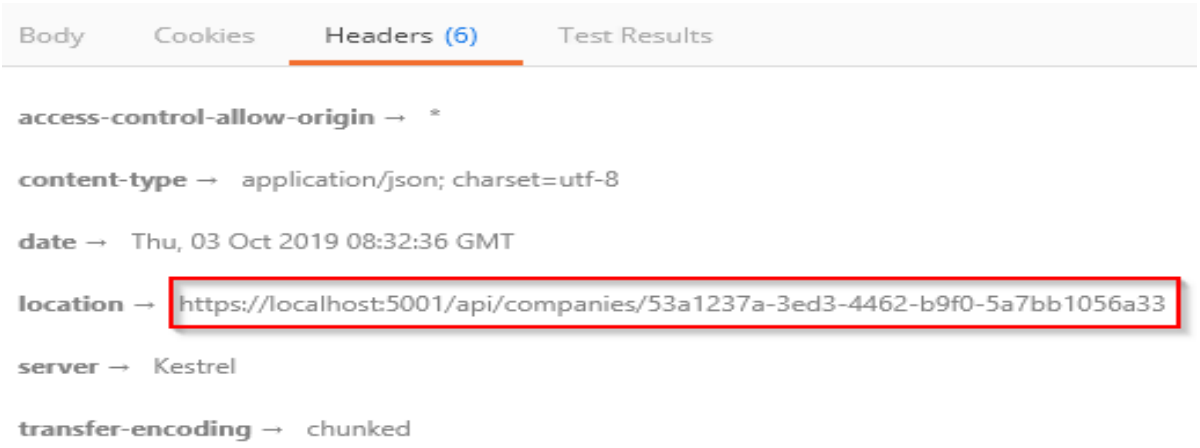
If we take a look at the headers part of our response, we are going to see a link to retrieve the created company:



Finally, from the previous example, we can confirm that the POST method is neither safe nor idempotent. We saw that when we send the POST request, it is going to create a new resource in the database — thus changing the resource representation. Furthermore, if we try to send this request a couple of times, we will get a new object for every request (it will have a different Id for sure).

Let's continue with child resources creation.

Creating a Child Resource

While creating our company, we created the DTO object required for the CreateCompany action. So, for employee creation, we are going to do the same thing:

```
public class EmployeeForCreationDto
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Position { get; set; }
}
```

We don't have the `Id` property because, we are going to create that Id on the server side. But additionaly, we don't have the `CompanyId` because

we accept that parameter through the route:

```
[Route("api/companies/{companyId}/employees")]
```

The next step is to modify the **IEmployeeRepository** interface:

```
public interface IEmployeeRepository
{
    IEnumerable<Employee> GetEmployees(Guid companyId, bool trackChanges);
    Employee GetEmployee(Guid companyId, Guid id, bool trackChanges);
    void CreateEmployeeForCompany(Guid companyId, Employee employee);
}
```

Of course, we have to implement this interface:

```
public void CreateEmployeeForCompany(Guid companyId, Employee employee)
{
    employee.CompanyId = companyId;
    Create(employee);
}
```

Because we are going to accept the employee DTO object in our action, but we also have to send an employee object to this repository method, we have to create an additional mapping rule in the **MappingProfile** class:

```
CreateMap<EmployeeForCreationDto, Employee>();
```

Now, we can add a new action in the EmployeesController:

```
[HttpPost]
public IActionResult CreateEmployeeForCompany(Guid companyId, [FromBody]
EmployeeForCreationDto employee)
{
    if(employee == null)
    {
        _logger.LogError("EmployeeForCreationDto object sent from client is null.");
        return BadRequest("EmployeeForCreationDto object is null");
    }

    var company = _repository.Company.GetCompany(companyId, trackChanges: false);
    if(company == null)
    {
        _logger.LogInfo($"Company with id: {companyId} doesn't exist in the
database.");
        return NotFound();
    }

    var employeeEntity = _mapper.Map<Employee>(employee);

    _repository.Employee.CreateEmployeeForCompany(companyId, employeeEntity);
    _repository.Save();
```

```
    var employeeToReturn = _mapper.Map<EmployeeDto>(employeeEntity);

    return CreatedAtRoute("GetEmployeeForCompany", new { companyId, id =
employeeToReturn.Id }, employeeToReturn);
}
```

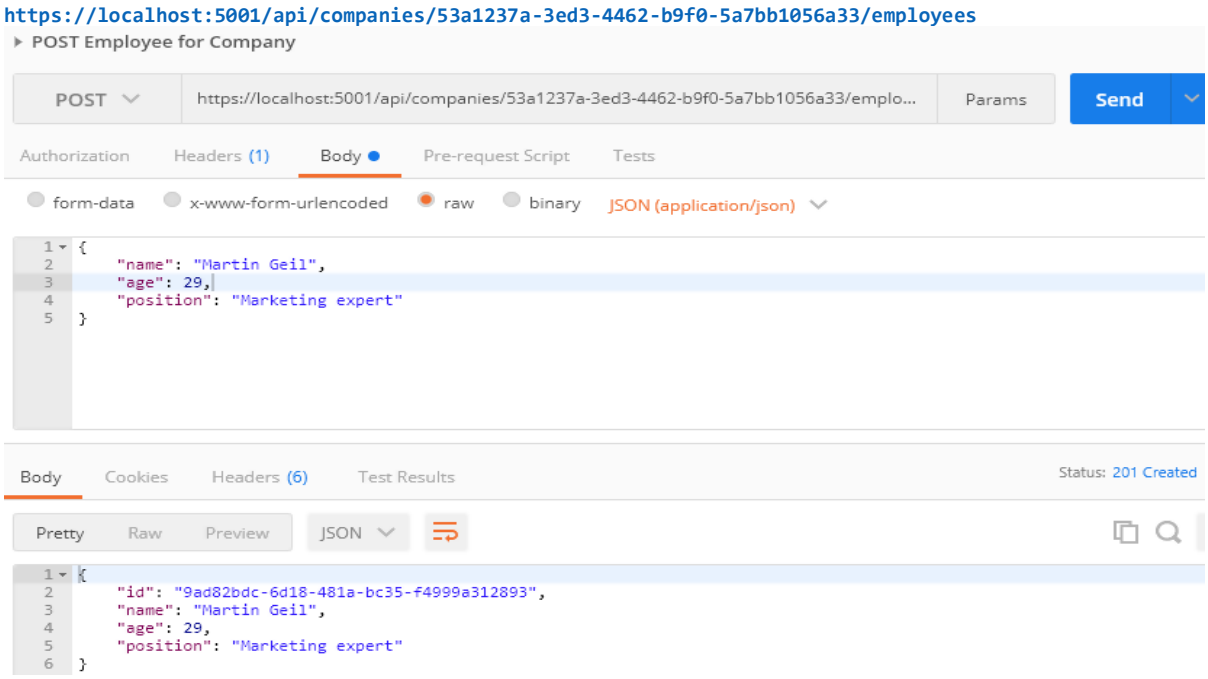
There are some differences in this code compared to the `CreateCompany` action. The first is that we have to check whether that company exists in the database because there is no point in creating an employee for a company that does not exist.

The second difference is the return statement, which now has two parameters for the anonymous object.

For this to work, we have to modify the HTTP attribute above the GetEmployeeForCompany action:

```
[HttpGet("{id}", Name = "GetEmployeeForCompany")]
```

Let's give this a try:



Excellent. A new employee was created.