HTTP Methods

Following well known HTTP methods are commonly used in REST based architecture.

- **GET** - Provides a read only access to a resource.
- **PUT** - Used to create a new resource.
- **DELETE** - Used to remove a resource.
- **POST** - Used to update a existing resource or create a new resource.

1

HTTP Request

A HTTP Request has five major parts:

- **Verb**- Indicate HTTP methods such as GET, POST, DELETE, PUT etc.
- **URI**- Uniform Resource Identifier (URI) to identify the resource on server
- **HTTP Version**- Indicate HTTP version, for example HTTP v1.1 .
- **Request Header**- Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by client, format of message body, cache settings etc.
- **Request Body**- Message content or Resource representation.

2

HTTP Response

A HTTP Response has four major parts:

- **Status/Response Code**- Indicate Server status for the requested resource. For example 404 means resource not found and 200 means response is ok.
- **HTTP Version**- Indicate HTTP version, for example HTTP v1.1 .
- **Response Header**- Contains metadata for the HTTP Response message as key-value pairs. For example, content length, content type, response date, server type etc.
- **Response Body**- Response message content or Resource representation.

3

HTTP Status Codes

- **Information** 1xx
- **Success** 2xx
- **Redirection** 3xx
- **Client Error** 4xx
  - Unauthorized 401
  - Forbidden 403
  - Not Found 404
- **Server Error** 5xx

4

HTTP Status

- 200: OK. The standard success code and default option.
- 201: Object created. Useful for the store actions.
- 204: No content. When an action was executed successfully, but there is no content to return.
- 206: Partial content. Useful when you have to return a paginated list of resources.
- 400: Bad request. The standard option for requests that fail to pass validation.
- 401: Unauthorized. The user needs to be authenticated.
- 403: Forbidden. The user is authenticated, but does not have the permissions to perform an action.
- 404: Not found. This will be returned automatically by Laravel when the resource is not found.
- 500: Internal server error. Ideally you're not going to be explicitly returning this, but if something unexpected breaks, this is what your user is going to receive.
- 503: Service unavailable. Pretty self explanatory, but also another code that is not going to be returned explicitly by the application.

5

Laravel API Routes

- Add the resource API route in “routes/api.php”.

```
use App\Http\Controllers\API\BookController;

Route::resource('books', BookController::class);
```

- Create the resource controller in “app/Http/Controllers/API/”.

```
php artisan make:controller Api/BookController -r
```

6

API Response – Retrieve API

- Add the following code in “app/Http/Controllers/API/ BookController”.

```
use App\Models\Book;

public function index()
{
        $book = Book::get();
        return response()->json($book,200);
 }
```

7

API Response – Create API

- Add the following code in “app/Http/Controllers/API/BookController”.
- Add Accept => application/json in Headers and form-data in Body.

```
public function store(Request $request)
  {
      $validated = $request->validate([
          'code_number' => 'required',
          'title' => 'required',
          'edition'  => 'required'
      ]);
      $book = new Book([
          'code_number' => $request->code_number,
          'title' => $request->title,
          'edition' => $request->edition
        ]);
      $book->save();
      $data = [
          'status' => 200,
          'message' => 'Book created successfully.'
      ];
      return response()->json($data, 200);
  }
```

8

API Token Authentication

- Laravel Sanctum allows you to issue API tokens or personal access tokens that may be used to authenticate API requests to your application.
- API token should be included in the Authorization header as a Bearer token.
- API tokens are hashed using SHA-256 hashing.
- Use Sanctum we need to use HasApiTokens Trait Class in User Model.

9

API Token Authentication

- Create AuthController to handle all authentication related to API.

```
php artisan make:controller Api/AuthController
```

- Add the following API in “routes/api.php”.

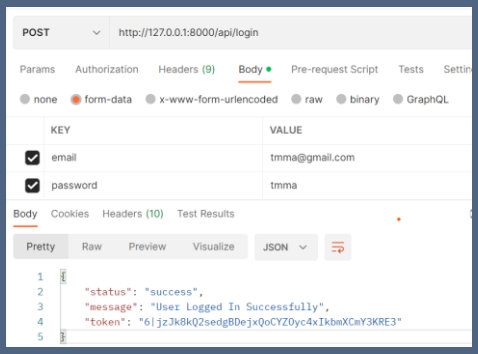
```
use App\Http\Controllers\API\AuthController;

Route::post('login', [AuthController::class, 'login']);
```

10

API Token Authentication

- Add the following API in “app/Http/Controllers/API/AuthController” login function.

```
try {
    $validated = $request->validate([
        'email' => 'required|email',
        'password' => 'required'
    ]);
    if(!Auth::attempt($request->only(['email', 'password']))){
        return response()->json(['status' => 'fail',
            'message' => 'Email & Password does not match.',
        ], 401);
    }
    $user = User::where('email', $request->email)->first();
    return response()->json([
        'status' => 'success',
        'message' => 'User Logged In Successfully',
        'token' => $user->createToken("API TOKEN")->plainTextToken
    ], 200);
} catch (\Throwable $th) {
    return response()->json('Something wrong in Login API!', 500);
}
```



API Token Authentication

- Use auth:sanctum middleware to protect API With Authentication.

```
Route::resource('books', BookController::class)->middleware('auth:sanctum');
```

- After added middleware in API route, You can see “Unauthenticated” message.
- Add token in the Authorization header as a Bearer token.