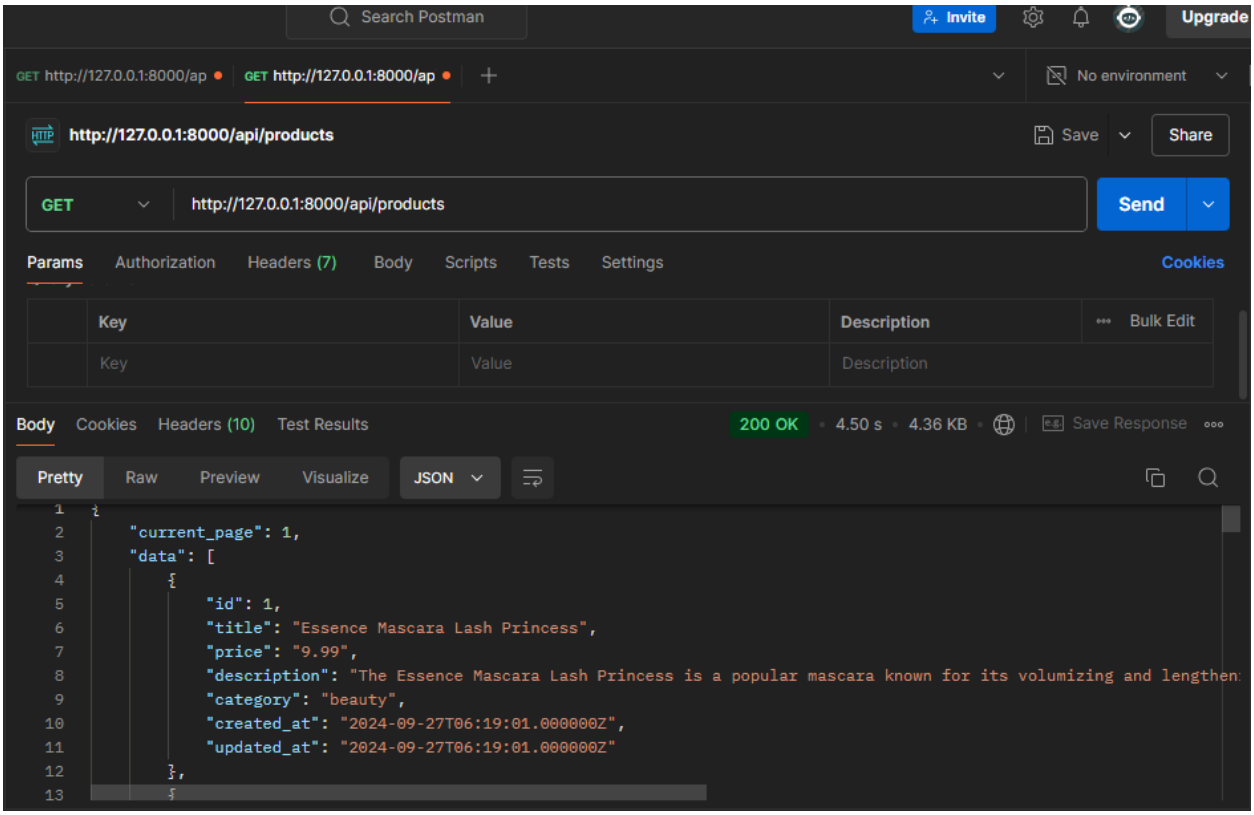
Pagination using LengthAwarePaginator

```php
// Pagination settings
$page = $request->input(key: 'page', default: 1); // Default to page 1
$perPage = $request->input(key: 'per_page', default: 10); // Default to 10 items per page

// paginatION OF the products manually
$paginatedProducts = new LengthAwarePaginator(
    // Slice the products for the current page
    items: $products->forPage(page: $page, perPage: $perPage),
    // Total number of products
    total: $products->count(),
    // Number of items per page
    perPage: $perPage,
    // Current page
    currentPage: $page,
    // Preserve query parameters
    options: ['path' => $request->url(), 'query' => $request->query()]

);

// Return the paginated data as JSON
return response()->json(data: [
    // Paginated products      You, 1 second ago • Uncommitted changes
    'products' => $paginatedProducts->items(),
    'current_page' => $paginatedProducts->currentPage(),
    'last_page' => $paginatedProducts->lastPage(),
    'per_page' => $paginatedProducts->perPage(),
    'total' => $paginatedProducts->total(),
    'next_page_url' => $paginatedProducts->nextPageUrl(),
    'prev_page_url' => $paginatedProducts->previousPageUrl(),
]);
```

OUTPUT



Database Integration: Instead of a local cache file, store the fetched data in a MySQL database. Use database queries for filtering, sorting, and search operations.



User Authentication: Add an authentication mechanism to restrict access to the API.

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;
use App\Http\Controllers\AuthController;        You, 28 minutes ago • Uncommitted changes

Route::get(uri: '/user', action: function (Request $request): mixed {
    return $request->user();
})->middleware(middleware: 'auth:sanctum');



Route::post(uri: '/register', action: [AuthController::class, 'register']);
Route::post(uri: '/login', action: [AuthController::class, 'login']);
Route::post(uri: '/logout', action: [AuthController::class, 'logout'])->middleware(middleware: 'auth:sanctum');


// Group routes with throttle middleware to limit requests
Route::middleware(middleware: ['auth:sanctum','throttle:50,60'])->group(callback: function (): void {
```

OUTPUT

GET http://127.0.0.1:8000/ap • POST http://127.0.0.1:8000/a •

No environment

http://127.0.0.1:8000/api/login?email=edward@gmail.com&password=12345678

Save Share

POST http://127.0.0.1:8000/api/login?email=edward@gmail.com&password=12345678 Send

Params • Authorization Headers (10) Body • Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "name": "Edward Kafuka",
3   "email": "edward@example.com",
4   "password": "password",
5   "password_confirmation": "password"
```

Body Cookies Headers (9) Test Results 200 OK • 589 ms • 408 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2     "access_token": "1|OJOWdLrEle807zHgyE5WXgnnXwoEWEeCsNAB8xg71c7c860c",
3     "token_type": "Bearer"
4 }
```

http://127.0.0.1:8000/api/products

Save Share

GET http://127.0.0.1:8000/api/products Send

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1
```

Body Cookies Headers (9) Test Results 401 Unauthorized • 636 ms • 357 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Unauthenticated."
3 }
```