JSON Patch in Asp.net Core Webapi

Scenario

Consider following Person entity
```
{
  "id": 1,
  "name": "Sam Kwee",
 "email": "skwee357@olddomain.com",
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

If there is a requirement to update email address of the person from "skwee357@olddomain.com" **to** "skwee357@newdomain.com" as shown below

```
{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@newdomain.com",
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

We would generally do an HttpPut operation for such partial updates, which incurs updating the entire Person entity, for a minor change of a property (email in this case) .
The disadvantage of such implementation is usage of unnecessary bandwidth, which will eventually effect the performance of webapi.

How do we avoid this ?

Short answer is JSON Patch .

What is JSON Patch ?

JSON Patch is a format for describing changes to a JSON document. It can be used to avoid sending a whole document when only a part has changed.

When used in combination with the HTTP PATCH method, it allows partial updates for HTTP APIs in a standards compliant way. The patch documents are themselves JSON documents.

For example :

Consider the document below

```
{
  "firstname": "Ajay",
  "lastname": "Nallanagula"
}
```

The patch

```
[
  { "op": "replace", "path": "/firstname", "value": "Ajay Kumar" },
  { "op": "add", "path": "/address", "value": ["epam jvp building hyderabad"] },
  { "op": "remove", "path": "/lastname"}
]
```

The result

{

```
"firstname": "Ajay",
"address":["epam jvp building hyderabad"]
```

}

How it works ?

A JSON Patch document is just a JSON file containing an array of patch operations.

The patch operations supported by JSON Patch are

- “add”,
- “remove”,
- “replace”,
- “move”,
- “copy”
- “test”.

The operations are applied in order: if any of them fail then the whole patch operation should abort.

JSON Pointer

JSON Pointer defines a string format for identifying a specific value within a JSON document. It is used by all operations in JSON Patch to specify the part of the document to operate on.

A JSON Pointer is a string of tokens separated by `/` characters, these tokens either specify keys in objects or indexes into arrays. For example, given the JSON

```
{
  "biscuits": [
    { "name": "Marie Gold" },
    { "name": "Good Day" }
  ]
}
```

ex : {"op":"add" , **"path":"/biscuits/-"** , "value": {"name":"Little Hearts" }}, here the value of path is JSON Pointer.

To point to the array **biscuits** in the JSON document we use **/biscuits.**

To point to **"Good Day"** we use **/biscuits/1/name.**

To point to the root of the document use an **empty string** for the pointer.

What if the key value in JSON document have '/' in its name for example : { "foo/bar~" : "football" } ?

In this case we need to **escape** the characters using **~0 and ~1** ,to form JSON pointer, **/foo~0bar~1**

If you need to refer to the **end of an array** you can **use `-`** instead of an index. For example, to refer to the end of the array of biscuits above you would use `/biscuits/-`. This is useful when you need to insert a value at the end of an array.

Add :

Adds a value to an object or inserts it into an array.

```
Operation : {"op":"add" ,  "path":"/biscuits/-" , "value": {"name":"Little Hearts" }}

Result : {

 "biscuits": [
    { "name": "Marie Gold" },
    { "name": "Good Day" },

    { "name": "Little Hearts" } ]

}
```

In the below mentioned case of an array, the value is inserted before the given index

```
Operation : {"op":"add" ,  "path":"/biscuits/1" , "value": {"name":"Bourbon" }}

Result : {

 "biscuits": [
    { "name": "Marie Gold" },

    { "name": "Bourbon" },  //new Item is added here

    { "name": "Good Day" },

 ]

}
```

Remove :

```
Operation : {"op":"remove" , "path":"/biscuits" }

Result : Will remove the JSON Array biscuits

Operation :  {"op":"remove" , "path":"/biscuits/0" }

Result :

{
  "biscuits": [
   { "name": "Good Day" }
  ]
}
```

Removes the first element of the array at `biscuits` (or just removes the "0" key if `biscuits` is an object)

Replace:

```
Operation { "op": "replace", "path": "/biscuits/0/name", "value": "Britania" }

Result :

{
  "biscuits": [
    { "name": "Britania" },

    { "name": "Good Day" }
  ]
}
```

Replaces a value. Equivalent to a "remove" followed by an "add".

Copy:

```
Operation : { "op": "copy", "from": "/biscuits/0", "path": "/best_biscuit" }
Result : {
  "biscuits": [
    { "name": "Marie Gold" },
    { "name": "Good Day" }
  ],

"best_biscuit" : "Marie Gold"

}
```

Copies a value from one location to another within the JSON document. Both `from` and `path` are JSON Pointers.

Move:

```
Operation : { "op": "move", "from": "/biscuits", "path": "/cookies" }
Result : {
  "biscuits": [
    { "name": "Marie Gold" },
    { "name": "Good Day" }
  ],"cookies" : [

   { "name": "Marie Gold" },
    { "name": "Good Day" }
  ]

}
```

```
Moves a value from one location to the other. Both from and path are JSON Pointers.
```

Test:

```
Operation { "op": "test", "path": "/best_biscuit/name", "value": "Marie Gold" }
```

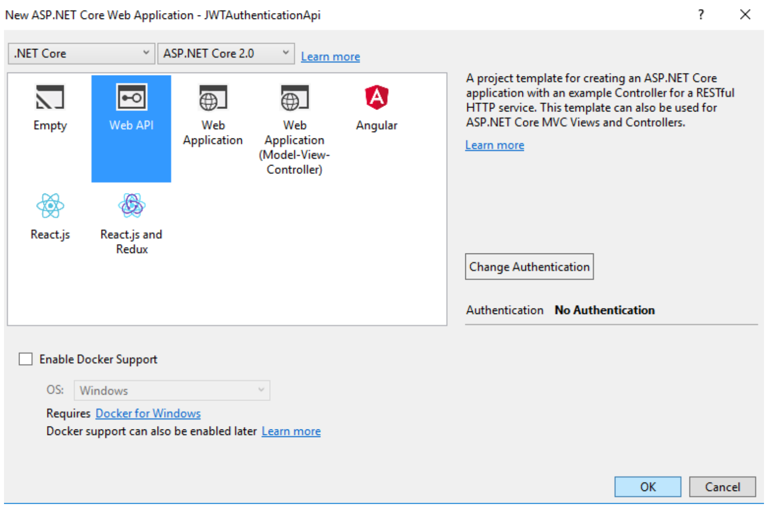
```
Tests that the specified value is set in the document. If the test fails, then the patch as a whole should not
apply.
```

The HTTP PATCH method is atomic and the patch should only be applied if all operations can be safely applied. The `test` operation can offer additional validation to ensure that patch preconditions or postconditions are met. If the test fails the whole patch is discarded. `test` is strictly an equality check.

How do we make use of JSON PATCH in ASP.NET CORE WebApi ?

NOTE: GET, POST , PUT, DELETE are straight forward operations, In this blog lets focus on PATCH operation.

Step 1: Create a new asp.net core project, select **WebApi project template** as shown below


New ASP.NET Core Web Application - JWTAuthenticationApi

.NET Core | ASP.NET Core 2.0

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Authentication **No Authentication**

Enable Docker Support

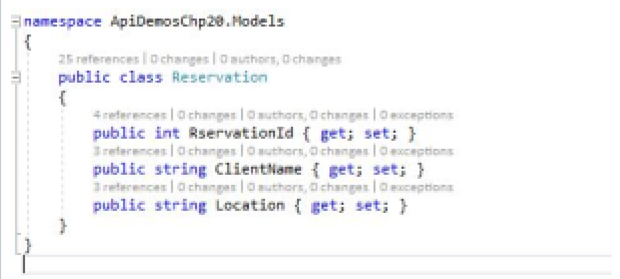
OS: Windows

Requires Docker for Windows

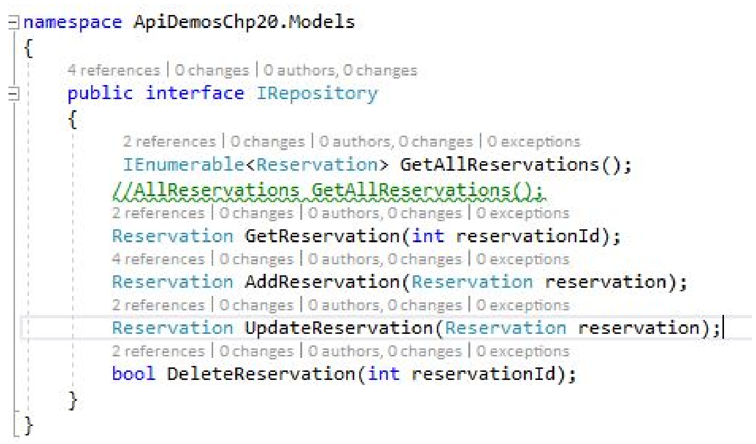
Docker support can also be enabled later Learn more

Step 2: Lets create these model classes, and Interface, data sources class (ideally these things should be retrieved from persistence storage like DB) for demo purpose this should be fine

Reservation Model	IRepository

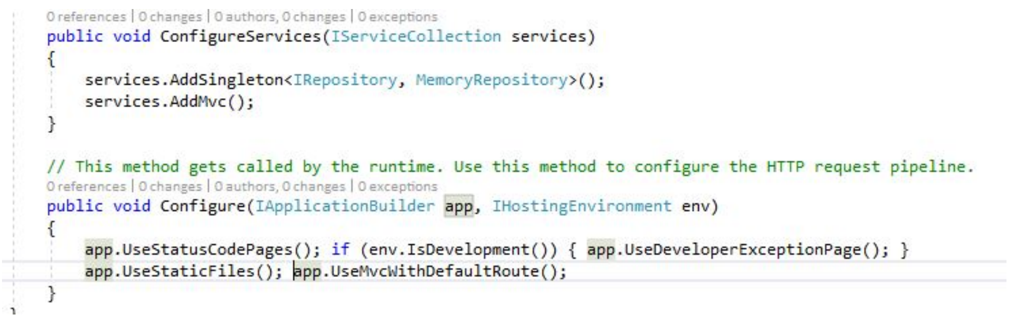


```
namespace ApiDemosChp20.Models
{
    25 references | 0 changes | 0 authors, 0 changes
    public class Reservation
    {
        4 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public int RservationId { get; set; }
        3 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string ClientName { get; set; }
        3 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string Location { get; set; }
    }
}
```



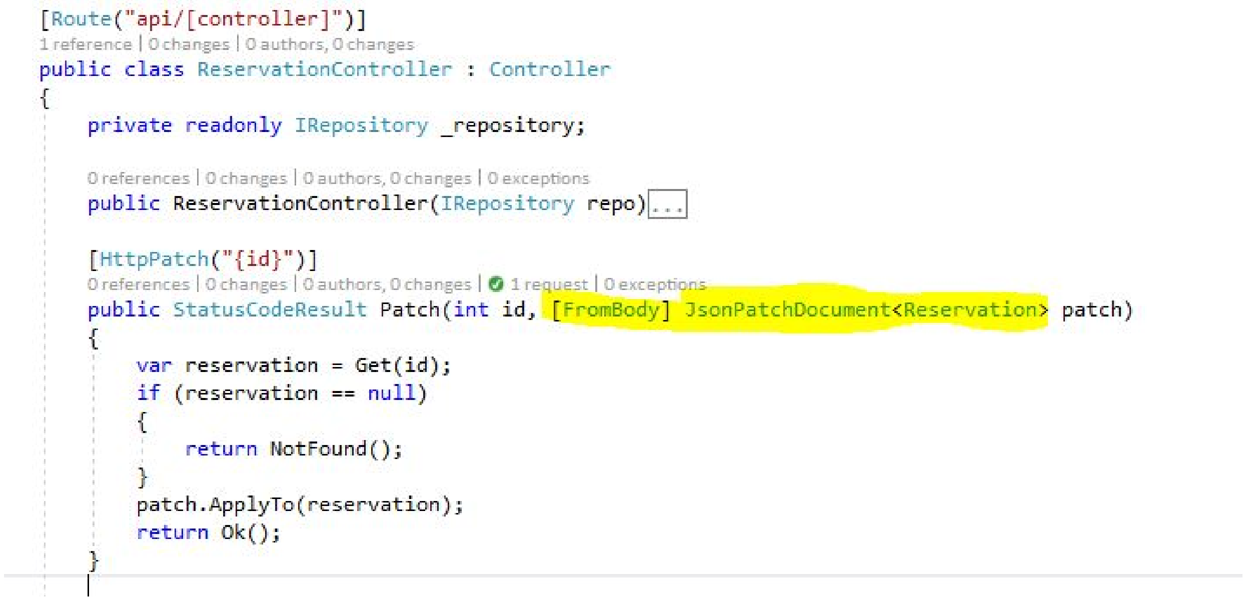
```
namespace ApiDemosChp20.Models
{
    4 references | 0 changes | 0 authors, 0 changes
    public interface IRepository
    {
         2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
         IEnumerable<Reservation> GetAllReservations();
        //AllReservations GetAllReservations();
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        Reservation GetReservation(int reservationId);
        4 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        Reservation AddReservation(Reservation reservation);
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        Reservation UpdateReservation(Reservation reservation);
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        bool DeleteReservation(int reservationId);
    }
}
```

Step 3: Lets Configure Startup.cs, Insert the dependency MemoryResspoitory in ConfigureServices



```
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IRepository, MemoryRepository>();
        services.AddMvc();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseStatusCodePages(); if (env.IsDevelopment()) { app.UseDeveloperExceptionPage(); }
        app.UseStaticFiles(); app.UseMvcWithDefaultRoute();
    }
```

Step 4: Now lets implement WebApi methods , The main focus here is on HTTP PATCH operation , rest of the operations can be seen in source code uploaded on github .



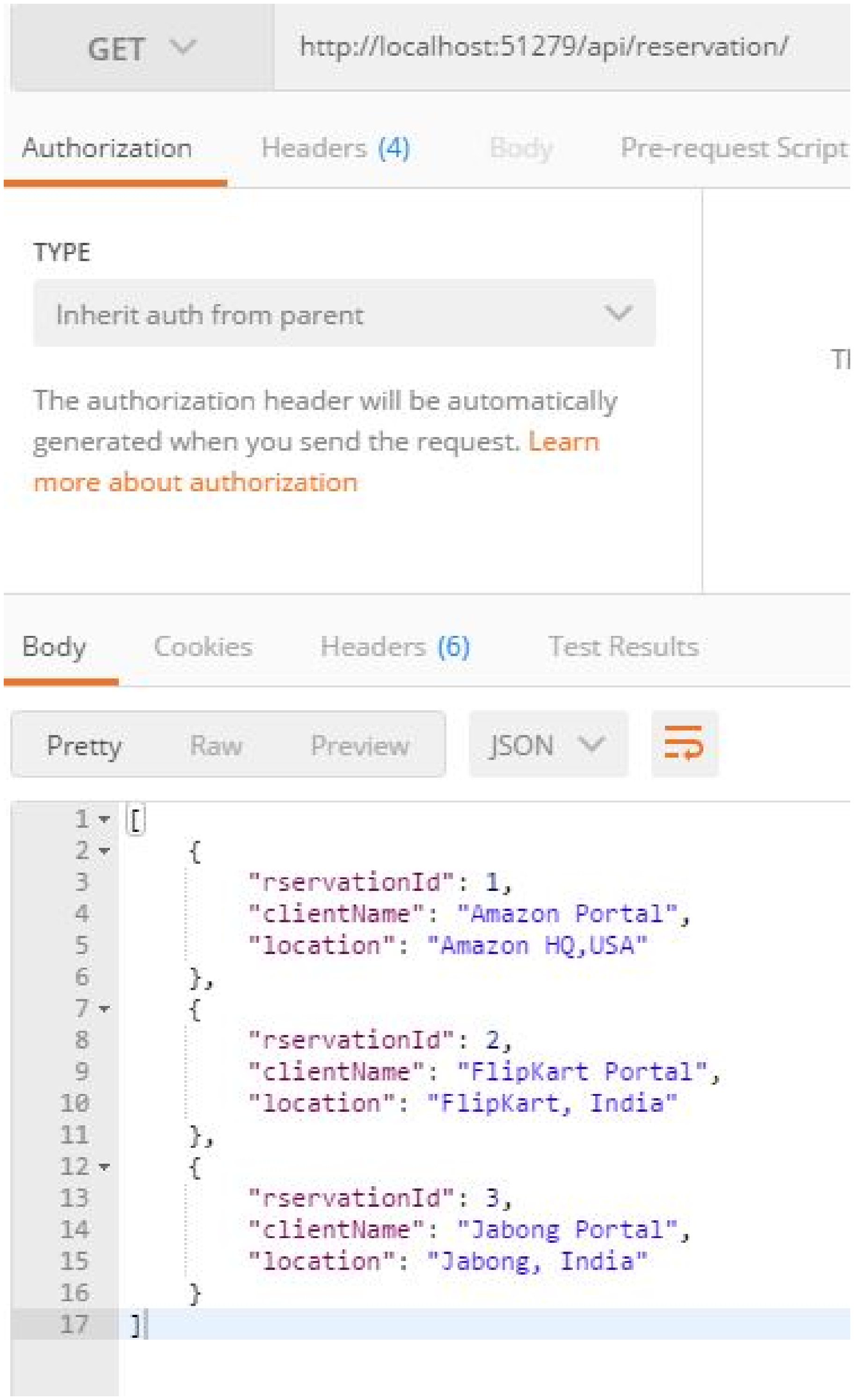
```
[Route("api/[controller]")]
1 reference | 0 changes | 0 authors, 0 changes
public class ReservationController : Controller
{
    private readonly IRepository _repository;

    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public ReservationController(IRepository repo)...

    [HttpPatch("{id}")]
    0 references | 0 changes | 0 authors, 0 changes | 1 request | 0 exceptions
    public StatusCodeResult Patch(int id, [FromBody] JsonPatchDocument<Reservation> patch)
    {
        var reservation = Get(id);
        if (reservation == null)
        {
            return NotFound();
        }
        patch.ApplyTo(reservation);
        return Ok();
    }
```

Is this code working ??

Lets test that, I am using postman here , alternatively you can use fiddler, curl and so on.

Get Url : http://localhost:51279/api/reservation/ (the port will be different if you are trying in your local box)

Result :

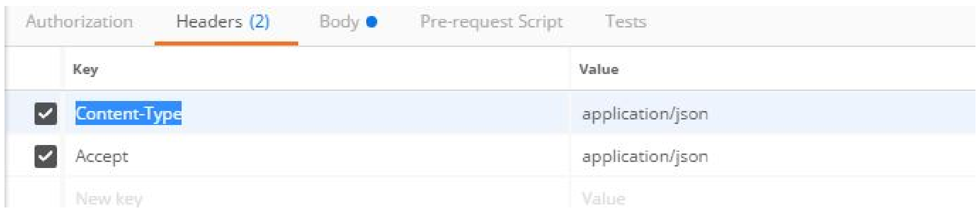

GET http://localhost:51279/api/reservation/

The authorization header will be automatically generated when you send the request. Learn more about authorization

```
[
    {
        "rservationId": 1,
        "clientName": "Amazon Portal",
        "location": "Amazon HQ,USA"
    },
    {
        "rservationId": 2,
        "clientName": "FlipKart Portal",
        "location": "FlipKart, India"
    },
    {
        "rservationId": 3,
        "clientName": "Jabong Portal",
        "location": "Jabong, India"
    }
]
```

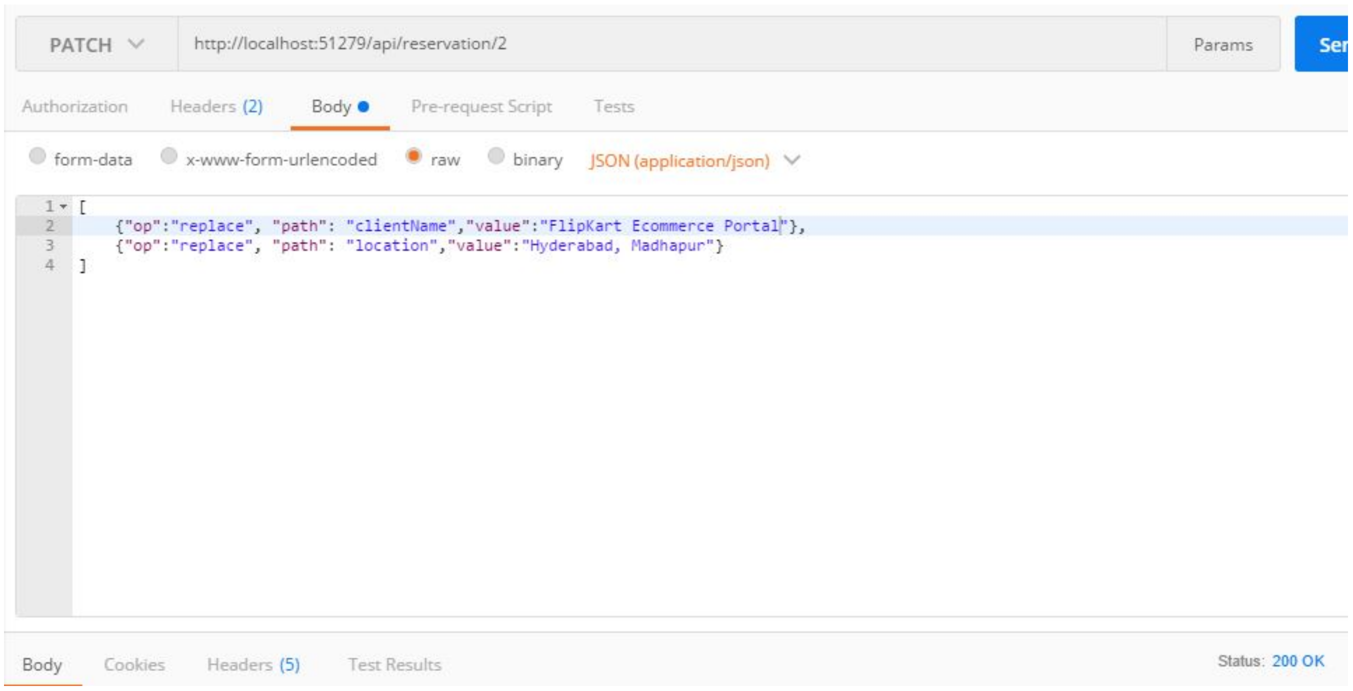
Patch Url : http://localhost:51279/api/reservation/2

Headers



Key	Value
✓ Content-Type	application/json
✓ Accept	application/json
New key	Value

Result :



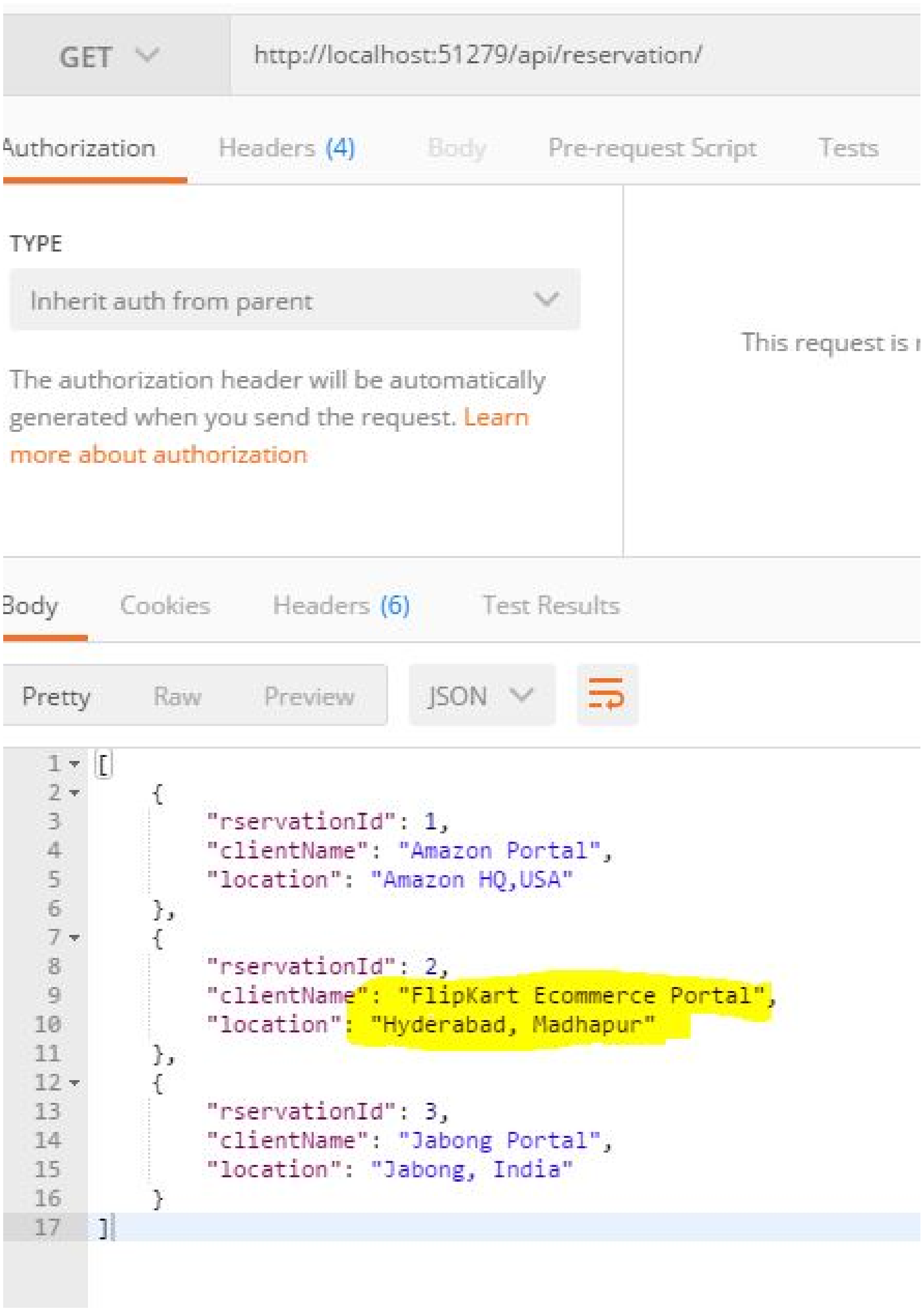
http://localhost:51279/api/reservation/2

```
[
    {"op":"replace", "path": "clientName","value":"FlipKart Ecommerce Portal"},
    {"op":"replace", "path": "location","value":"Hyderabad, Madhapur"}
]
```

Status: 200 OK

Issue a get request again and see the partial updates:

Get Url : http://localhost:51279/api/reservation/



http://localhost:51279/api/reservation/

TYPE

Inherit auth from parent

The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is

```
[
    {
        "rservationId": 1,
        "clientName": "Amazon Portal",
        "location": "Amazon HQ,USA"
    },
    {
        "rservationId": 2,
        "clientName": "FlipKart Ecommerce Portal",
        "location": "Hyderabad, Madhapur"
    },
    {
        "rservationId": 3,
        "clientName": "Jabong Portal",
        "location": "Jabong, India"
    }
]
```

You can see the source code here .