

REST API with Symphony

greenwich.edu.vn



Alliance with  Education

Table of Contents

- **RESTful Web Services**
 - Representational State Transfer (REST)
 - CRUD Operations and HTTP Methods
 - Postman – REST Client



Alliance with  Education

RESTFUL WEB SERVICES

Lightweight Architecture for Web Services

What is REST?

"Representational State Transfer (REST) is a software architecture style consisting of guidelines and best practices for creating scalable Web services."

http://en.wikipedia.org/wiki/Representational_State_Transfer

- Application state and functionality are resources
 - Every resource is associated with unique URI
 - Each resource supports standard operations (CRUD)
- This natively maps to the HTTP protocol
 - HTTP methods: GET, POST, PUT, DELETE, PATCH, OPTIONS, ...

- Request methods:
 - GET - Get resource
 - POST - Create resource
 - PUT - Update resource
 - PATCH - Partly update resource
 - DELETE - Delete resource
 - HEAD - Get headers for resource
- Response: json, xml

CRUD Operations in REST APIs

URL	HTTP Verb	POST Body	Result
http://yourdomain.com/api/entries	GET	empty	Returns all entries
http://yourdomain.com/api/entries	POST	JSON String	New entry Created
http://yourdomain.com/api/entries/:id	GET	empty	Returns single entry
http://yourdomain.com/api/entries/:id	PUT	JSON string	Updates an existing entry
http://yourdomain.com/api/entries/:id	DELETE	empty	Deletes existing entry

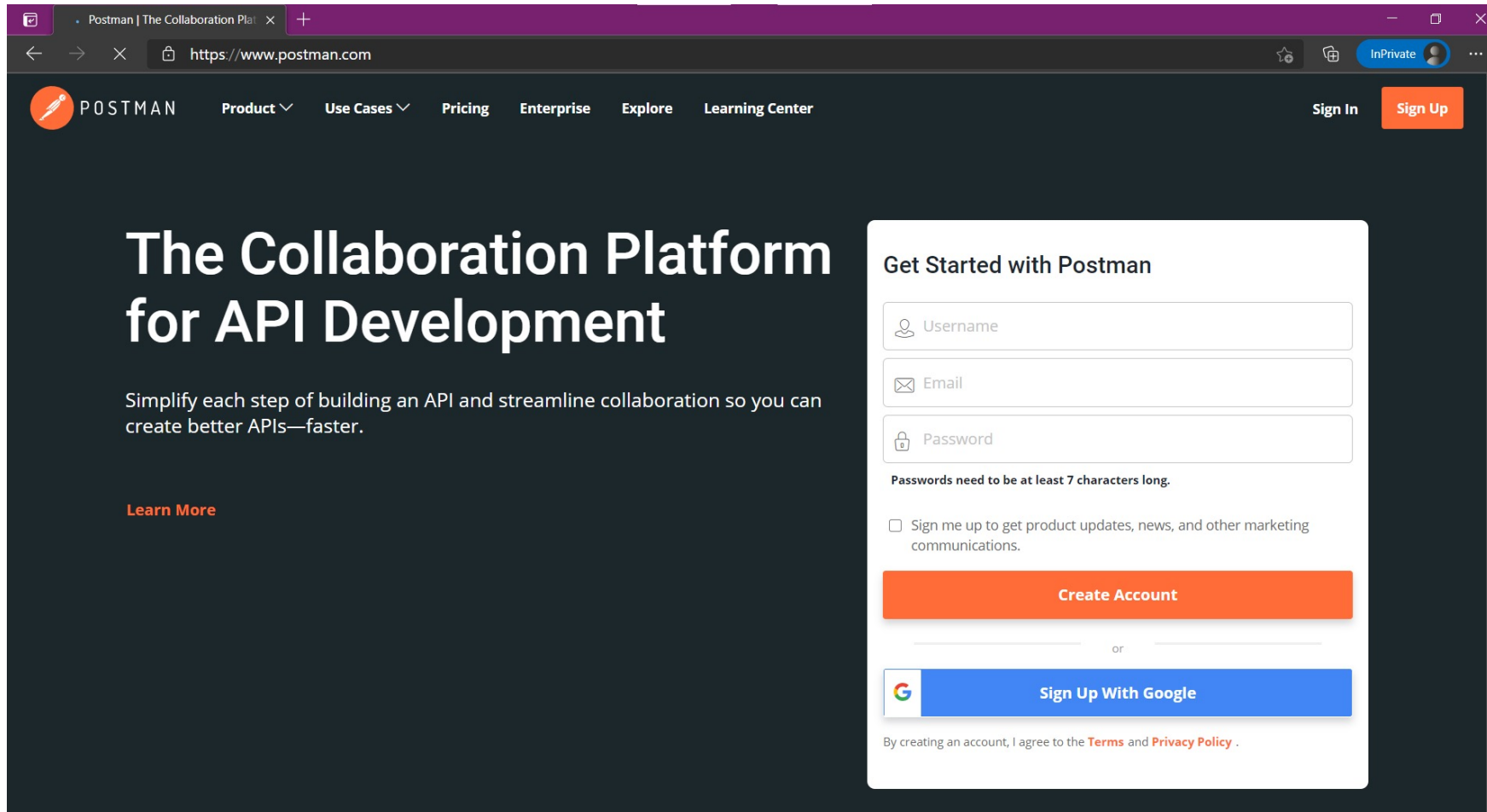
RESTful Web Services and HTTP Methods

- One URI per resource
 - Multiple operations per URI
- Get all resources / single resource by
 - GET <http://myservice.com/api/Books>
 - GET <http://myservice.com/api/Books/3>
- Add a new resource
 - POST <http://myservice.com/api/Books>
- Modify (update) a resource
 - PUT <http://myservice.com/api/Books/3>

/mongohq		
GET	/mongohq	List all tables.
POST	/mongohq	Create one or more tables.
PATCH	/mongohq	Update properties of one or more tables.
DELETE	/mongohq	Delete one or more tables.
GET	/mongohq/{table_name}	Retrieve multiple records.
POST	/mongohq/{table_name}	Create one or more records.
PUT	/mongohq/{table_name}	Update (replace) one or more records.
PATCH	/mongohq/{table_name}	Update (merge) one or more records.
DELETE	/mongohq/{table_name}	Delete one or more records.
GET	/mongohq/{table_name}/{id}	Retrieve one record by identifier.
POST	/mongohq/{table_name}/{id}	Create one record by identifier.
PUT	/mongohq/{table_name}/{id}	Update (replace) one record by identifier.
PATCH	/mongohq/{table_name}/{id}	Update (merge) one record by identifier.
DELETE	/mongohq/{table_name}/{id}	Delete one record by identifier.

RESTful Web Services and HTTP Methods (2)

- Delete (remove) a resource
 - DELETE <http://myservice.com/api/Books/3>
- Update a resource partially
 - PATCH <http://myservice.com/api/Books/3>
- Retrieve resource meta-data
 - HEAD <http://myservice.com/api/Books/3>
- Inspect resource (typically used in AJAX to request permissions)
 - OPTIONS <http://myservice.com/api/Books/3>



The screenshot shows the Postman website homepage. The browser's address bar displays "https://www.postman.com". The navigation bar includes the Postman logo, links to "Product", "Use Cases", "Pricing", "Enterprise", "Explore", and "Learning Center", and buttons for "Sign In" and "Sign Up". The main content area features the headline "The Collaboration Platform for API Development" and a sub-headline "Simplify each step of building an API and streamline collaboration so you can create better APIs—faster." Below this is a "Learn More" link. On the right, a "Get Started with Postman" form contains input fields for "Username", "Email", and "Password", a password strength note, a checkbox for marketing updates, a "Create Account" button, a "Sign Up With Google" button, and a footer note about terms and privacy policy.

POSTMAN Product Use Cases Pricing Enterprise Explore Learning Center Sign In Sign Up

The Collaboration Platform for API Development

Simplify each step of building an API and streamline collaboration so you can create better APIs—faster.

[Learn More](#)

Get Started with Postman

Username

Email


Password

Passwords need to be at least 7 characters long.

☐ Sign me up to get product updates, news, and other marketing communications.

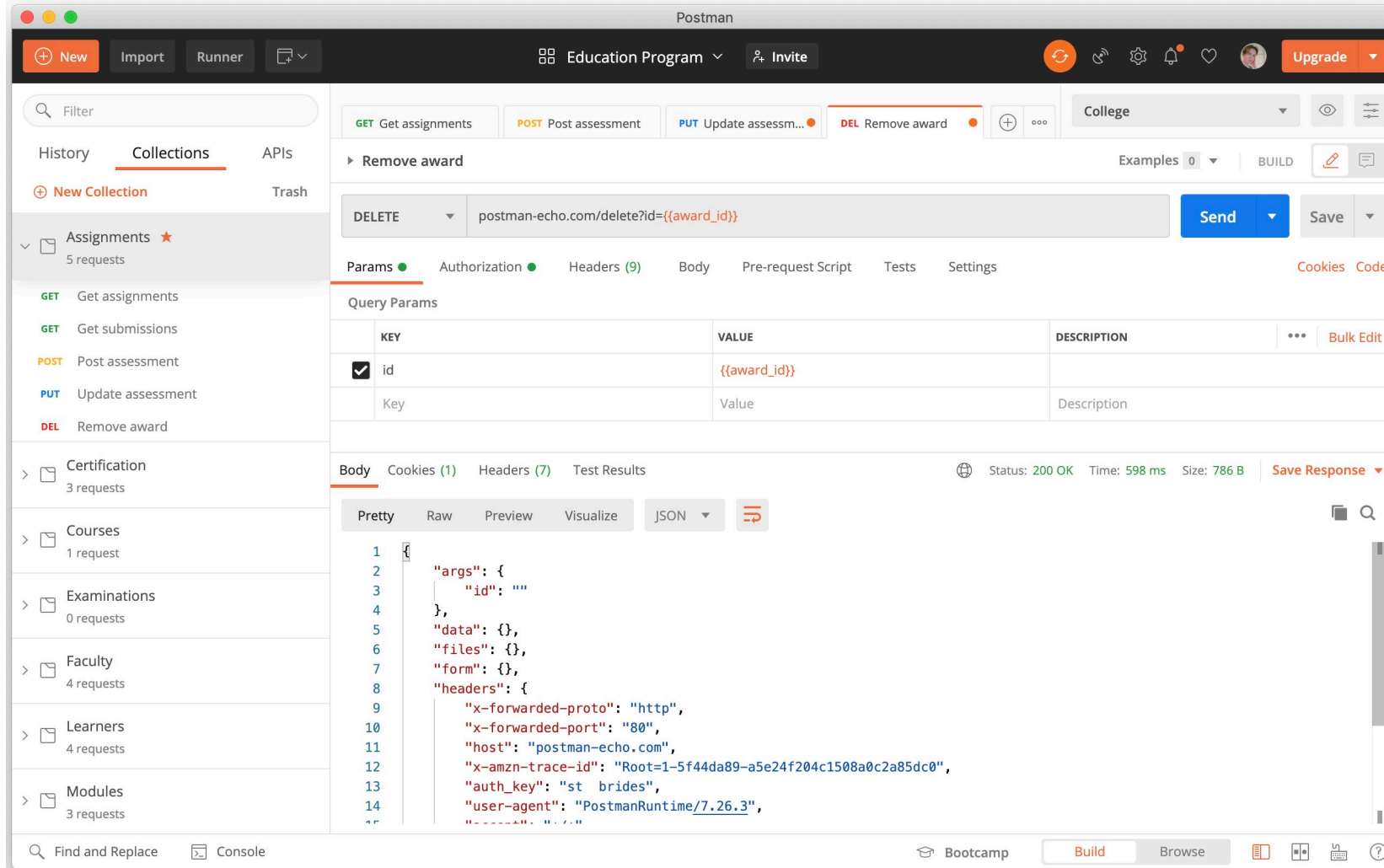
Create Account

or

 Sign Up With Google

By creating an account, I agree to the [Terms](#) and [Privacy Policy](#).

Postman – REST Client



The screenshot displays the Postman application window. The left sidebar shows a 'Collections' view with a tree structure including 'Assignments' (5 requests), 'Certification' (3 requests), 'Courses' (1 request), 'Examinations' (0 requests), 'Faculty' (4 requests), 'Learners' (4 requests), and 'Modules' (3 requests). The main workspace is configured for a DELETE request to 'postman-echo.com/delete?id={{award_id}}'. The 'Params' tab is active, showing a single query parameter 'id' with the value '{{award_id}}'. The 'Body' tab is also visible. The response section at the bottom shows a successful status of 200 OK, with a time of 598 ms and a size of 786 B. The response body is displayed in JSON format, showing a collection of objects including 'args', 'data', 'files', 'form', and 'headers'.

Request Details:

- Method: DELETE
- URL: `postman-echo.com/delete?id={{award_id}}`
- Params:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	<code>{{award_id}}</code>	
Key	Value	Description

Response Details:

- Status: 200 OK
- Time: 598 ms
- Size: 786 B
- Save Response

Response Body (JSON):

```

1 {
2   "args": {
3     "id": ""
4   },
5   "data": {},
6   "files": {},
7   "form": {},
8   "headers": {
9     "x-forwarded-proto": "http",
10    "x-forwarded-port": "80",
11    "host": "postman-echo.com",
12    "x-amzn-trace-id": "Root=1-5f44da89-a5e24f204c1508a0c2a85dc0",
13    "auth_key": "st_brides",
14    "user-agent": "PostmanRuntime/7.26.3",
15    "..."

```

REST Controllers

- A Symfony controller configured for specific http method

```
/**
 * Class CarController
 * @package App\Controller\Api
 * @Route("/api", name="car_api")
 */
class CarController extends AbstractController
{
    /**
     * @param CarRepository $carRepository
     * @return JsonResponse
     * @Route("/cars", name="api_get_cars", methods={"GET"})
     */
    public function getCars(CarRepository $carRepository, Connection $conn) :JsonResponse
```

- Add prefix to your controller using the @Route annotation

```
/**  
 * @param CarRepository $carRepository  
 * @return JsonResponse  
 * @Route("/cars", name="api_get_cars",methods={"GET"})  
 */  
public function getCars(CarRepository $carRepository, Connection $conn) :JsonResponse
```

REST Controllers

- GET method

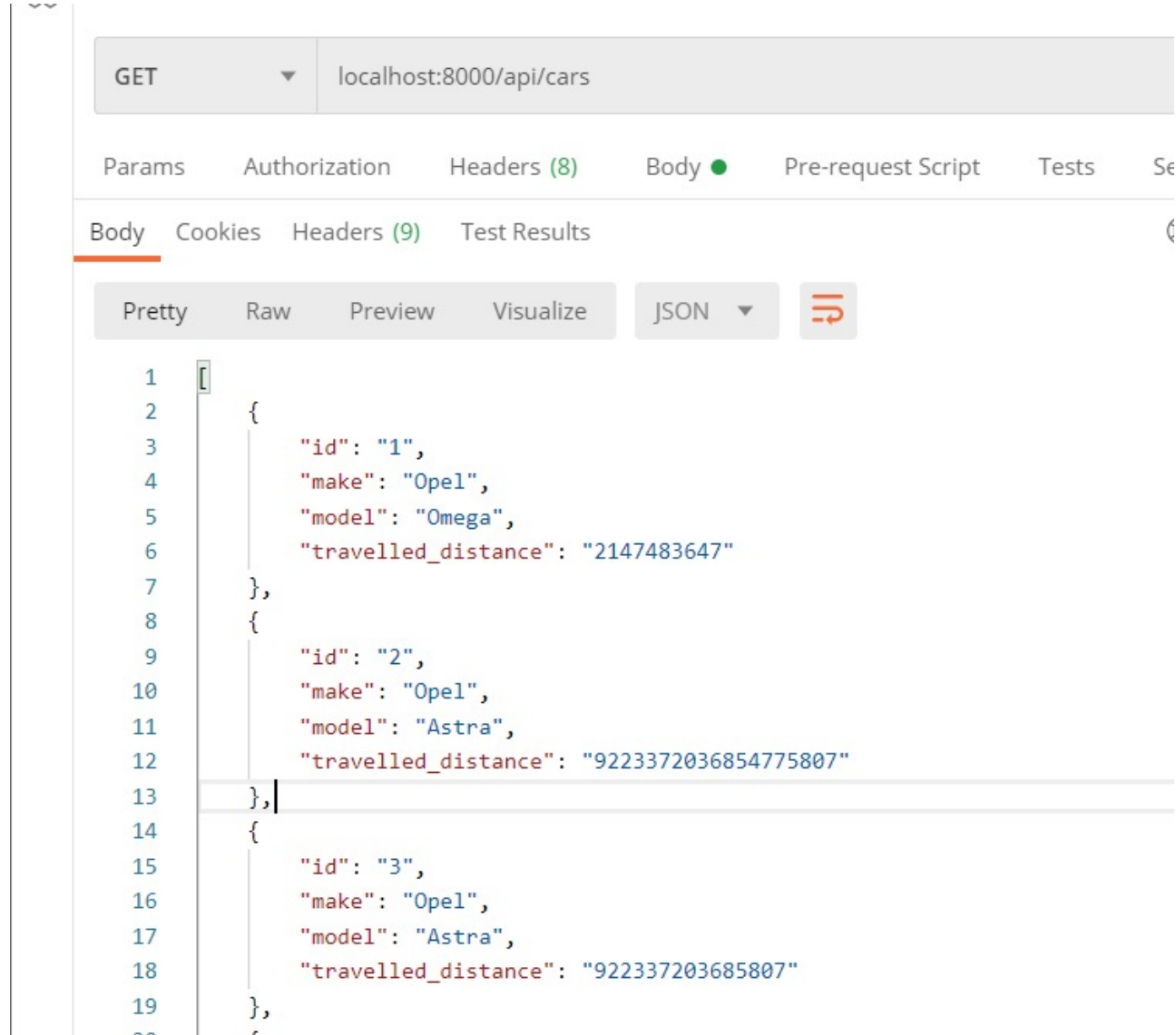
```
/**
 * @param CarRepository $carRepository
 * @return JsonResponse
 * @Route("/cars", name="api_get_cars",methods={"GET"})
 */
public function getCars(CarRepository $carRepository, Connection $conn) :JsonResponse
{
    $queryBuilder = $conn->createQueryBuilder();
    $data = $queryBuilder
        ->select('*')
        ->from('car','c')
        ->execute()
        ->fetchAll();

    return $this->response($data);
}

/**
 * Returns a JSON response
 *
 * @param array $data
 * @param $status
 * @param array $headers
 * @return JsonResponse
 */
public function response($data, $status = 200, $headers = [])
{
    return new JsonResponse($data, $status, $headers);
}
```

REST Controllers

- GET method



The screenshot shows a REST client interface with a GET request to `localhost:8000/api/cars`. The response is displayed in JSON format, showing a list of three cars. The first car has an id of 1, make of Opel, model of Omega, and a travelled distance of 2147483647. The second car has an id of 2, make of Opel, model of Astra, and a travelled distance of 9223372036854775807. The third car has an id of 3, make of Opel, model of Astra, and a travelled distance of 922337203685807.

```
1 [
2   {
3     "id": "1",
4     "make": "Opel",
5     "model": "Omega",
6     "travelled_distance": "2147483647"
7   },
8   {
9     "id": "2",
10    "make": "Opel",
11    "model": "Astra",
12    "travelled_distance": "9223372036854775807"
13  },
14  {
15    "id": "3",
16    "make": "Opel",
17    "model": "Astra",
18    "travelled_distance": "922337203685807"
19  },
20 ]
```

REST Controllers

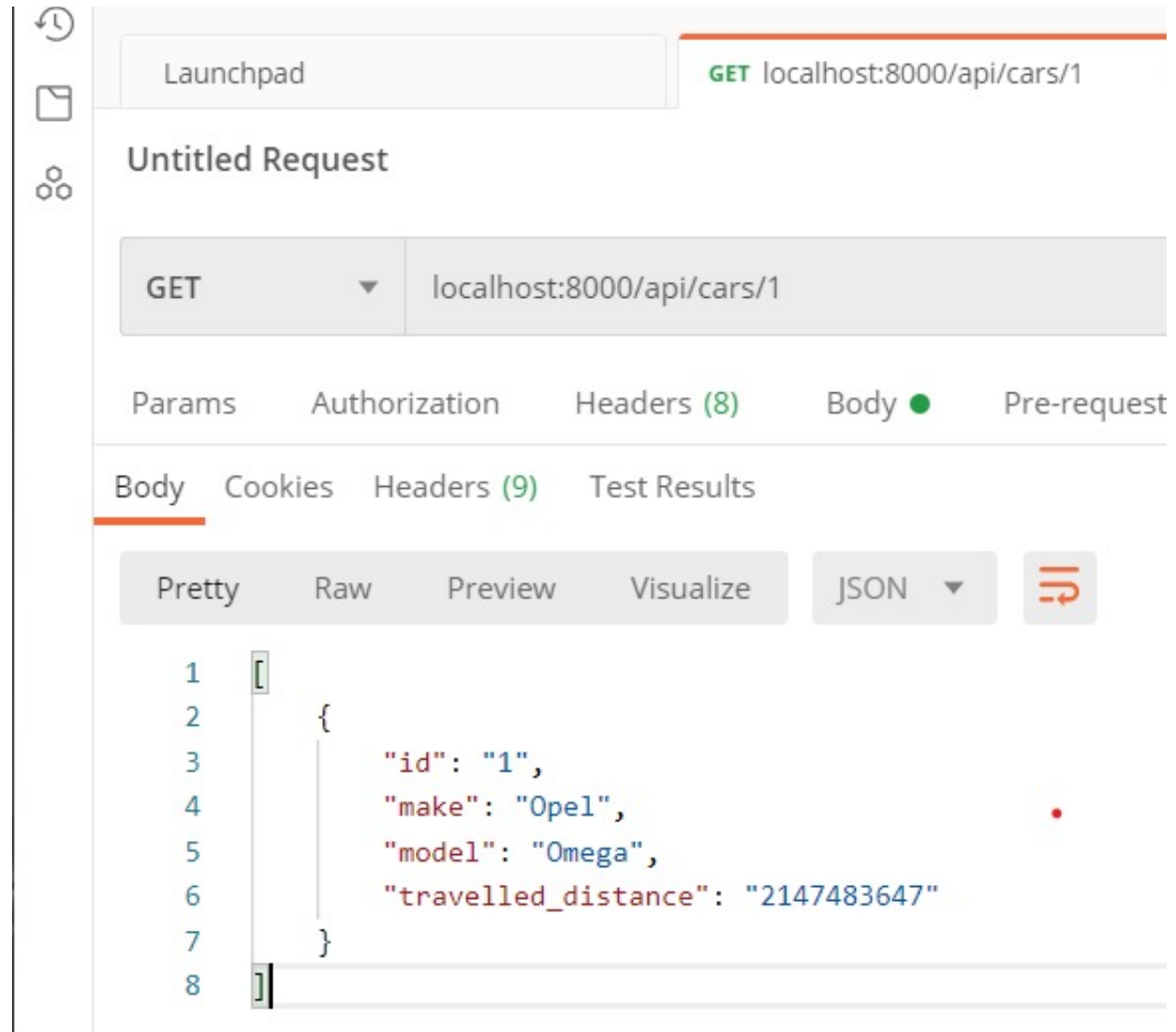
- GET by Id method

```
/**
 * @param $id
 * @param Connection $conn
 * @return JsonResponse
 * @Route("/cars/{id}",name="api_get_car_by_id")
 */
public function getCarById(int $id, Connection $conn) : JsonResponse{
    $queryBuilder = $conn->createQueryBuilder();
    $data = $queryBuilder
        ->select('*')
        ->from('car','c')
        ->where('c.id =:id')
        ->setParameter(':id', $id)
        ->execute()
        ->fetchAll();

    return $this->response($data);
}
```


REST Controllers

- GET by Id method



The screenshot displays a REST client interface. At the top, a 'Launchpad' tab shows the endpoint `GET localhost:8000/api/cars/1`. Below it, an 'Untitled Request' is configured with the `GET` method and the same endpoint. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response is a single object within an array, representing a car with the following details:

```
[
  {
    "id": "1",
    "make": "Opel",
    "model": "Omega",
    "travelled_distance": "2147483647"
  }
]
```


- POST method

```
protected function transformJsonBody(Request $request)
{
    $data = json_decode($request->getContent(), associative: true);

    if ($data === null) {
        return $request;
    }

    $request->request->replace($data);

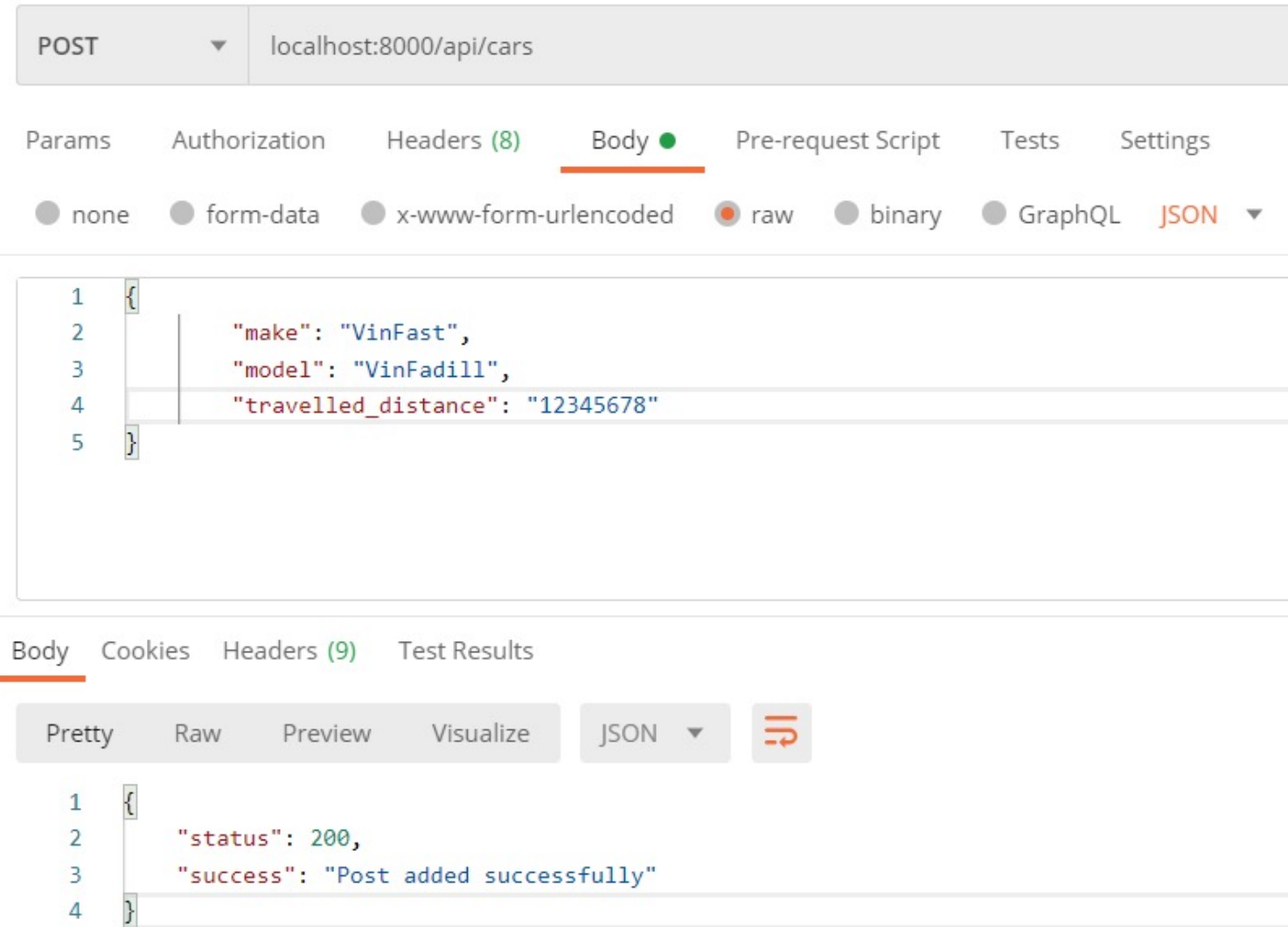
    return $request;
}
```

```
/**
 * @param Request $request
 * @param EntityManagerInterface $entityManager
 * @Route("/cars", name="cars_add", methods={"POST"})
 */
public function addCar(Request $request,
    EntityManagerInterface $entityManager)
{
    $request = $this->transformJsonBody($request);
    $car = new Car();
    $car->setMake($request->get(key: 'make'));
    $car->setModel($request->get(key: 'model'));
    $car->setTravelledDistance($request->get(key: 'travelled_distance'));

    $entityManager->persist($car);
    $entityManager->flush();
    $data = [
        'status' => 200,
        'success' => "Post added successfully",
    ];
    return $this->response($data);
}
```

REST Controllers

- POST method



The screenshot displays a REST client interface. At the top, the method is set to **POST** and the URL is `localhost:8000/api/cars`. Below this, several tabs are visible: **Params**, **Authorization**, **Headers (8)**, **Body** (which is selected and underlined), **Pre-request Script**, **Tests**, and **Settings**. Under the **Body** tab, there are radio buttons for different content types: **none**, **form-data**, **x-www-form-urlencoded**, **raw** (which is selected), **binary**, and **GraphQL**. A dropdown menu next to **JSON** is also present. The main area shows the request body as a JSON object:

```
1 {  
2   "make": "VinFast",  
3   "model": "VinFadill",  
4   "travelled_distance": "12345678"  
5 }
```

Below the request body, another set of tabs is shown: **Body** (selected and underlined), **Cookies**, **Headers (9)**, and **Test Results**. Under the **Body** tab, there are buttons for **Pretty**, **Raw**, **Preview**, and **Visualize**. A dropdown menu shows **JSON** and a button with a list icon. The response body is displayed as a JSON object:

```
1 {  
2   "status": 200,  
3   "success": "Post added successfully"  
4 }
```

REST Controllers

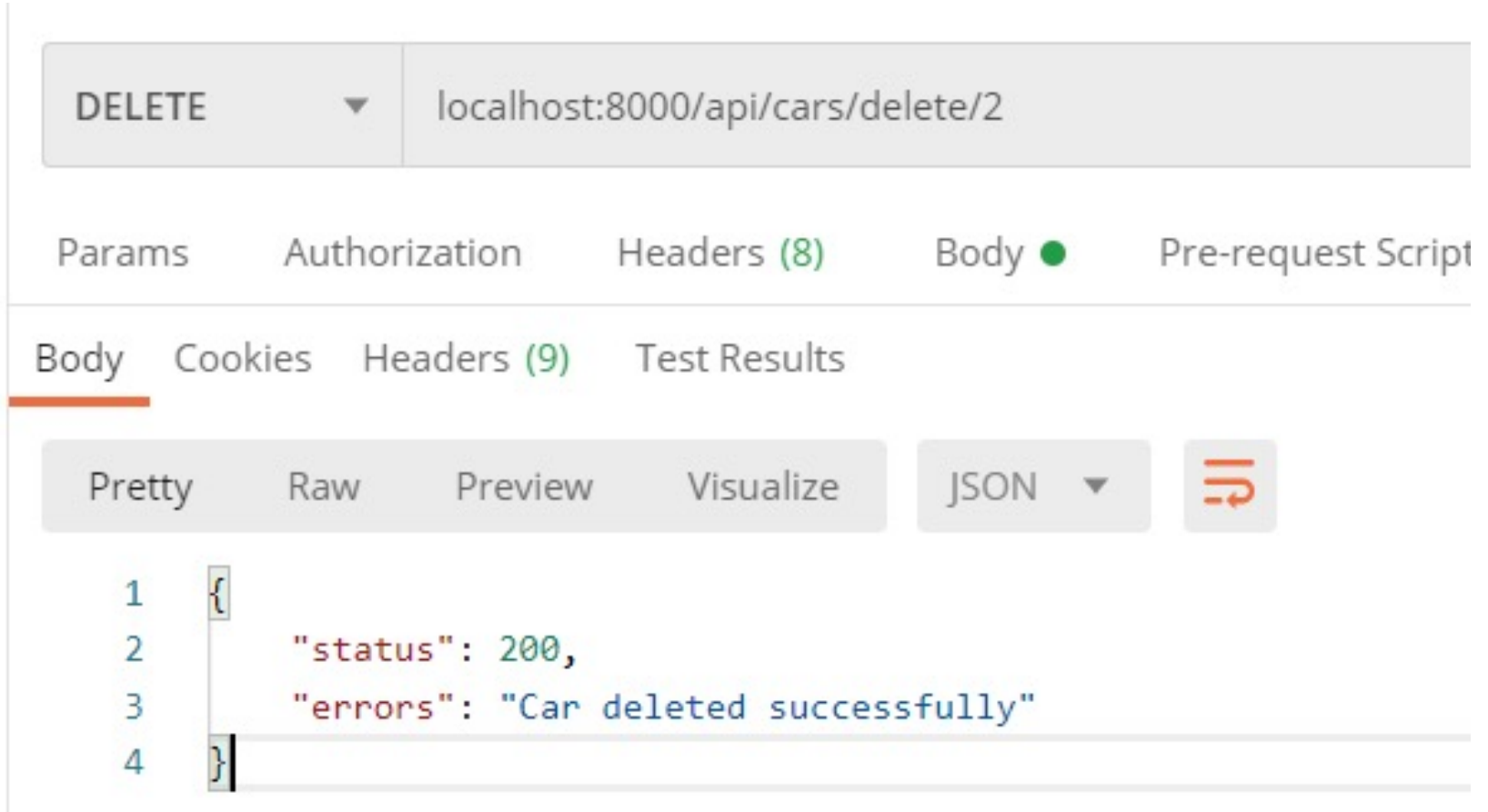
- DELETE method

```
/**
 * @param $id
 * @param EntityManagerInterface $entityManager
 * @param CarRepository $carRepository
 * @Route("/cars/delete/{id}", name="delete_car", methods={"DELETE"})
 */
public function deleteCar($id,
                          EntityManagerInterface $entityManager,
                          CarRepository $carRepository) : JsonResponse
{
    $carInDb = $carRepository->find($id);
    if (!$carInDb) {
        $data = [
            'status' => 404,
            'errors' => "Car not found",
        ];
        return $this->response($data, status: 404);
    }

    $entityManager->remove($carInDb);
    $entityManager->flush();
    $data = [
        'status' => 200,
        'errors' => "Car deleted successfully",
    ];
    return $this->response($data);
}
```

REST Controllers

- DELETE method



DELETE ▼ localhost:8000/api/cars/delete/2

Params Authorization Headers (8) Body ● Pre-request Script

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON ▼ ↺

```
1 {
2   "status": 200,
3   "errors": "Car deleted successfully"
4 }
```

REST Controllers

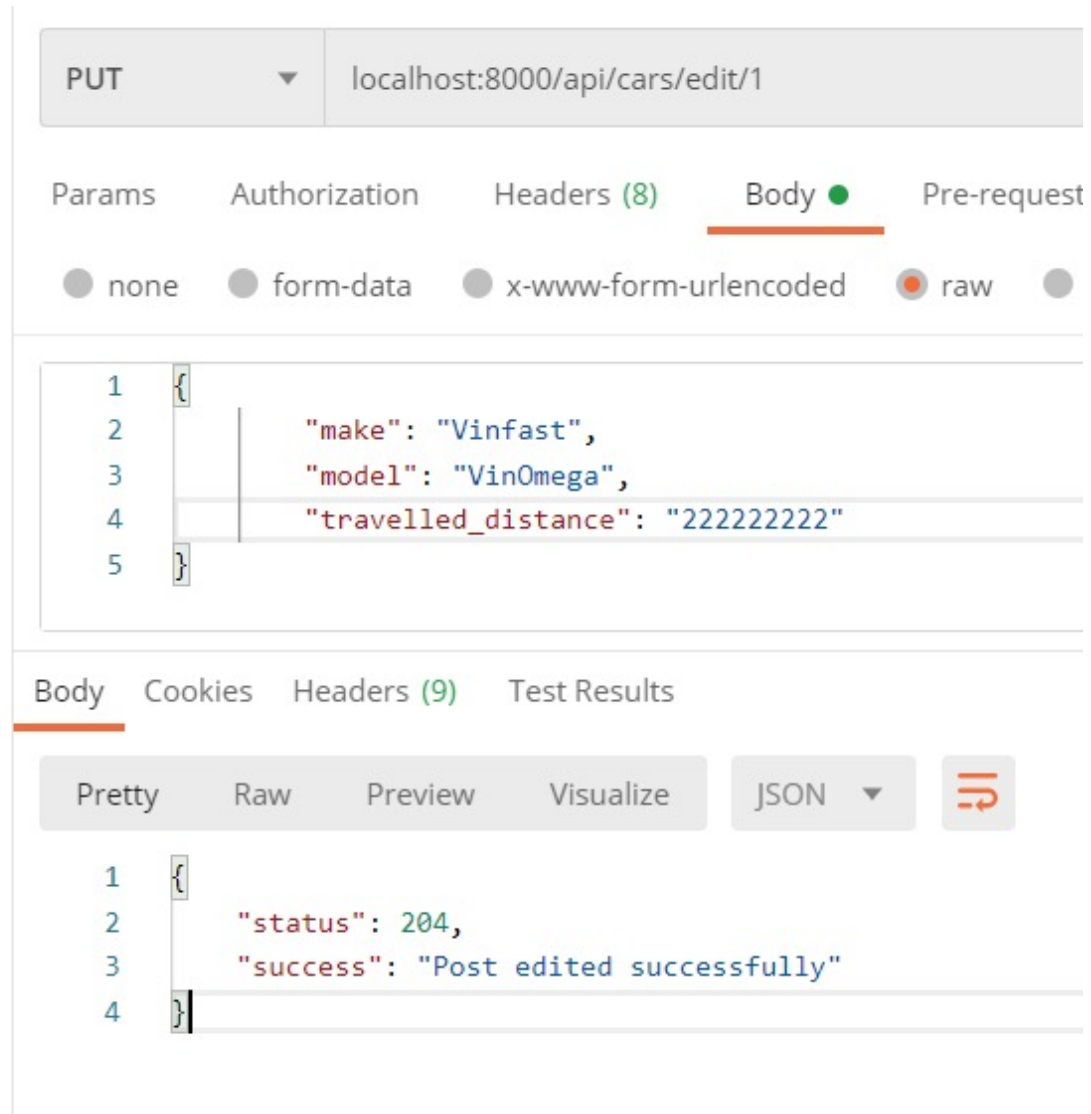
- PUT method

```
/**
 * @param $id
 * @param EntityManagerInterface $entityManager
 * @return JsonResponse
 * @Route("/cars/edit/{id}", name="edit_car", methods={"PUT"})
 */
public function editCar($id,
                        EntityManagerInterface $entityManager,
                        Request $request,
                        CarRepository $carRepository) : JsonResponse
{
    $request = $this->transformJsonBody($request);
    $car = $carRepository->find($id);
    $car->setMake($request->get(key: 'make'));
    $car->setModel($request->get(key: 'model'));
    $car->setTravelledDistance($request->get(key: 'travelled_distance'));

    $entityManager->persist($car);
    $entityManager->flush();
    $data = [
        'status' => 204,
        'success' => "Post edited successfully",
    ];
    return $this->response($data);
}
```

REST Controllers

- PUT method



The screenshot displays a REST client interface with the following details:

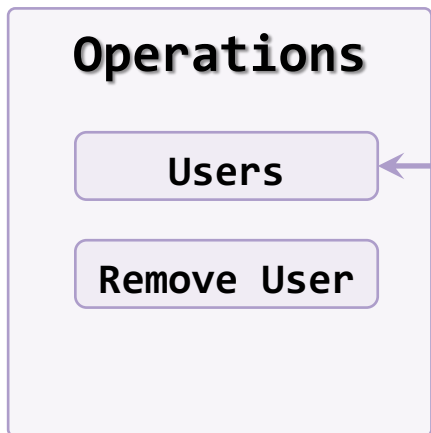
- Method:** PUT
- URL:** localhost:8000/api/cars/edit/1
- Body Tab:** Selected, showing a JSON payload:

```
1 {  
2   "make": "Vinfast",  
3   "model": "VinOmega",  
4   "travelled_distance": "222222222"  
5 }
```
- Response Tab:** Selected, showing a JSON response:

```
1 {  
2   "status": 204,  
3   "success": "Post edited successfully"  
4 }
```


RESTful API – Example

Server



Web Client (JavaScript and jQuery)

```
$.post("api/register", credentials, 'json');
```

```
$.post("api/login", credentials, 'json');
```

```
$.getJSON("api/users");
```

- JSON (**J**ava**S**cript **O**bject **N**otation)
 - Standard for representing data structures and associative arrays
 - Lightweight text-based open standard
 - Derived from the JavaScript language

```
{  
  "firstName": "John", "lastName": "Smith", "age": 25,  
  "address": { "streetAddress": "17 Tintyava Str.",  
    "city": "Sofia", "postalCode": "1113" },  
  "phoneNumber": [{ "type": "home", "number": "212 555-1234"},  
    { "type": "fax", "number": "646 555-4567" }]  
},  
{ "firstName": "Bay", "lastName": "Ivan", "age": 79 }
```



Summary

- REST
- Rest concepts
- Rest URIs
- Responses
- CRUD Operations and HTTP Methods