REST API with Symphony

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

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

RESTFUL WEB SERVICES

Lightweight Architecture for Web Services

What is REST?

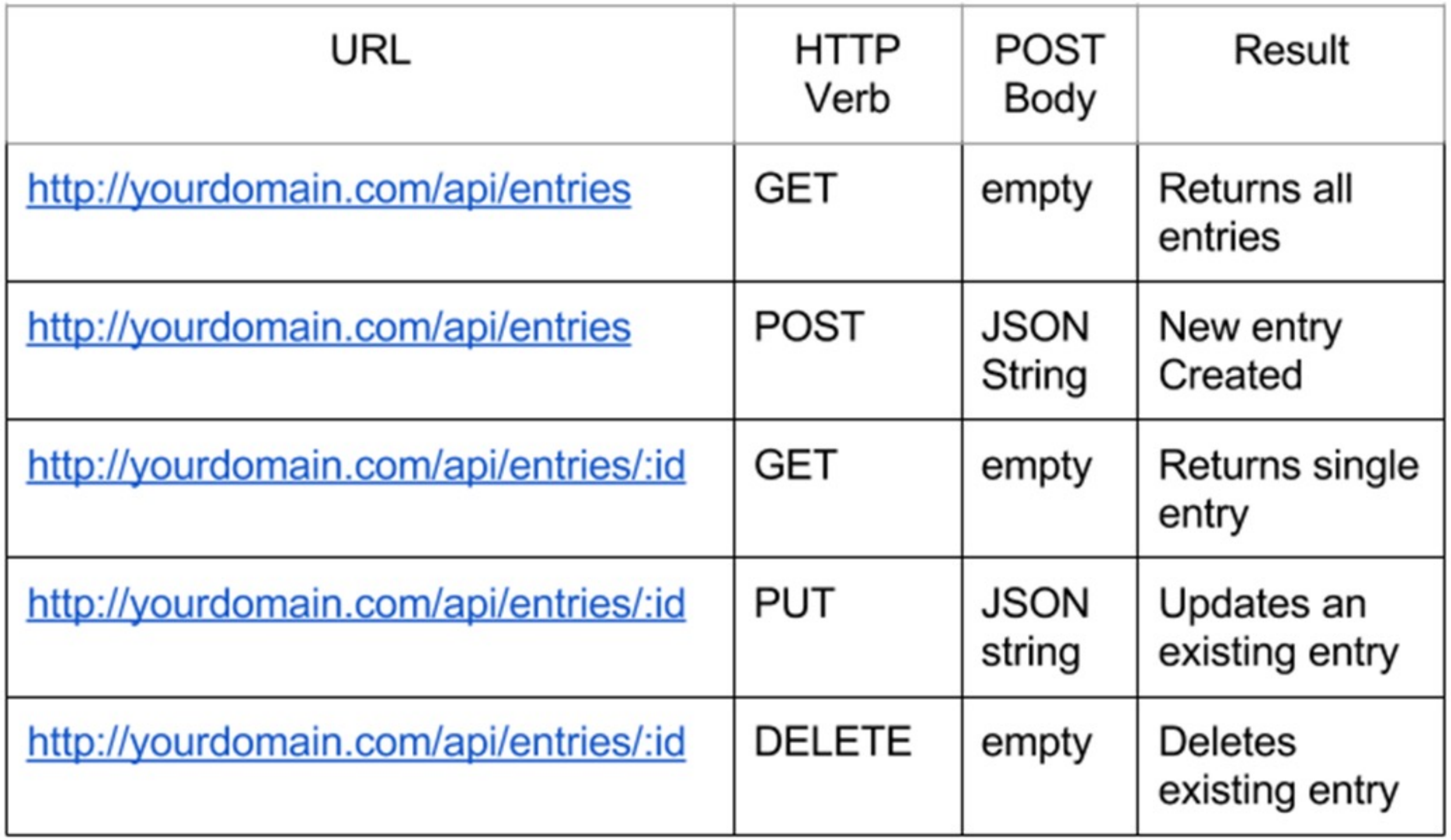
"Representational State Transfer (REST) is a software architecture style consisting of guidelines and best practices for creating scalable Web services." http://en.wikipedia.org/wiki/Representational_State_Transfer

- 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, …

REST & HTTP

- 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

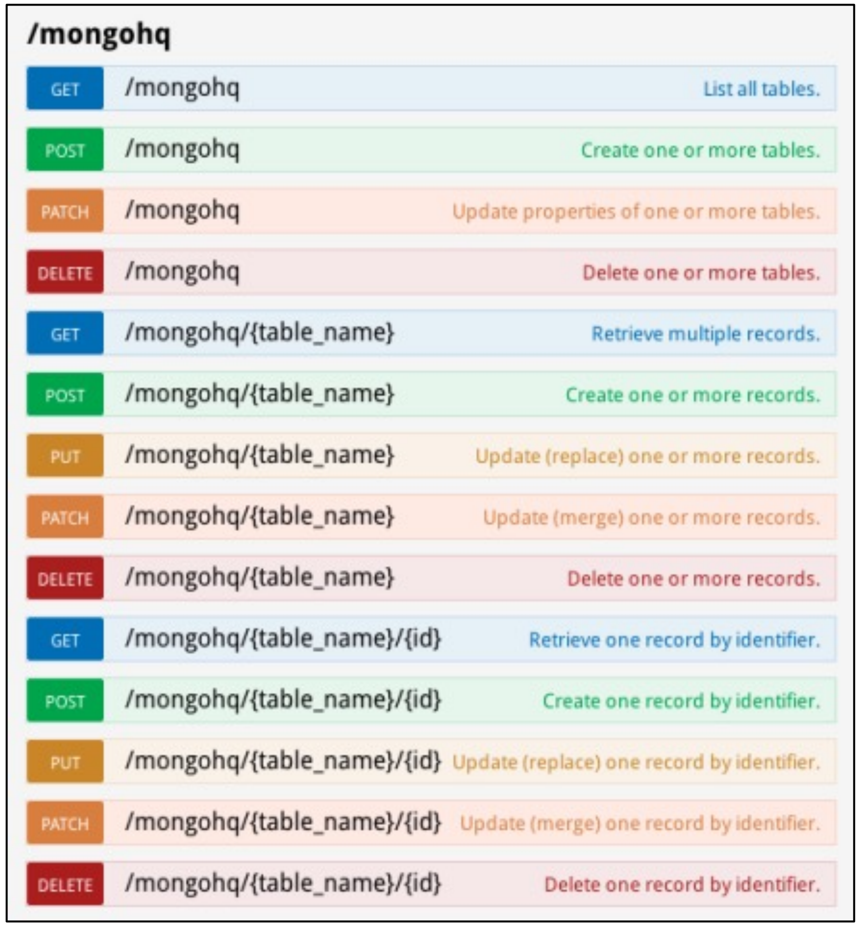
CRUD Operations in REST APIs



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



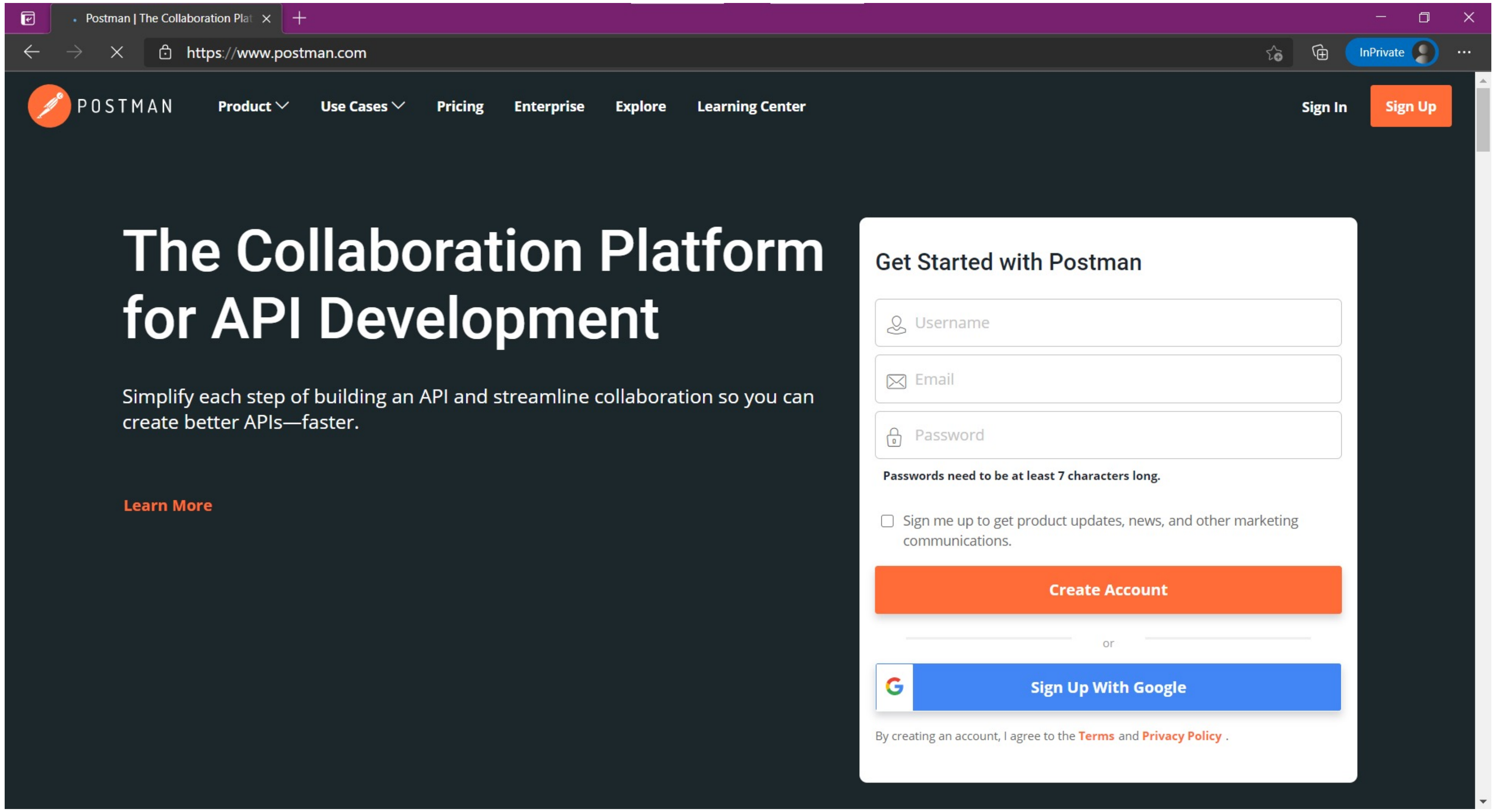
/mongohq

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

Postman



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.

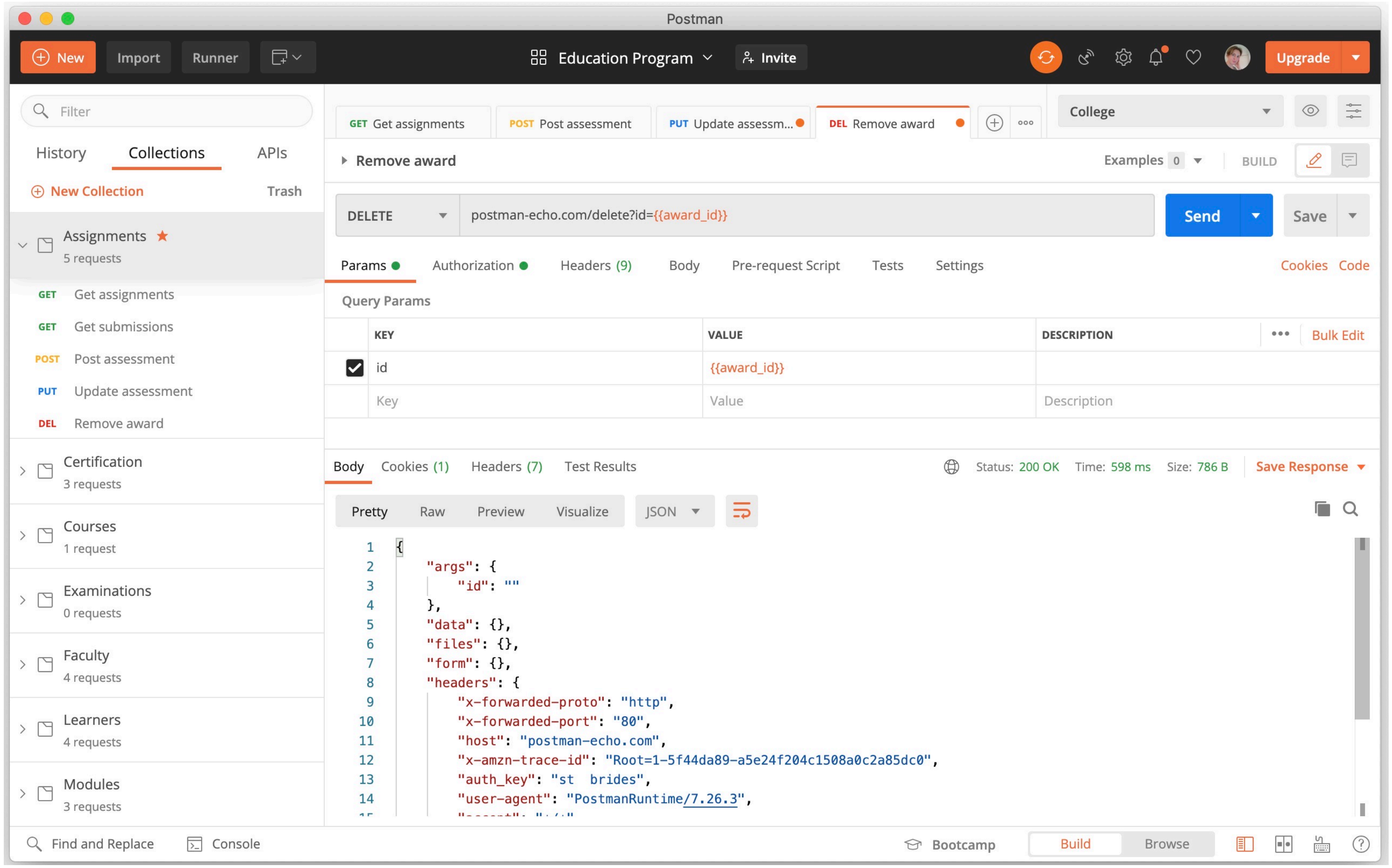
Create Account

or

Sign Up With Google

By creating an account, I agree to the Terms and Privacy Policy .

Postman – REST Client



DELETE postman-echo.com/delete?id={{award_id}}

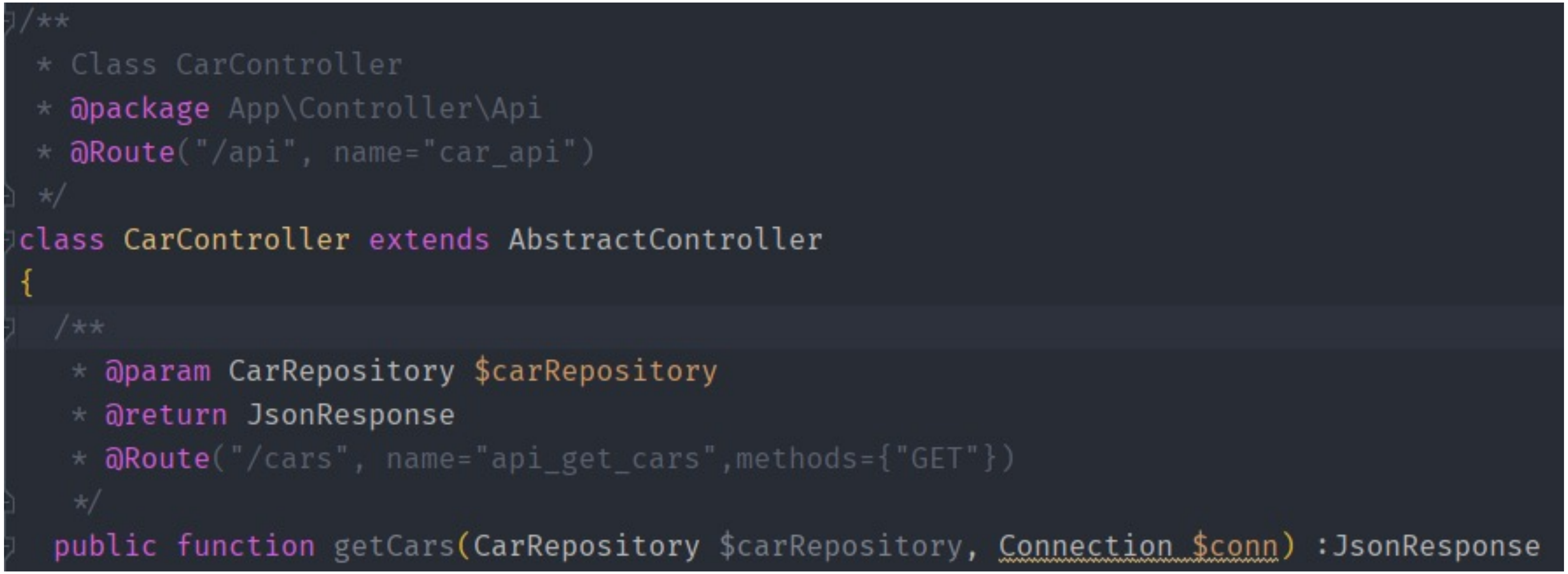
KEY	VALUE	DESCRIPTION
✓ id	{{award_id}}	
Key	Value	Description

Status: 200 OK Time: 598 ms Size: 786 B

```
{
    "args": {
        "id": ""
    },
    "data": {},
    "files": {},
    "form": {},
    "headers": {
        "x-forwarded-proto": "http",
        "x-forwarded-port": "80",
        "host": "postman-echo.com",
        "x-amzn-trace-id": "Root=1-5f44da89-a5e24f204c1508a0c2a85dc0",
        "auth_key": "st  brides",
        "user-agent": "PostmanRuntime/7.26.3",
```

REST Controllers

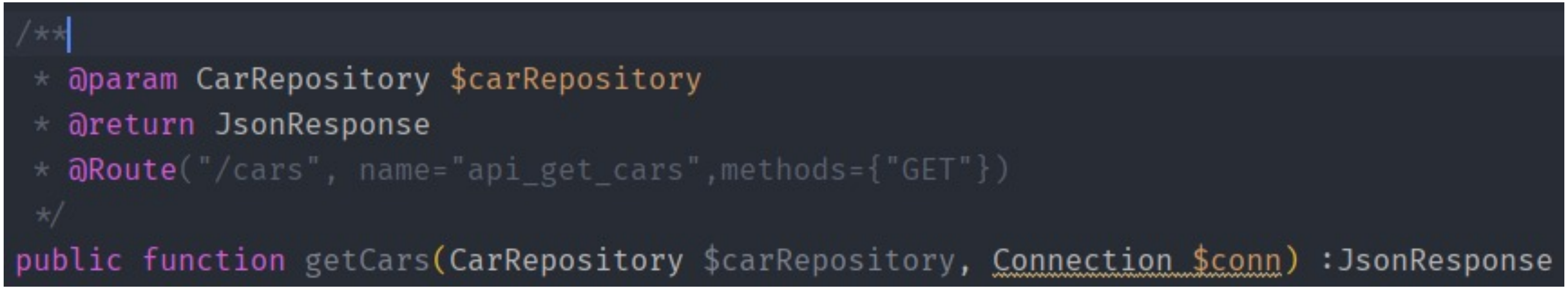
- A Symfony controller configured for specific http method



```
/**
 * 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
```

REST Controllers

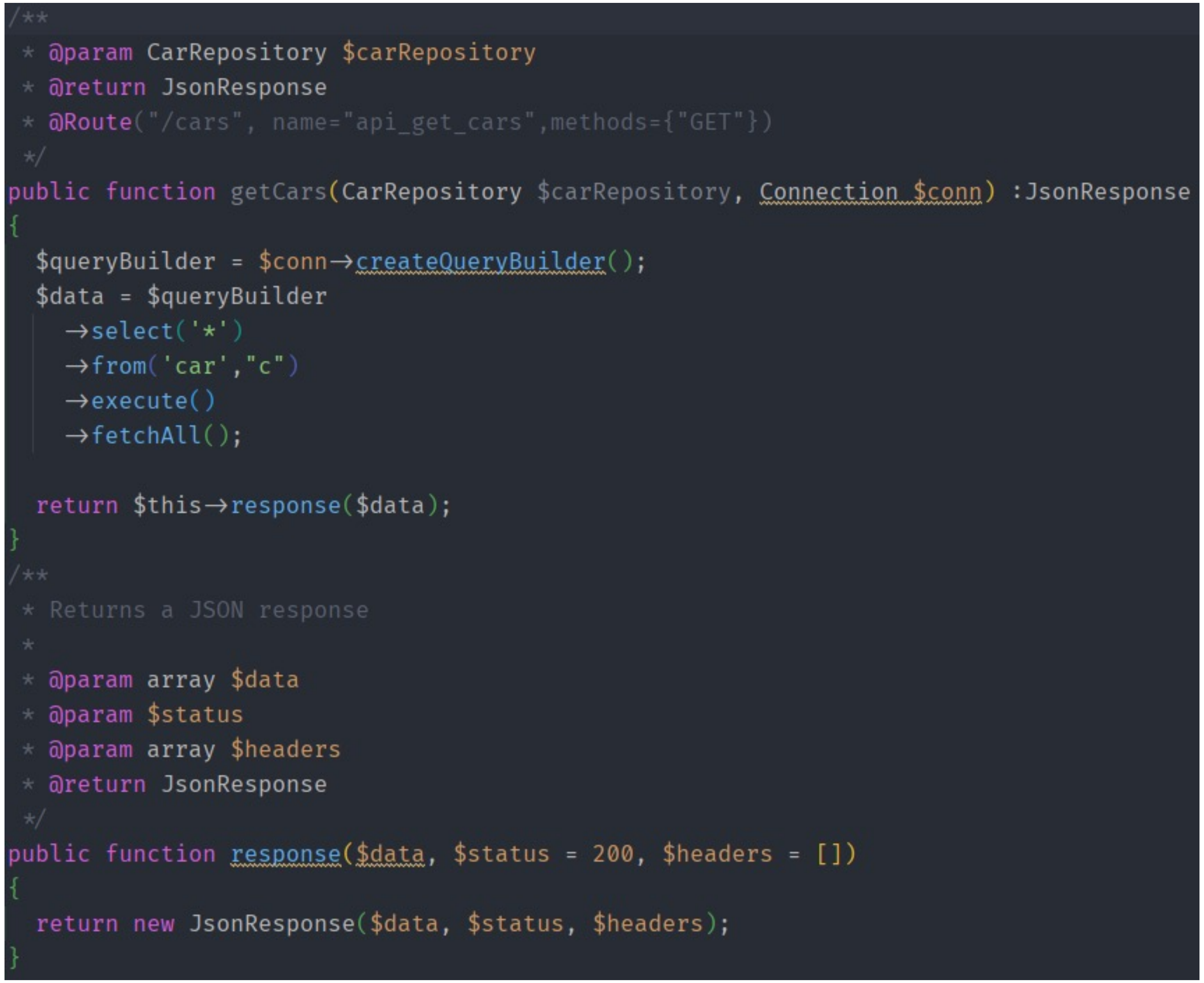
- Add prefix to your controller using the @Route annotation



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

REST Controllers

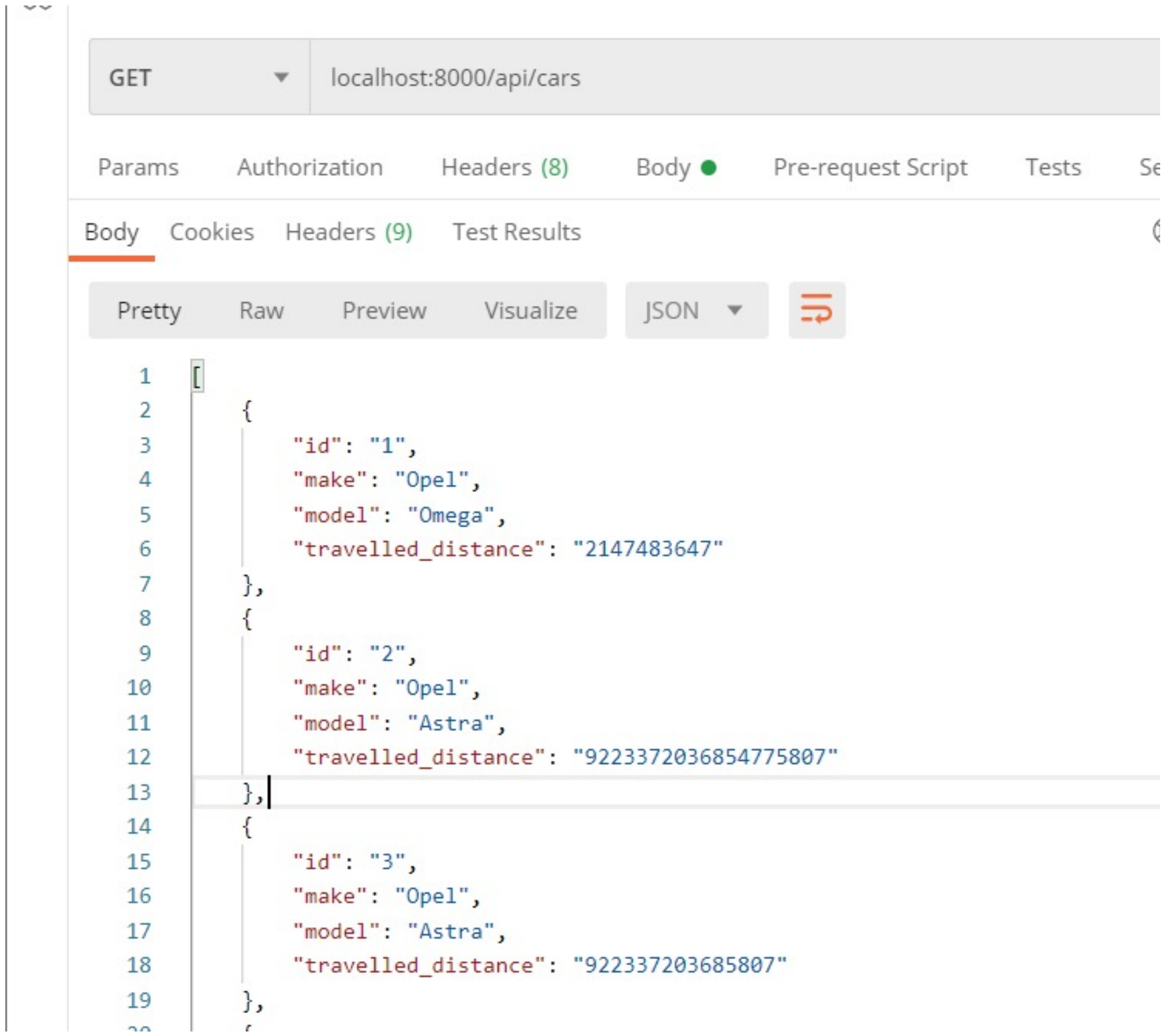
- GET method



```php
/**
 * @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

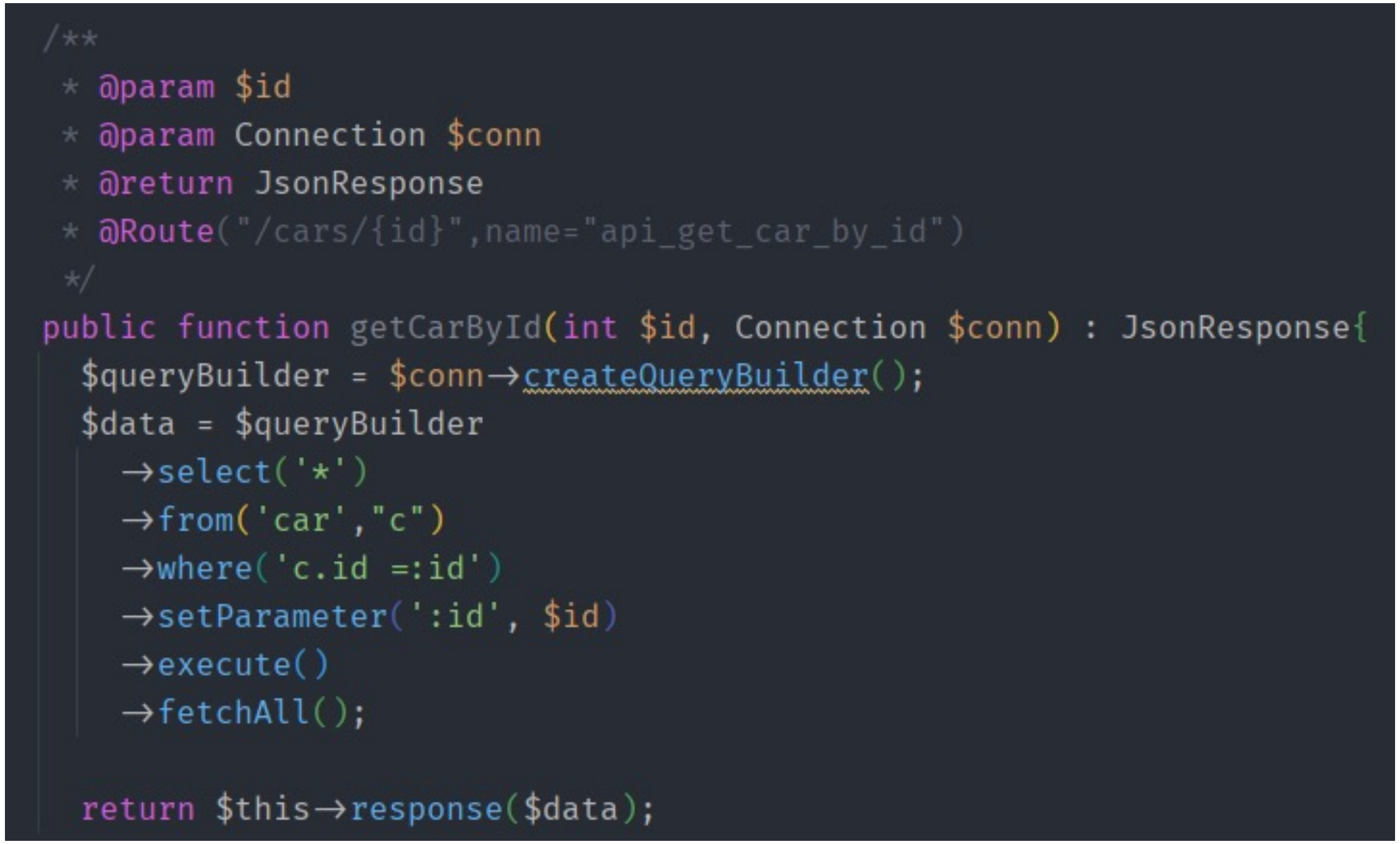


GET localhost:8000/api/cars

```
[
    {
        "id": "1",
        "make": "Opel",
        "model": "Omega",
        "travelled_distance": "2147483647"
    },
    {
        "id": "2",
        "make": "Opel",
        "model": "Astra",
        "travelled_distance": "9223372036854775807"
    },
    {
        "id": "3",
        "make": "Opel",
        "model": "Astra",
        "travelled_distance": "922337203685807"
    },
```

REST Controllers

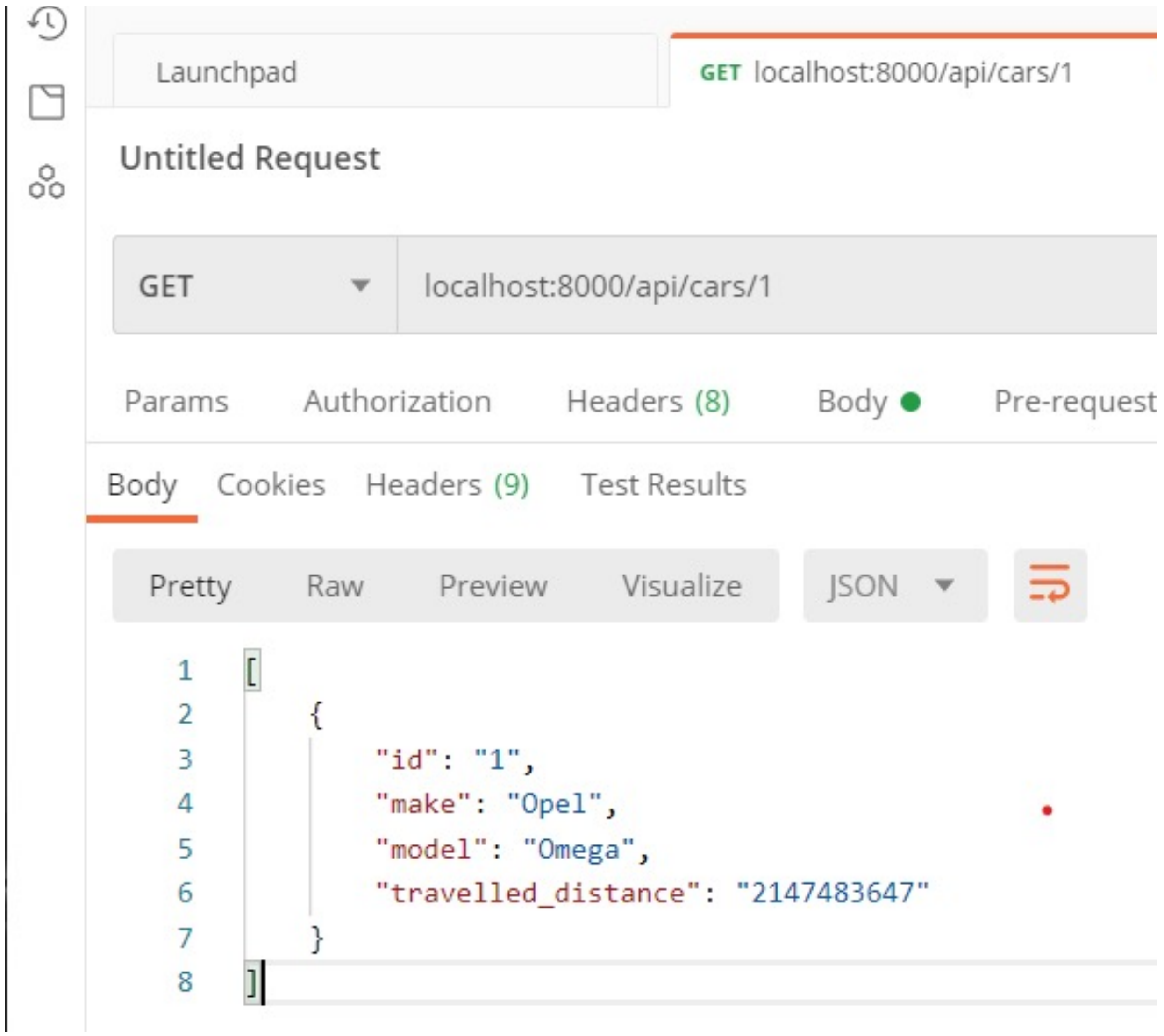
- GET by Id method



```
/**
 * @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



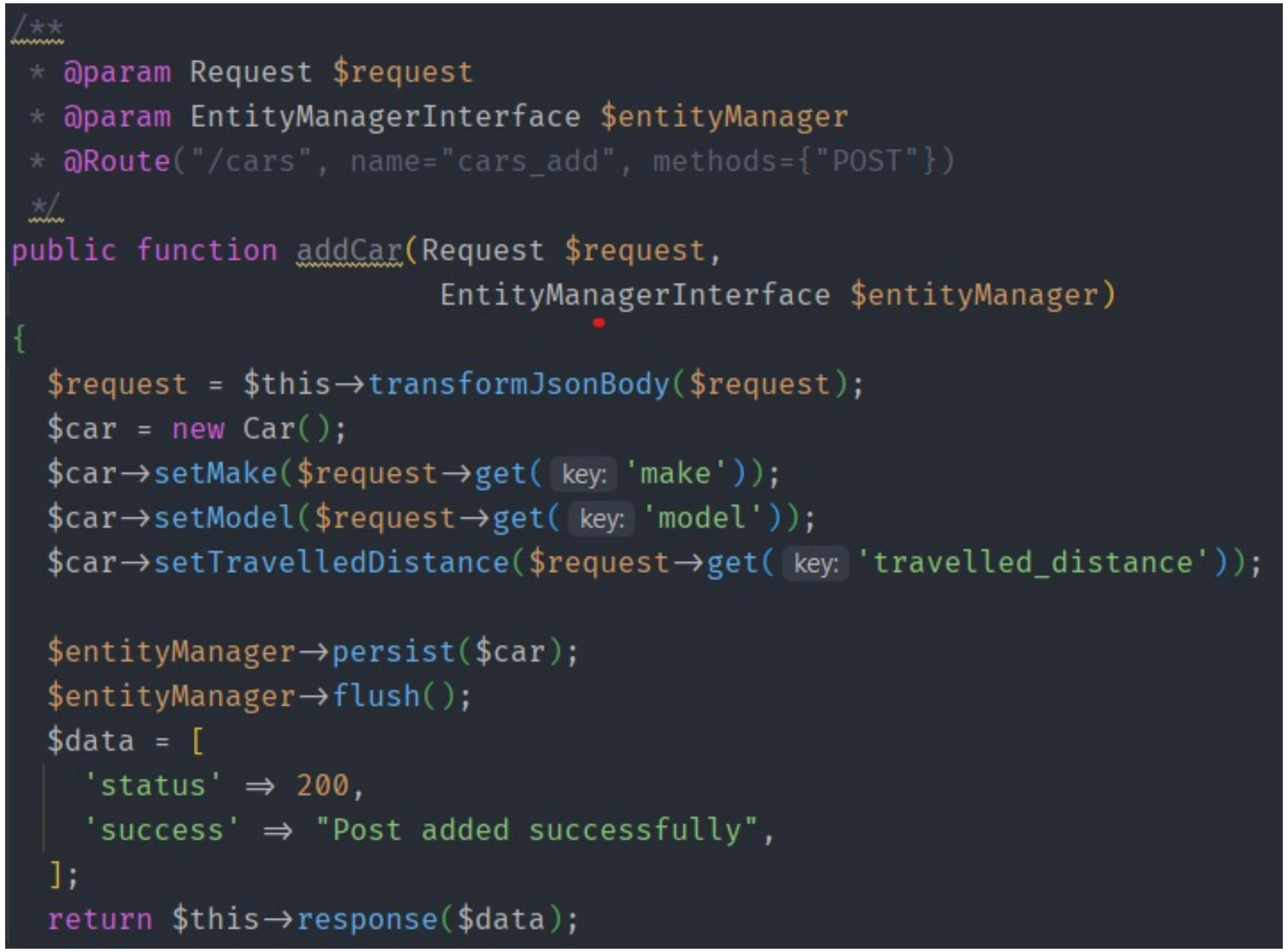
GET localhost:8000/api/cars/1

GET localhost:8000/api/cars/1

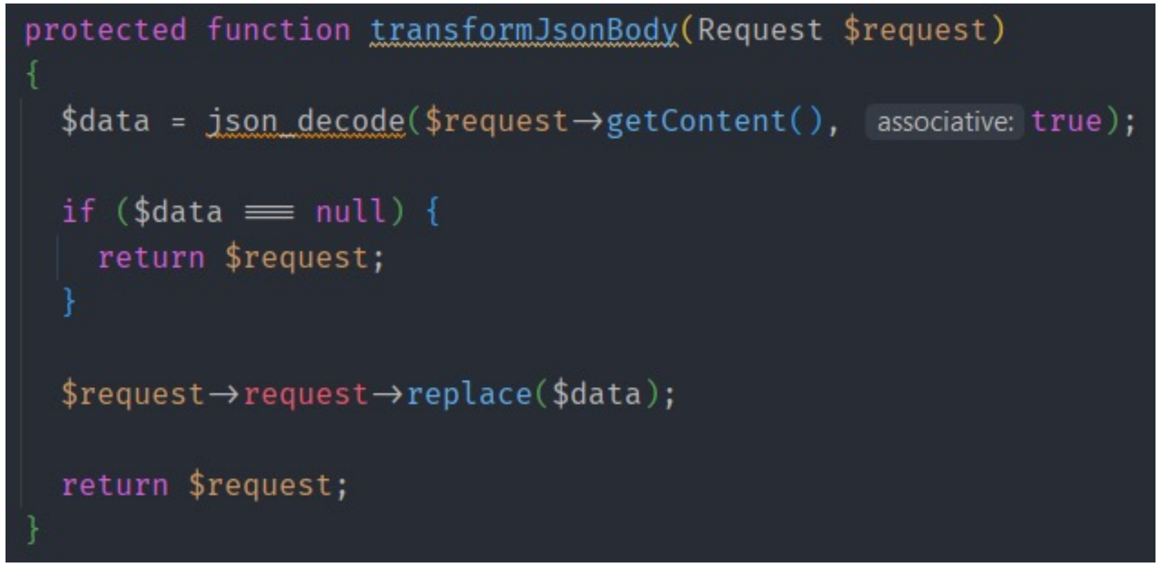
```
[
    {
        "id": "1",
        "make": "Opel",
        "model": "Omega",
        "travelled_distance": "2147483647"
    }
]
```

REST Controllers

- POST method



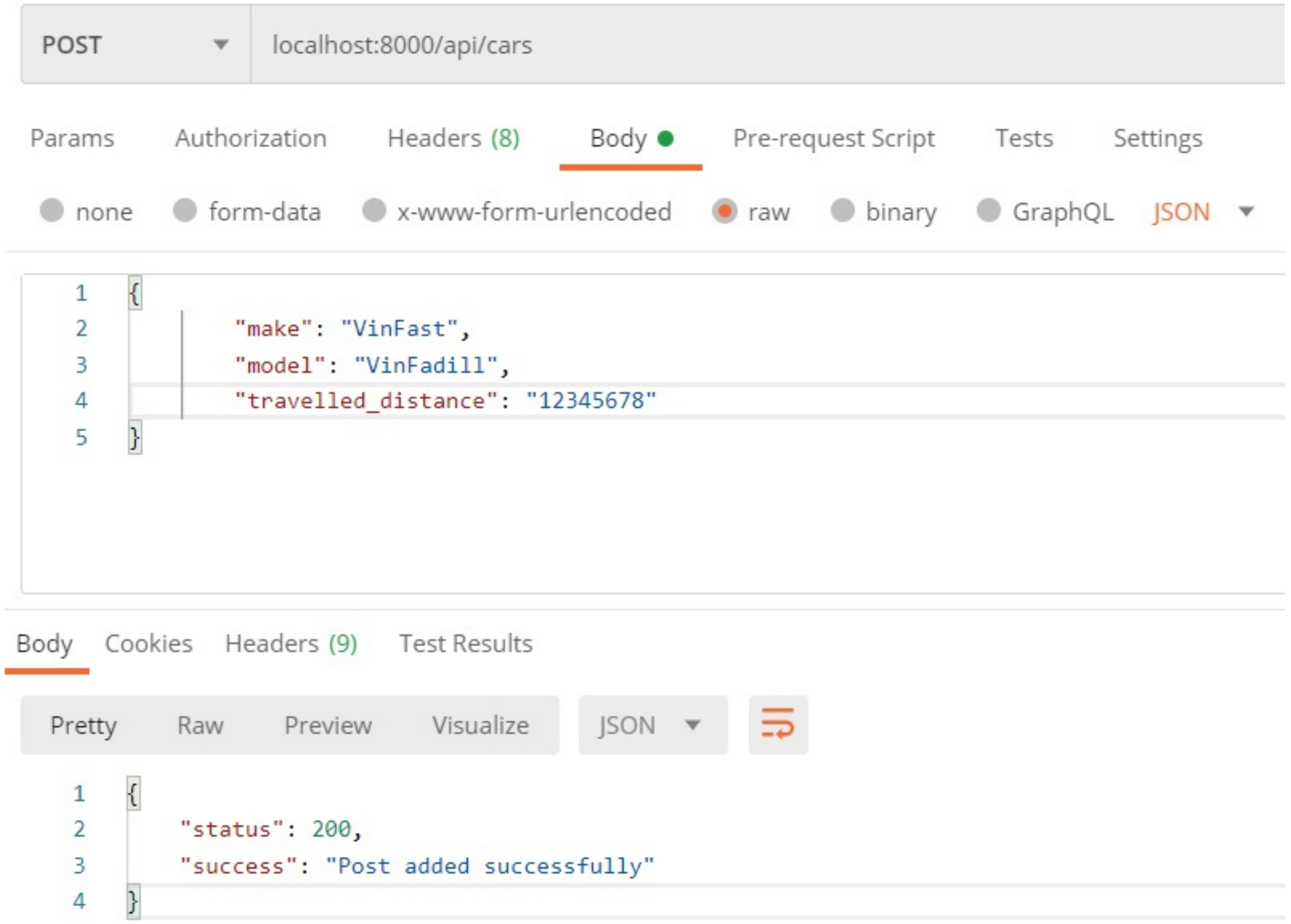
```php
/**
 * @param Request $request
 * @param EntityManagerInterface $entityManager
 * @Route("/cars", name="cars_add", methods={"POST"})
 */
public function addCar(Request $request,
                        EntityManagerInterface $entityManager)
{
    $request = $this->transformJsonBody($request);
    $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);
```



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

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

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

    return $request;
}
```

REST Controllers

- POST method



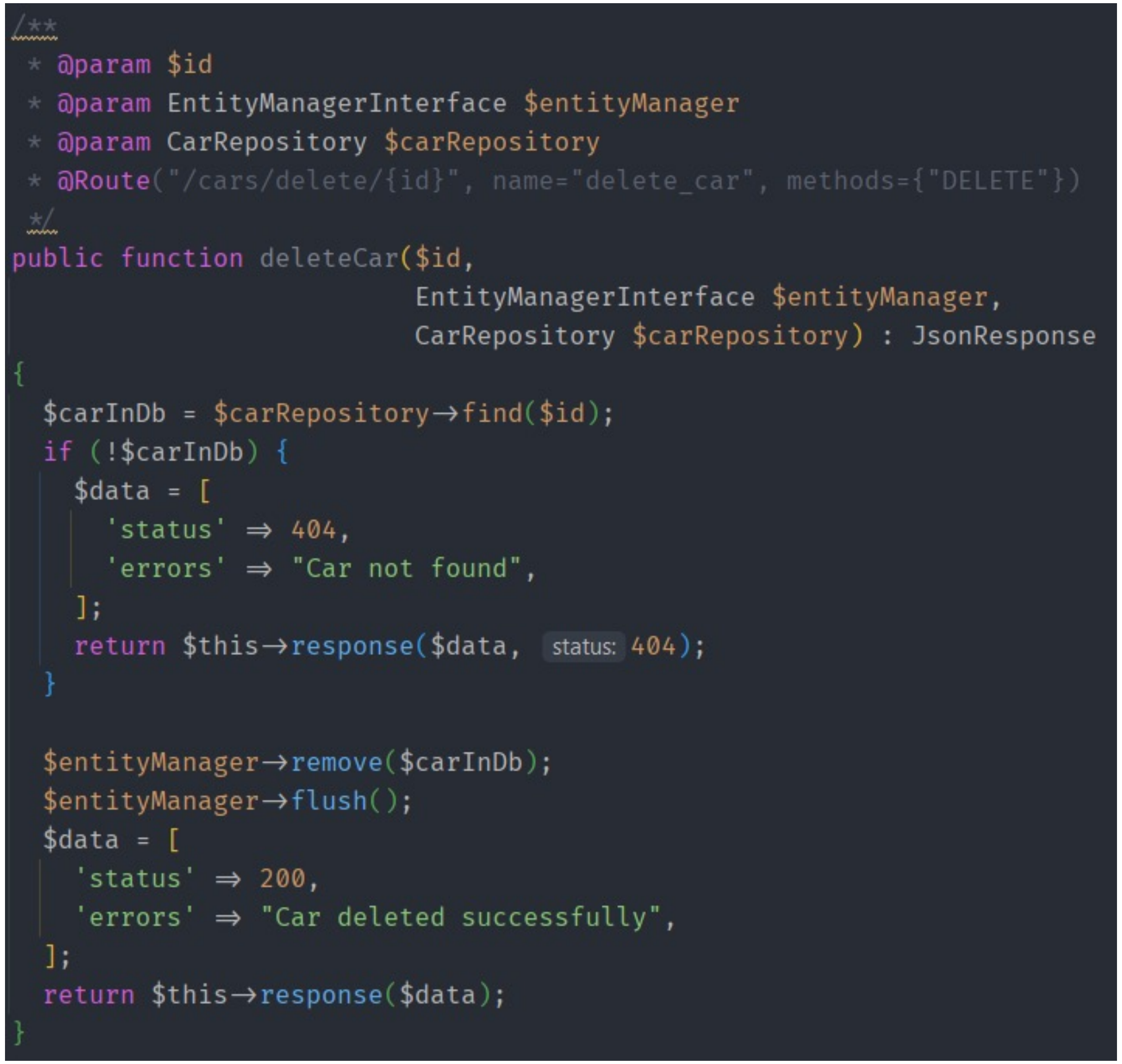
localhost:8000/api/cars

```
{
        "make": "VinFast",
        "model": "VinFadill",
        "travelled_distance": "12345678"
}
```

```
{
    "status": 200,
    "success": "Post added successfully"
}
```

REST Controllers

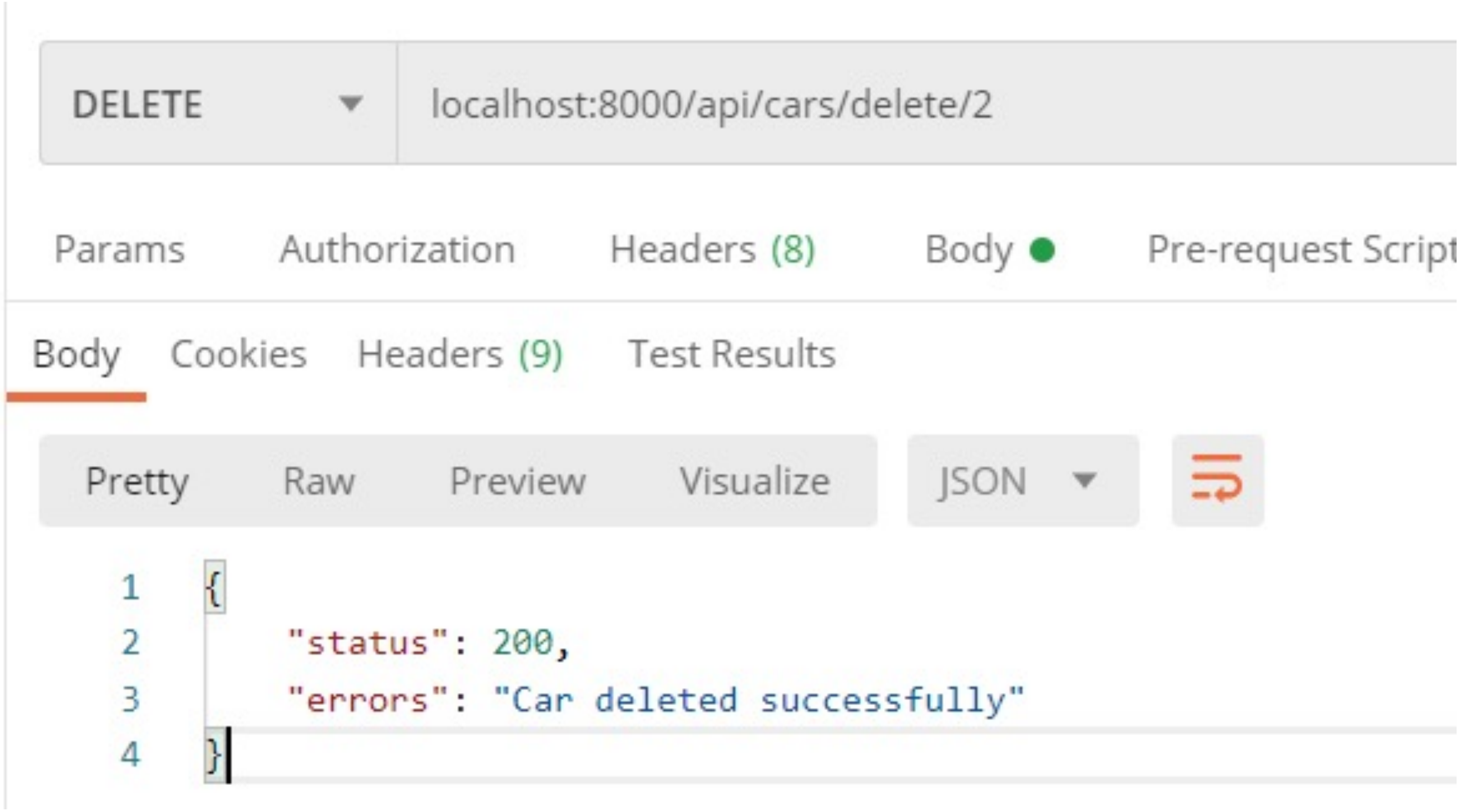
- DELETE method



```
/**
 * @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

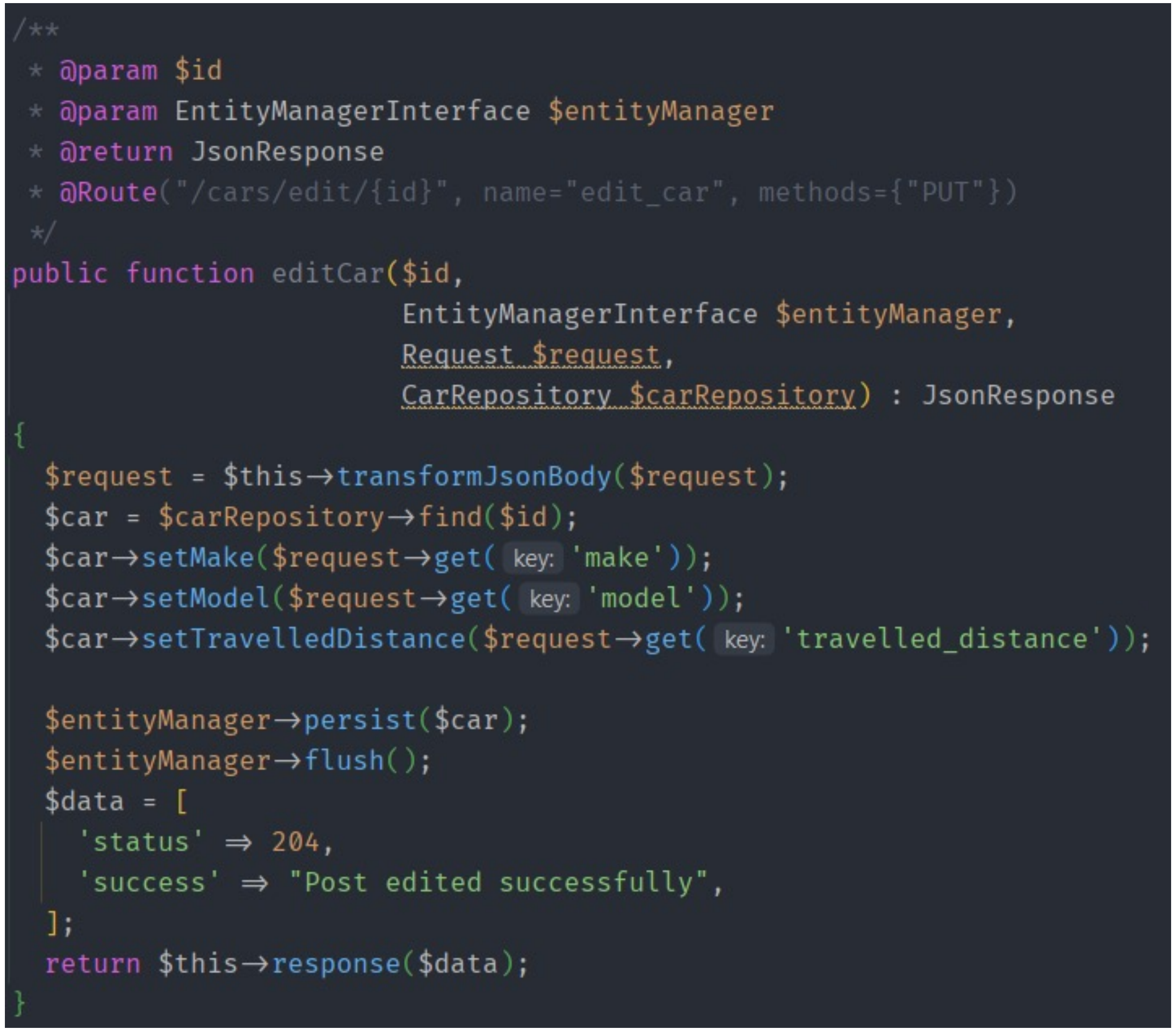


DELETE localhost:8000/api/cars/delete/2

```
{
    "status": 200,
    "errors": "Car deleted successfully"
}
```

REST Controllers

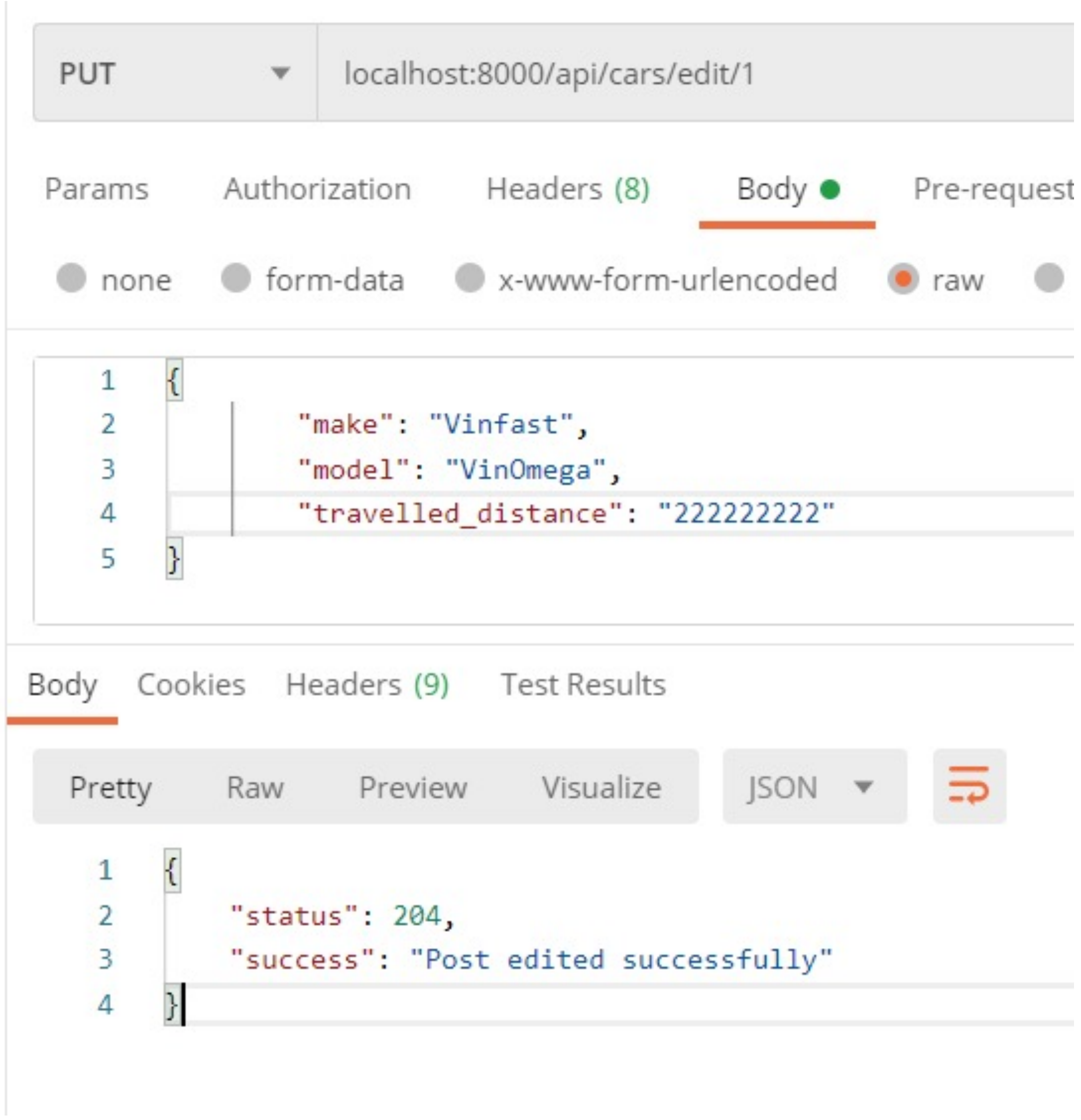
- PUT method



```php
/**
 * @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



PUT | localhost:8000/api/cars/edit/1

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

none | form-data | x-www-form-urlencoded | [✓ raw]

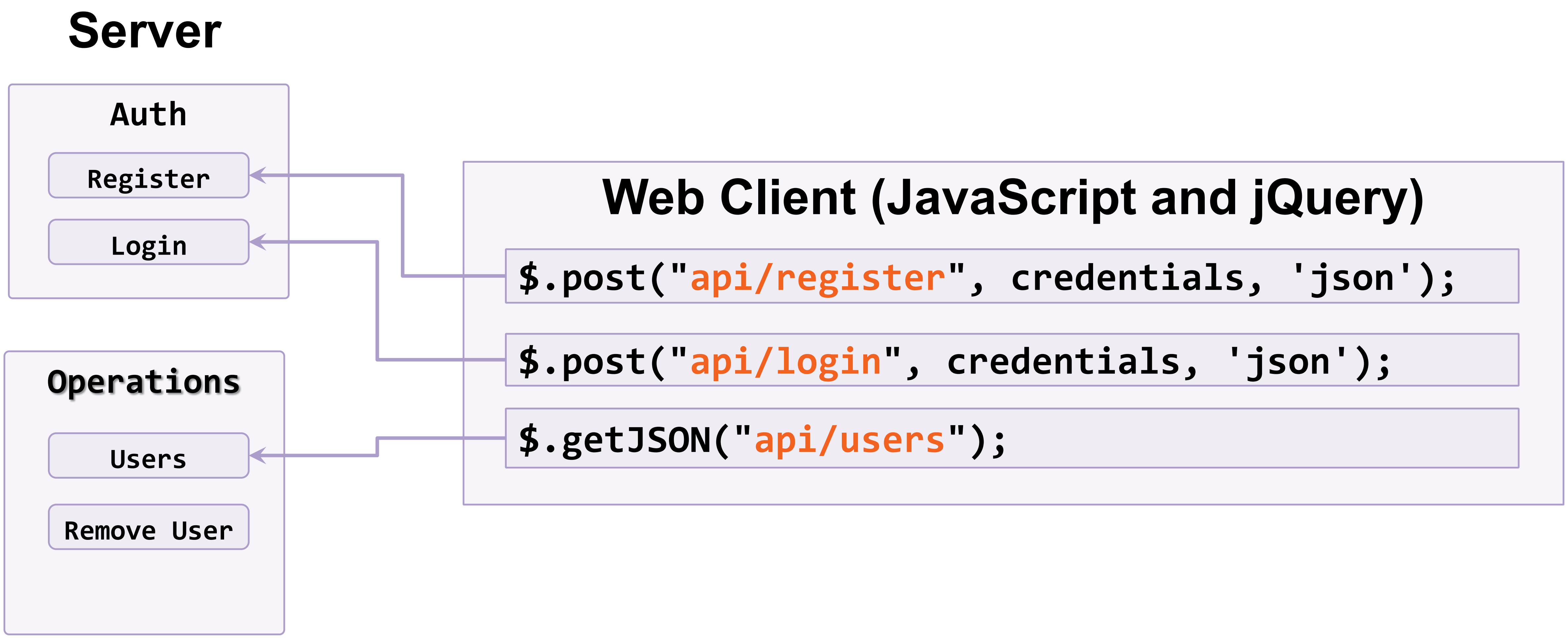
```
{
        "make": "Vinfast",
        "model": "VinOmega",
        "travelled_distance": "222222222"
}
```

Body | Cookies | Headers (9) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "status": 204,
    "success": "Post edited successfully"
}
```

RESTful API – Example



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

JSON

- JSON (JavaScript Object Notation)
  - 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