REST API based Employee Management App

General Info

The application is secured and access is restricted for some resources. Some endpoints are not exposed to the public by default. You must login with your user account and have required permissions to access secured endpoints. The permission to access these endpoints is provided by means of ROLES which are assigned to users. That means even if the user is authenticated i.e. the user is logged in with username and password, he will not be able to access protected resources if he doesn't have permission, the role or authority needed.

Software components

The application has been built using Spring Boot 2.7.1 with the following Maven dependencies.

- Spring Web
- Spring Data JPA
- Spring Security
- H2 Database

Java Version: 17

Application endpoints for different operations

Assumptions

Hostname: localhost

Port: 8086

Roles/Authorities: USER, ADMIN, GUEST

Public/Guest endpoints

http://localhost:8086/api/roles/list // view all roles

http://localhost:8086/api/roles/add // add a role

http://localhost:8086/api/users/list // view all users

http://localhost:8086/api/users/add // add a user

All operations are permitted. Roles and Users can be created, viewed etc.

http://localhost:8086/h2-console // h2 database access portal

User endpoints

A user can access all public endpoints. In addition to that he can access the following endpoints.

http://localhost:8086/api/employees/list // view all employees

http://localhost:8086/api/employees/get/1 // view an employee by id

http://localhost:8086/api/employees/update // update an employee's data

http://localhost:8086/api/employees/search/gl // search employees by firstName: gl

http://localhost:8086/api/employees/sort?order=asc // sort employees in ASC or DSC order

Admin endpoints

An admin can access all public endpoints and users' specific endpoints. In addition to that he can access the following endpoints.

http://localhost:8086/api/employees/add // add an employee

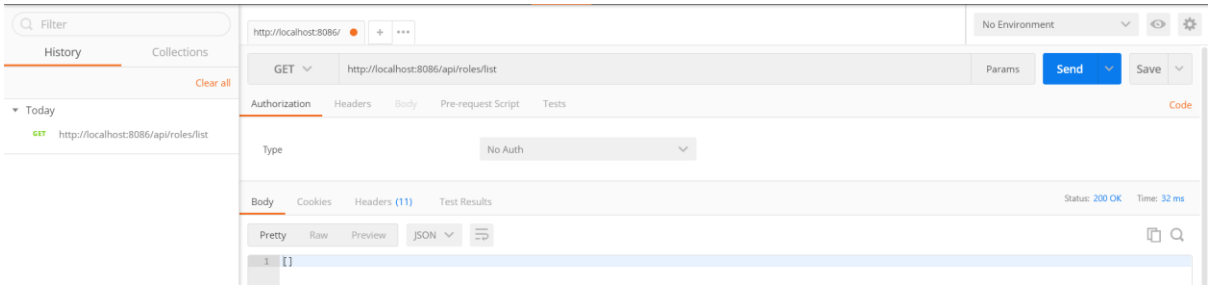
http://localhost:8086/api/employees/delete/1 // delete an employee by id

Screenshots: Application Testing

Creating a role

Let us see the list of available roles first.

http://localhost:8086/api/roles/list (GET request)

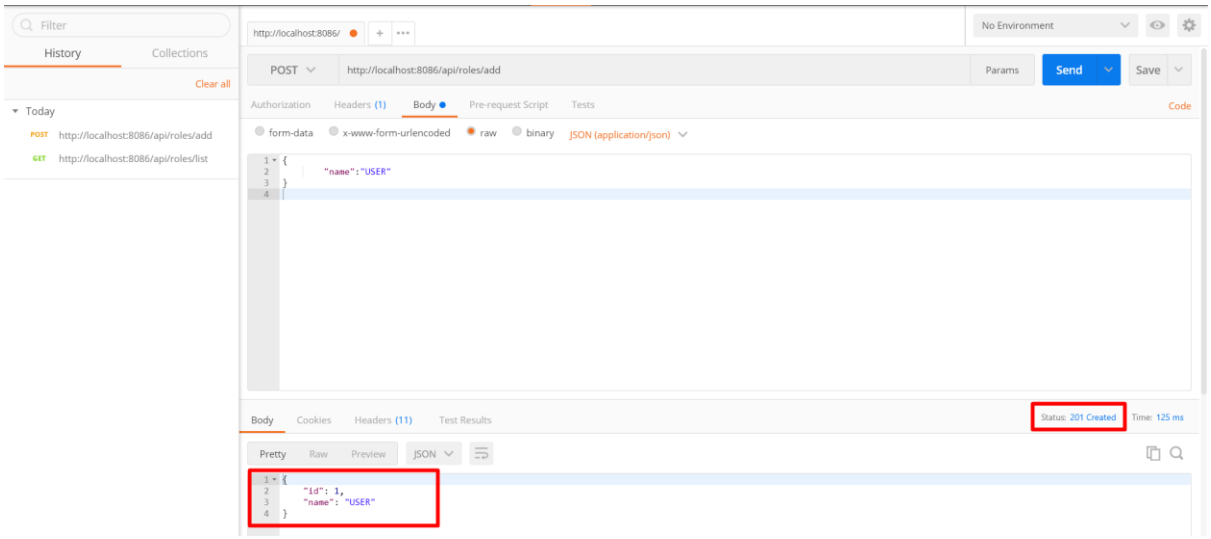


No roles as of now.

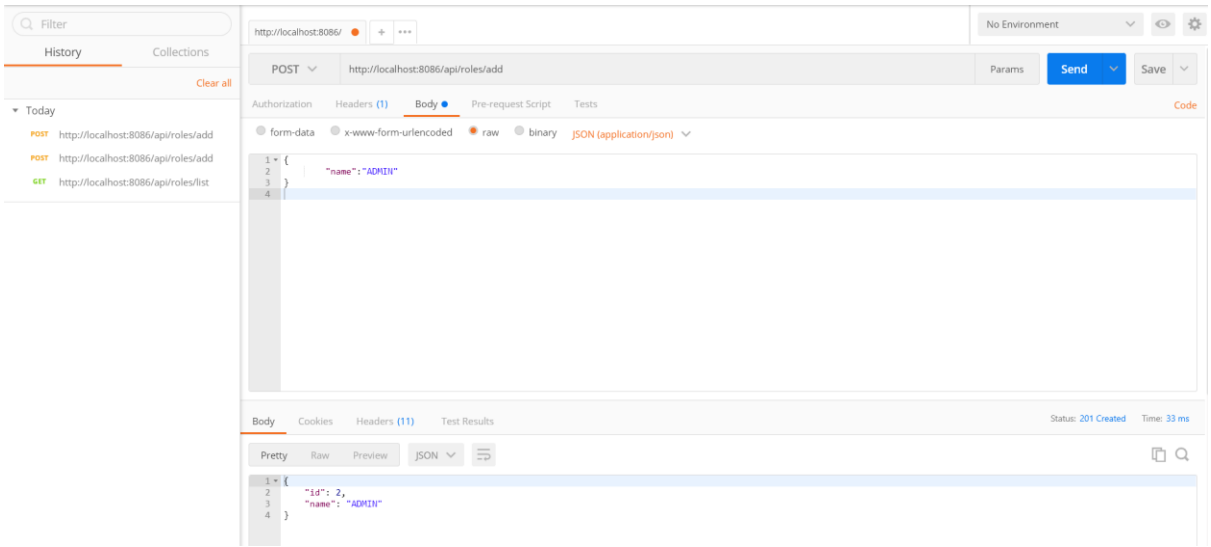
Create three roles: USER, ADMIN, GUEST

http://localhost:8086/api/roles/add (POST request)

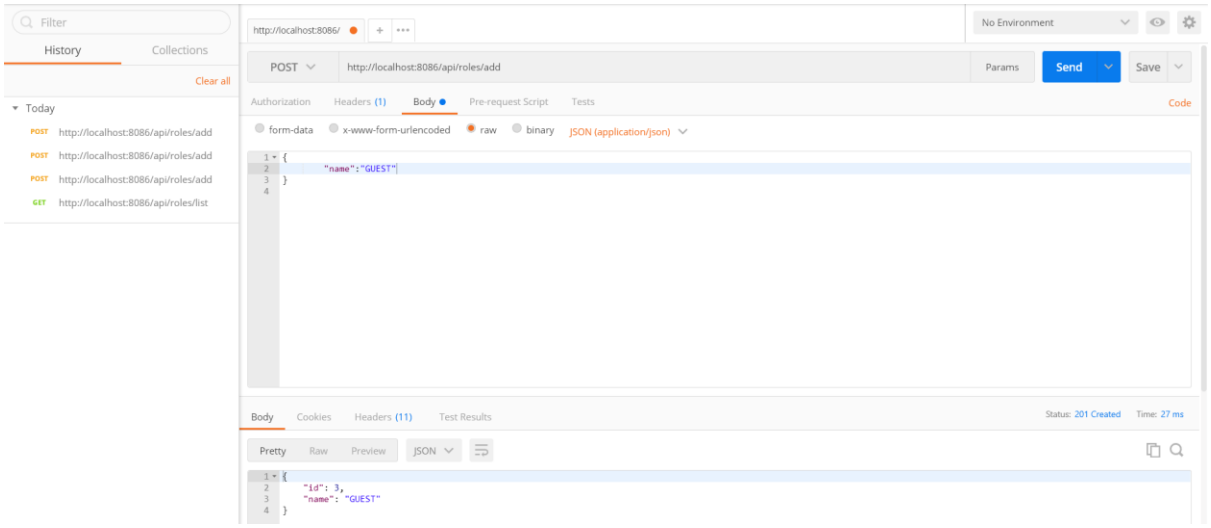
{
"name":"USER"
}
	{
	"name":"ADMIN"
	}
		{
		"name":"GUEST"
		}



Created a role USER with id = 1

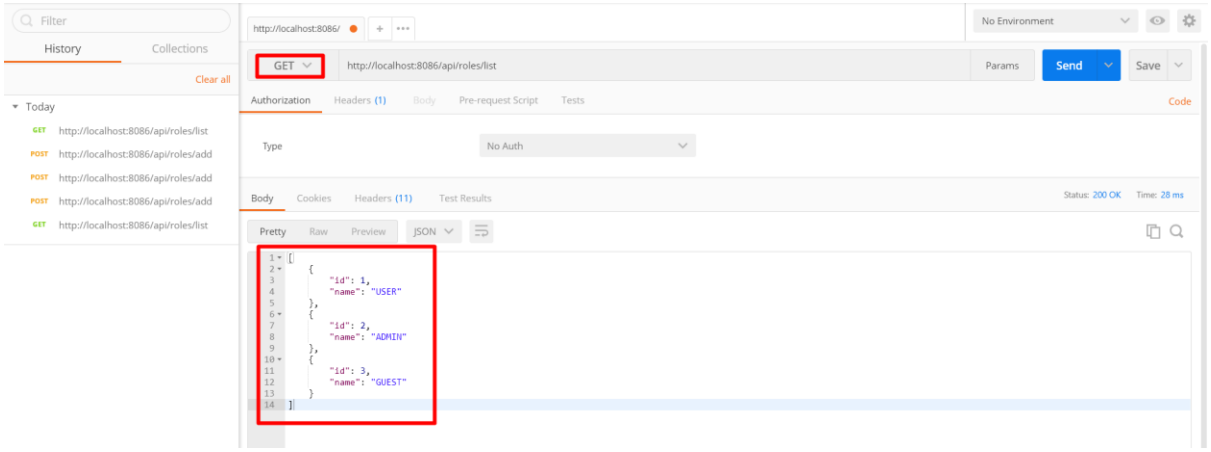


Created a role ADMIN with id = 2



Created a role GUEST with id = 3

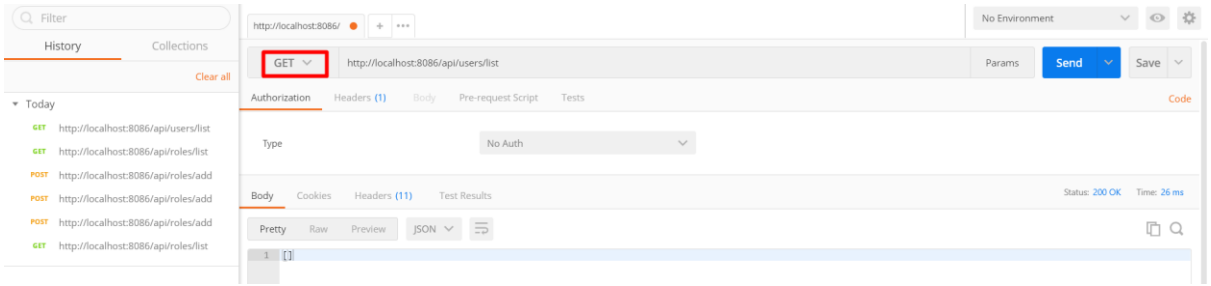
See the list of roles.



Creating a user

Let us see the list of users.

http://localhost:8086/api/users/list (GET request)

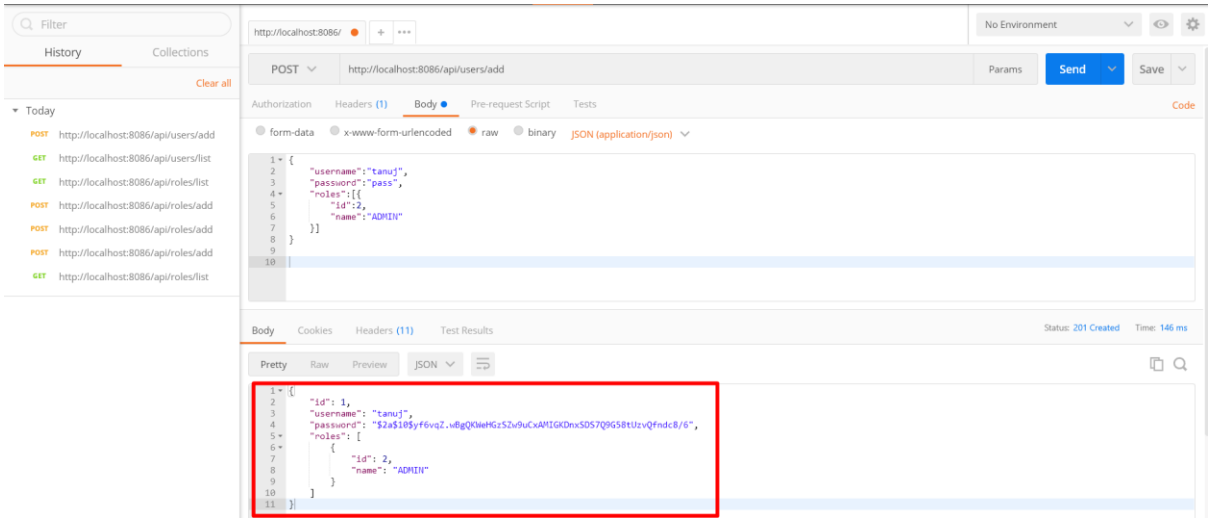


No user as of now.

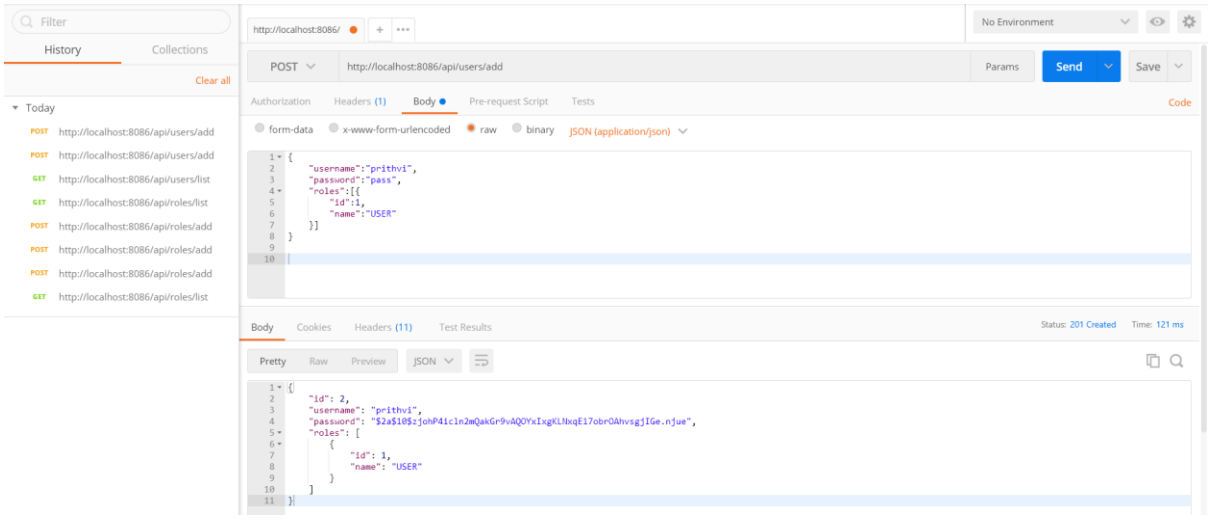
Create three users: tanuj (admin), prithvi(user), homi(guest)

{
"username":"tanuj",
"password":"pass",
"roles":[{
"id":2,
"name":"ADMIN"
}]
}
	{
	"username":"prithvi",
	"password":"pass",
	"roles":[{
	"id":1,
	"name":"USER"
	}]
	}
		{
		"username":"homi",
		"password":"pass",
		"roles":[{
		"id":3,
		"name":"GUEST"
		}]
		}

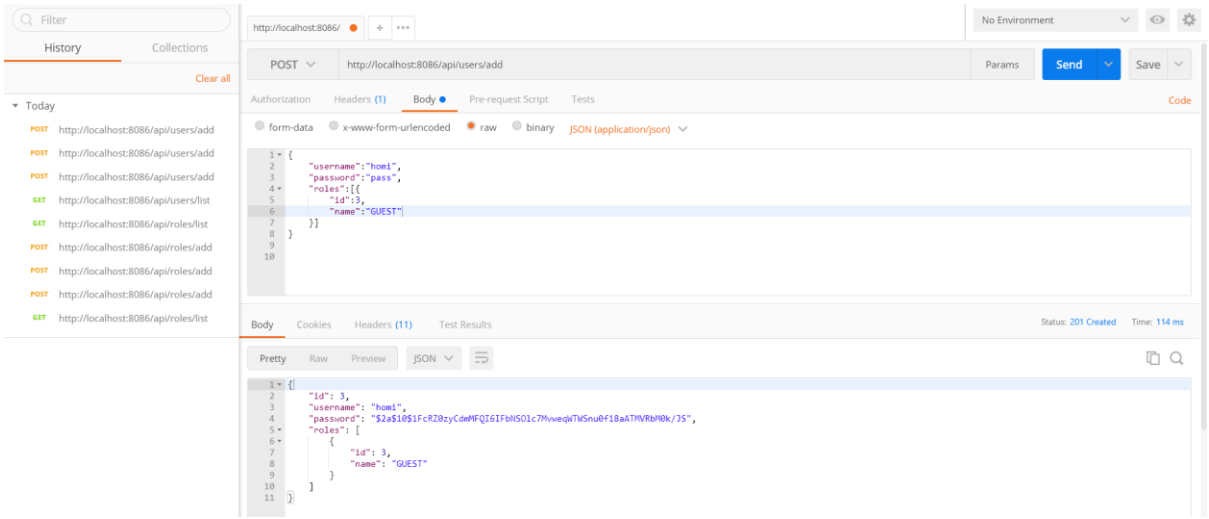
http://localhost:8086/api/users/add (POST request)



Created user *tanuj* with id = 1

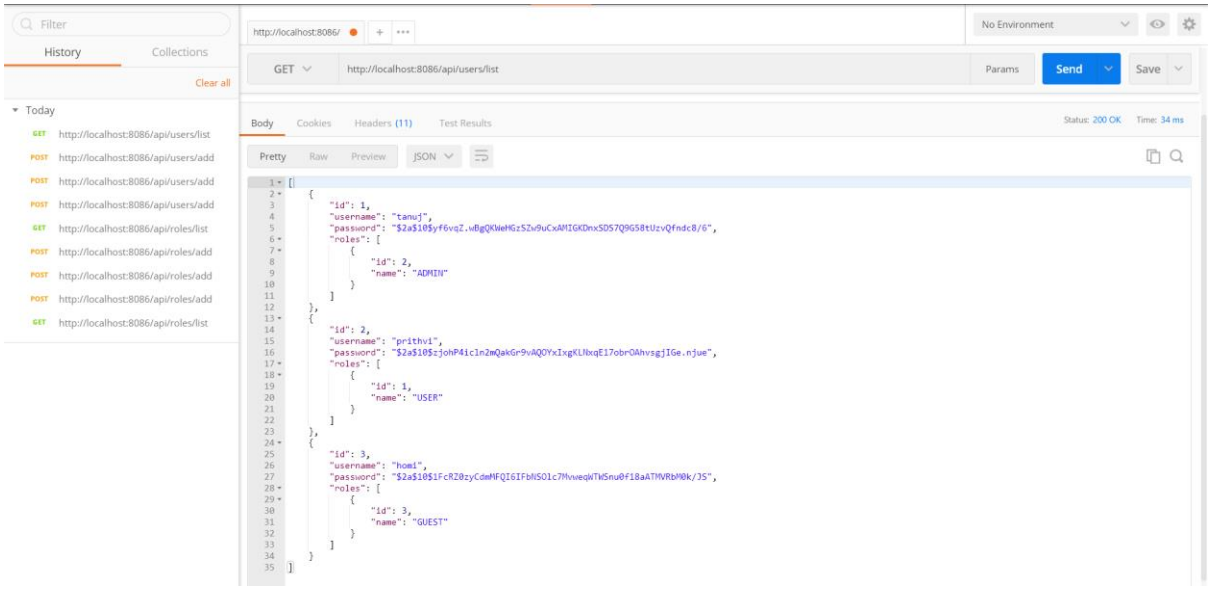


Created user *prithvi* with id = 2



Created user *homi* with id = 3

See the list of users again.

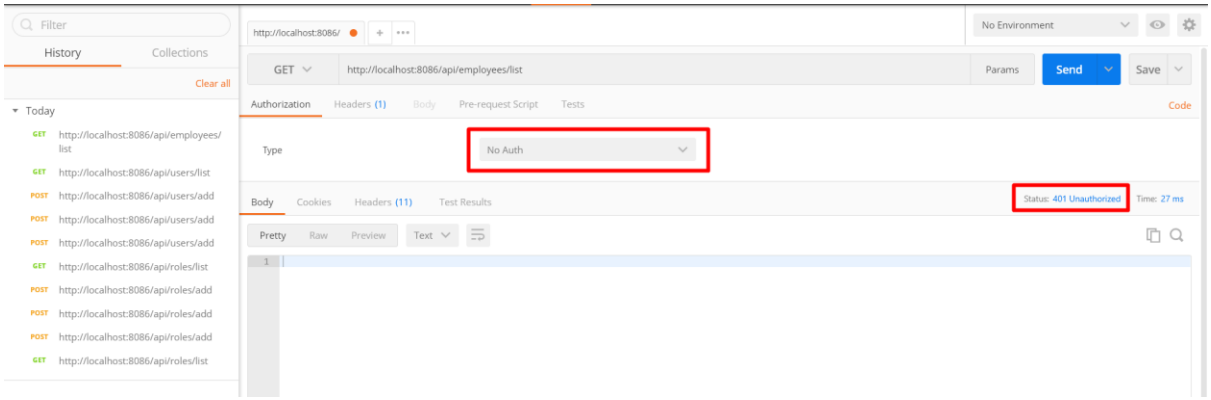


Three users with different authorities are available now.

Viewing employee list

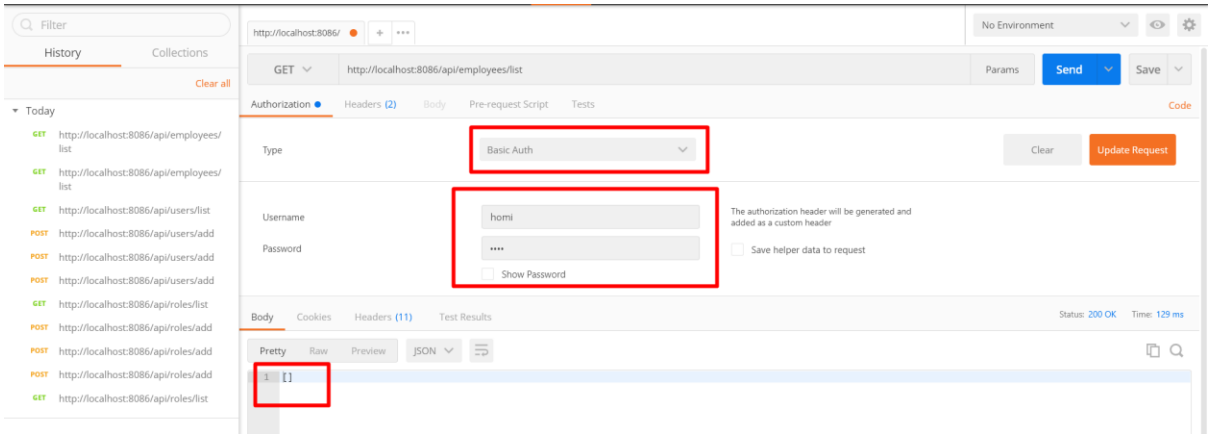
http://localhost:8086/api/employees/list (GET request)

Try to access with normal user without any authentication.



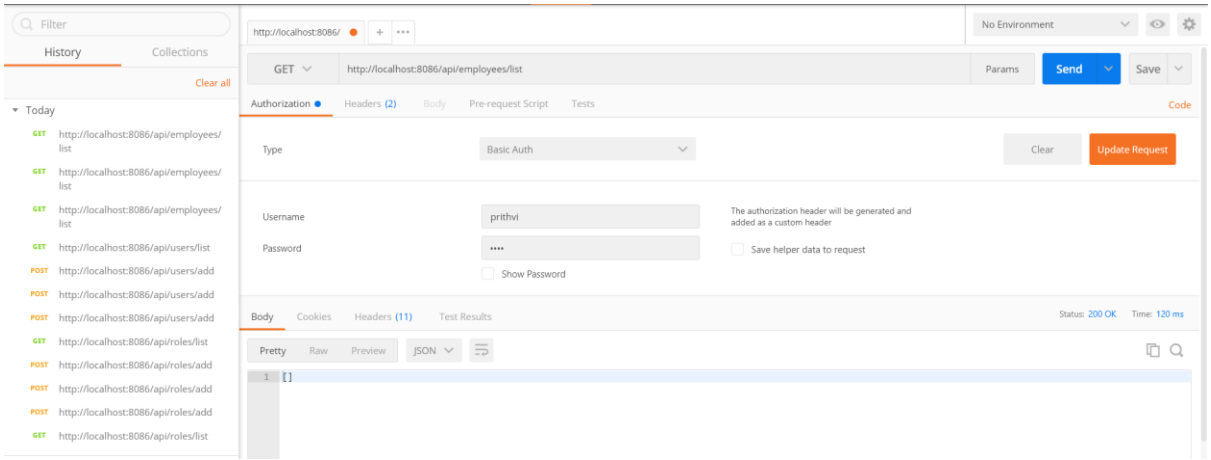
As we can see, there is nothing in the response section of postman app. It is because this endpoint is protected, it means only authenticated users are allowed to see the screen if they have permission(role) to access it.

Login with user homi(guest).



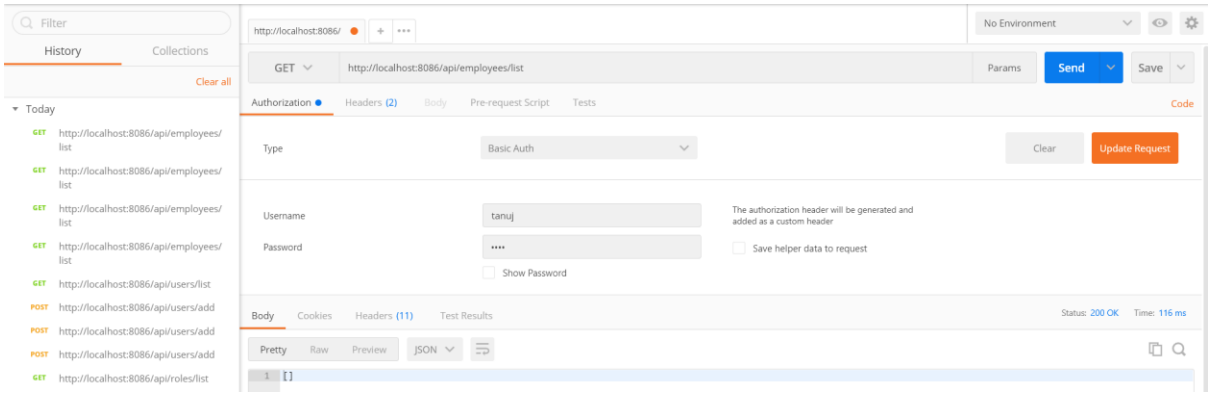
homi can see the list. We can see there is no employee record as of now.

Login with user prithvi(user).



He can also access the employee list.

Login with user tanuj(admin).



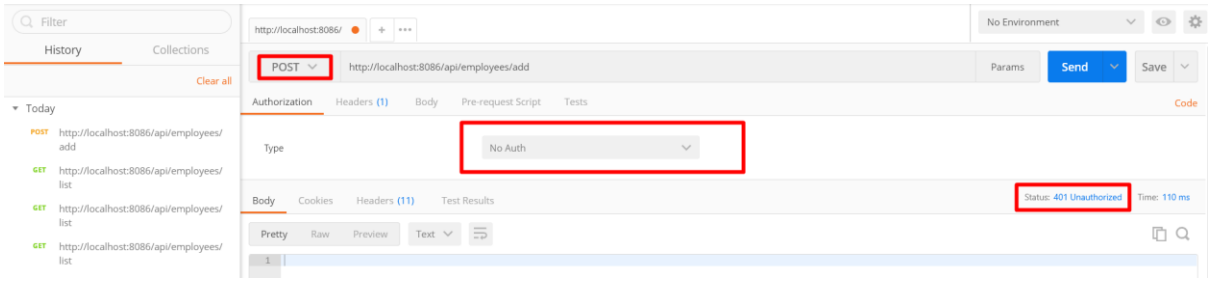
The employee list is visible to user tanuj also.

It is the expected behavior of the application that any authenticated user can see employee list but not unauthenticated visitors.

Adding an employee

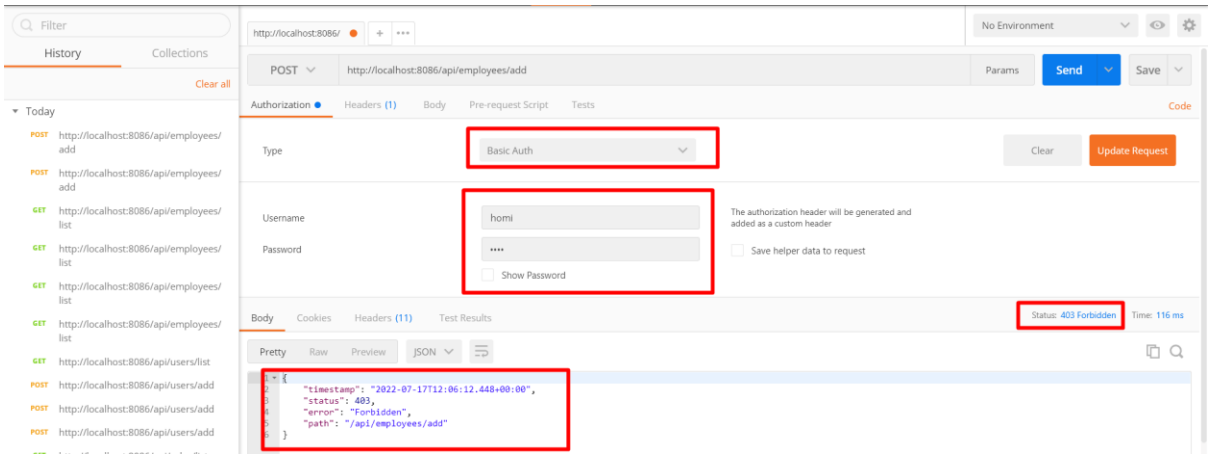
http://localhost:8086/api/employees/add (POST request)

Unauthenticated user tries to add an employee.



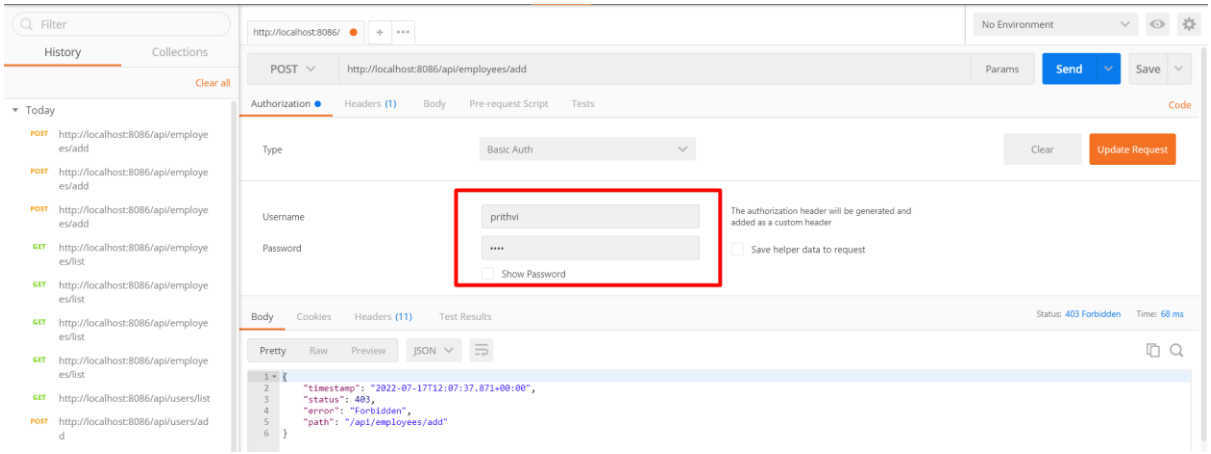
He cannot access the endpoint.

User home(guest) tries to add an employee.



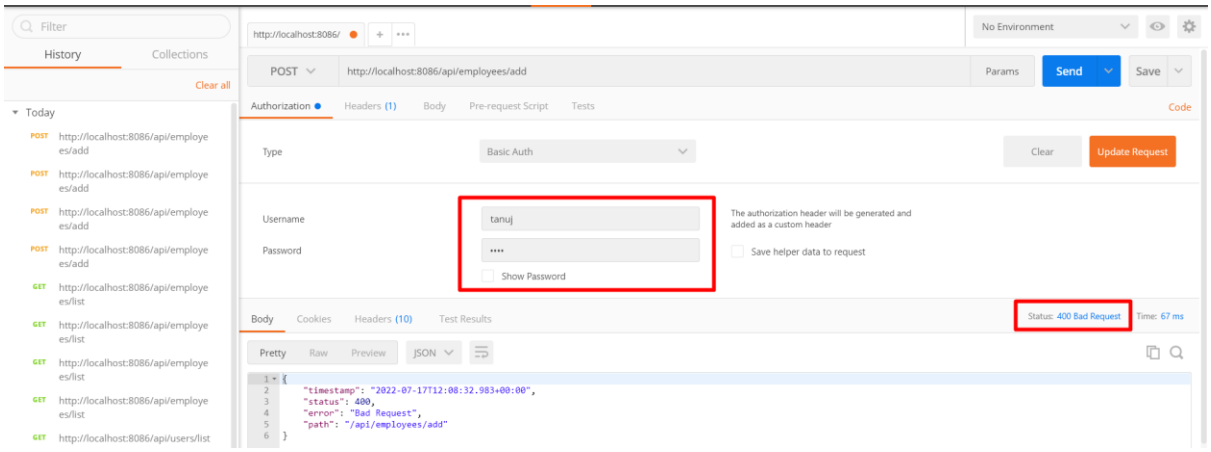
Though he is authenticated he cannot access the endpoint for adding an employee.

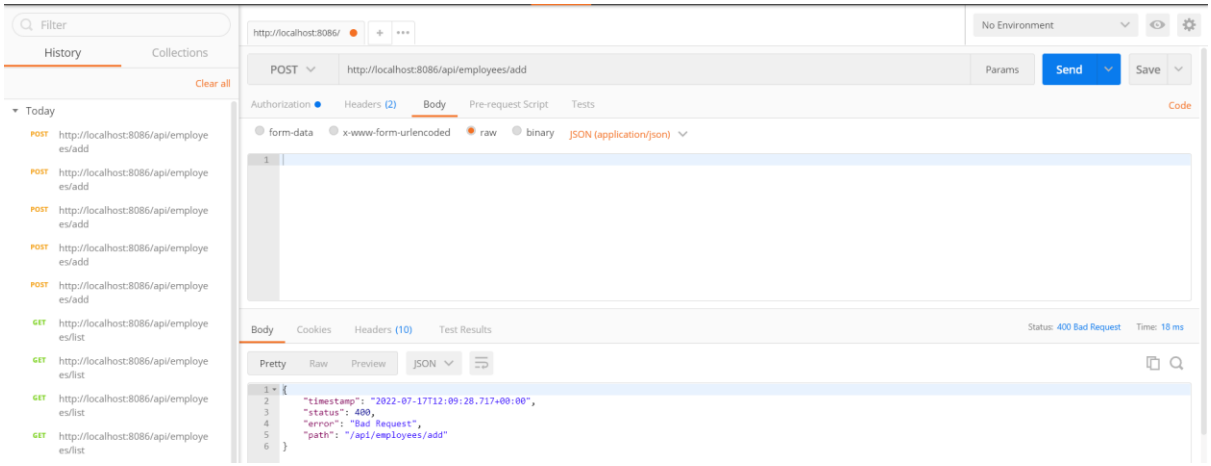
User prithvi(user) tries to add an employee.



He is also authenticated but cannot access the endpoint.

User tanuj(admin) tries to add an employee.



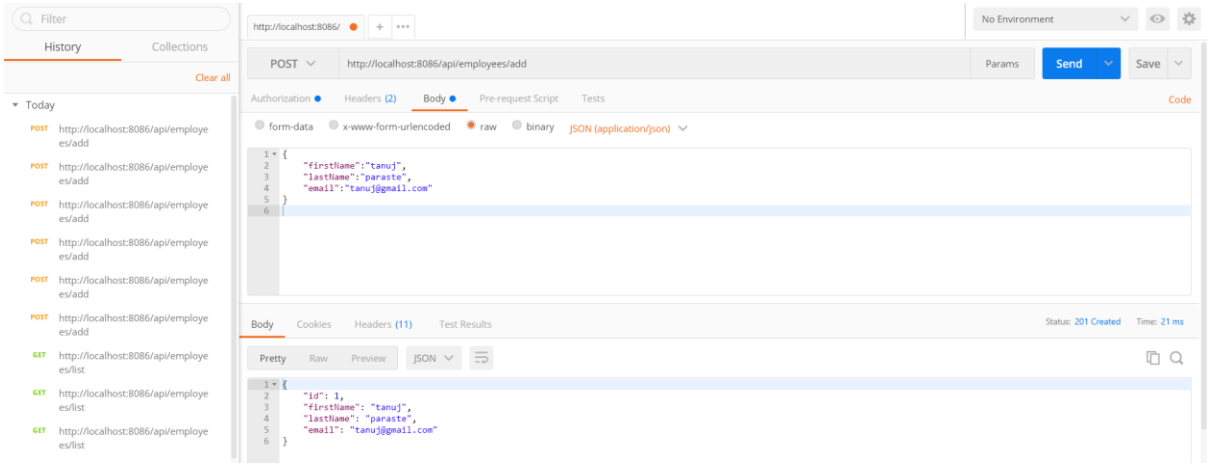


He can access the endpoint for adding an employee. He is not getting 403 Forbidden response status. This is the expected behavior of the application: only ADMIN user can add an employee.

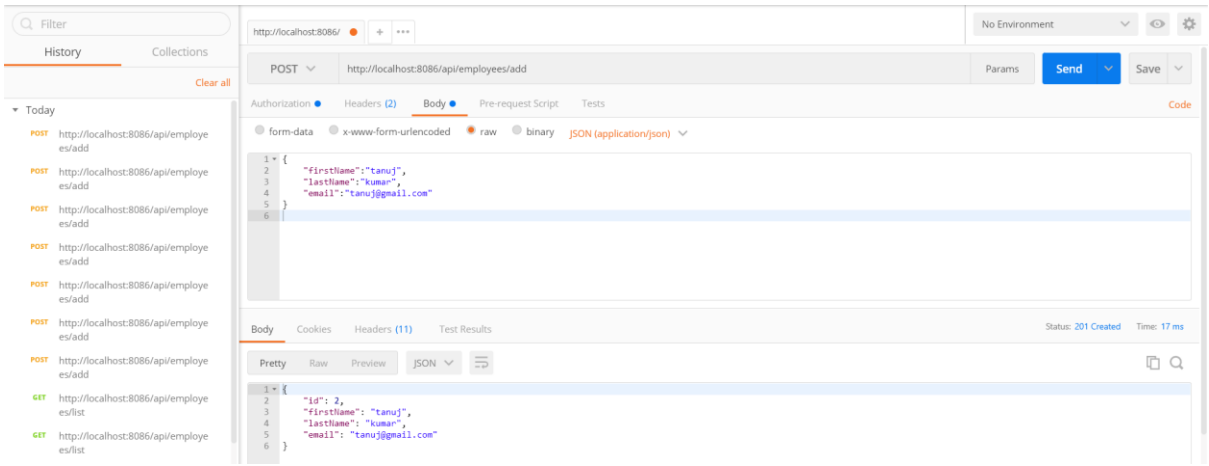
Add some employees.

{
"firstName":"tanuj",
"lastName":"paraste",
"email":"tanuj@gmail.com"
}
	{
	"firstName":"tanuj",
	"lastName":"kumar",
	"email":"tanuj@gmail.com"
	}
		{
		"firstName":"prakash",
		"lastName":"singh",
		"email":"prit@gmail.com"
		}

{
"firstName":"rahul",
"lastName":"pandey",
"email":"homi@gmail.com"
}
	{
	"firstName":"ashoka",
	"lastName":"choudhary",
	"email":"ashoka@gmail.com"
	}
		{
		"firstName":"kamal",
		"lastName":"raj",
		"email":"raj@gmail.com"
		}



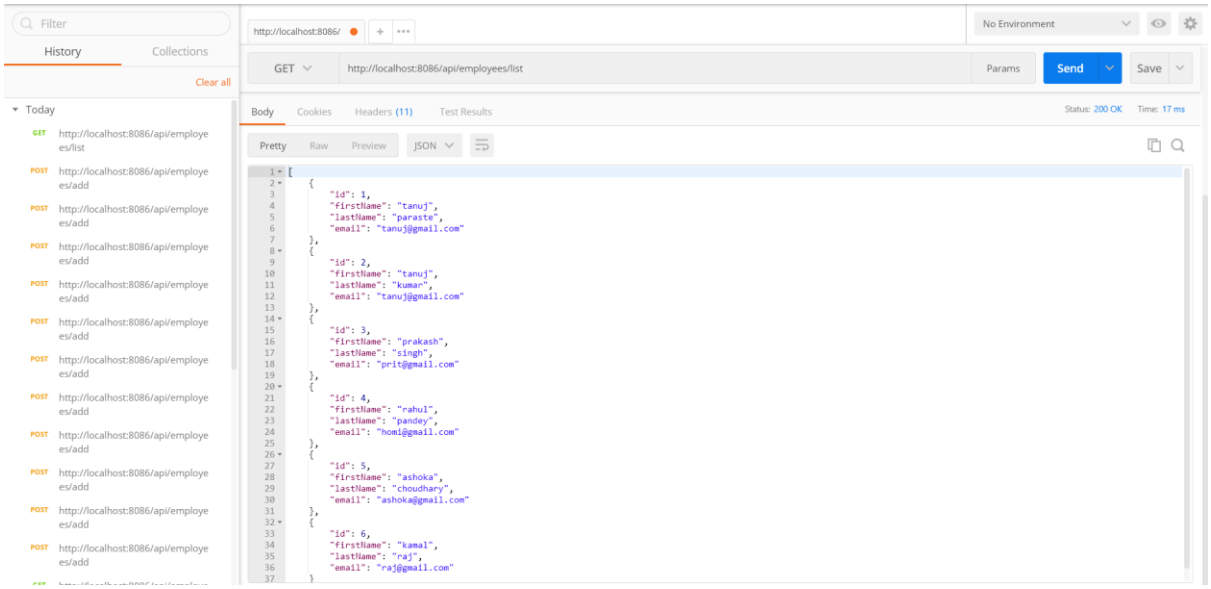
Employee *tanuj paraste* was added with id = 1



Employee *tanuj kumar* was added with id = 2

Adding other employees as …

List of all employees.

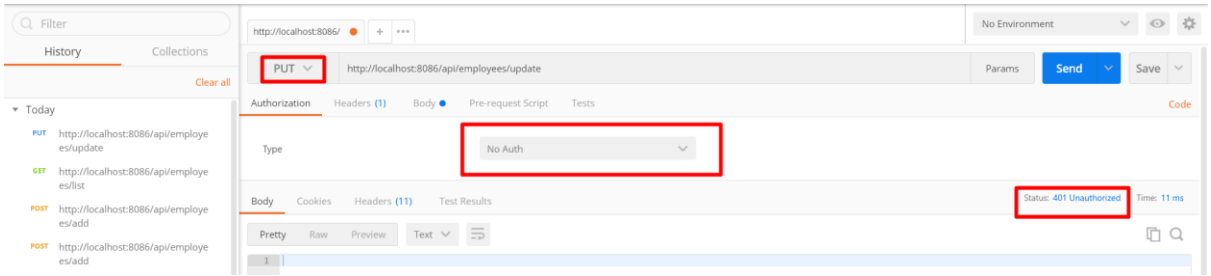


There are 6 employees in the list.

Updating an employee

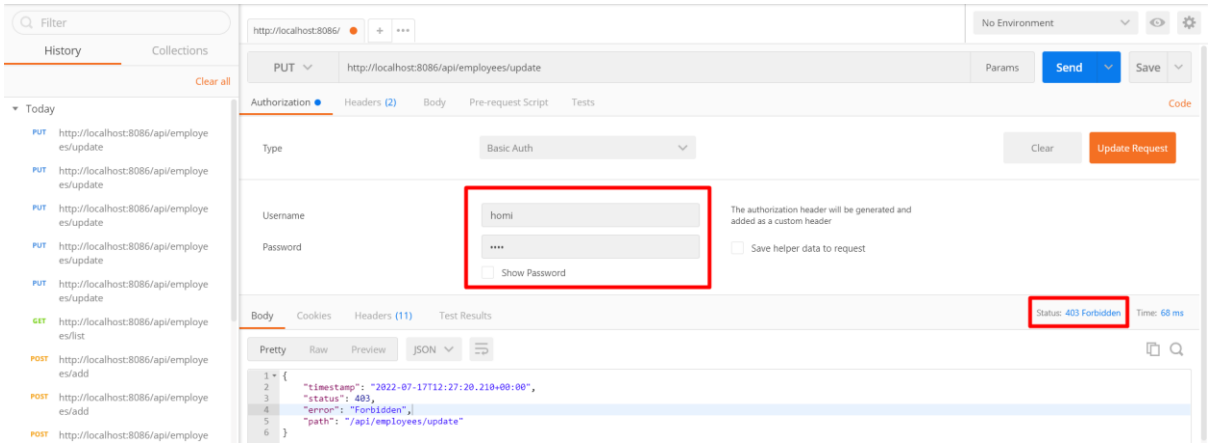
http://localhost:8086/api/employees/update (PUT request)

Try to update employee records by using normal user without authentication.



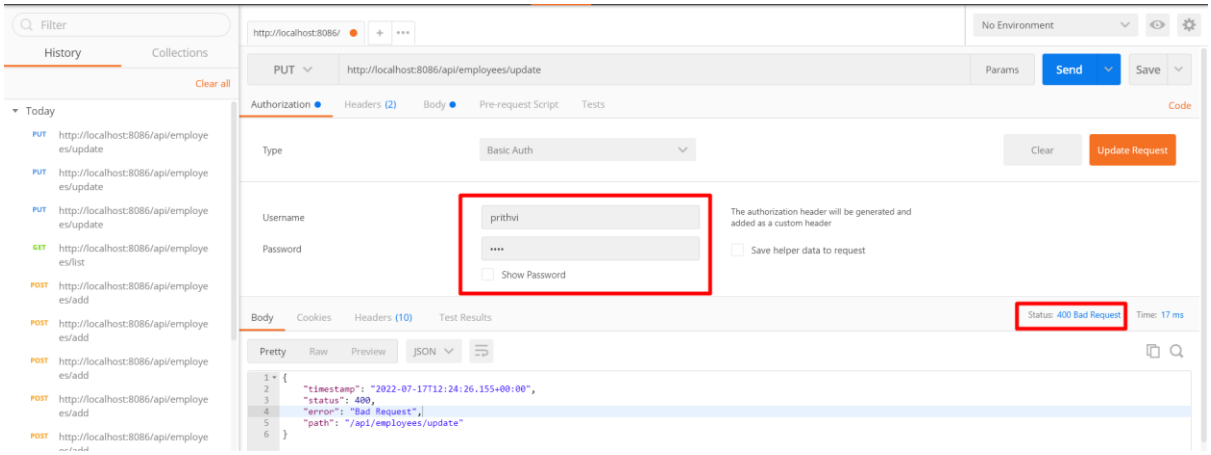
He cannot access the endpoint.

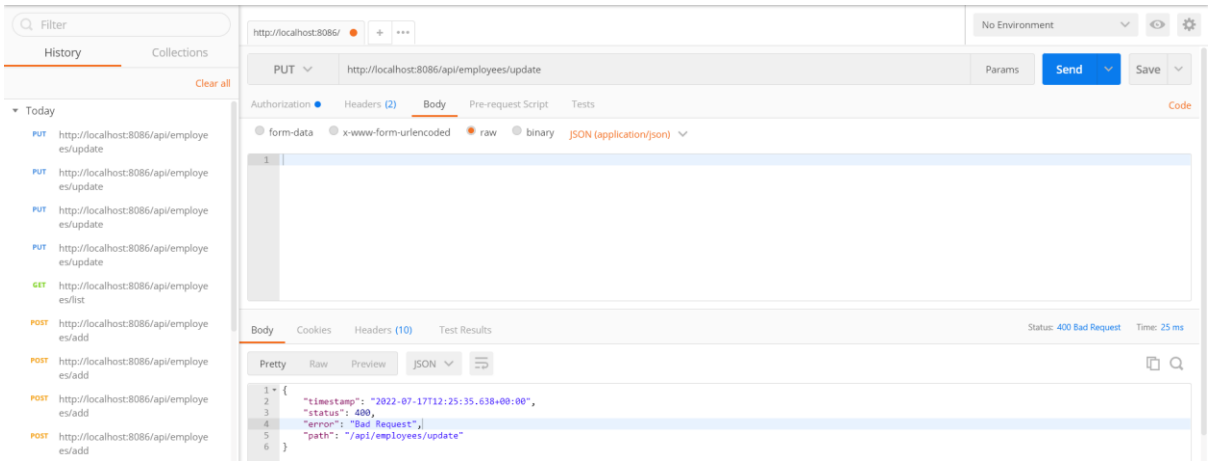
User homi(guest) tries to access after getting authenticated.



He cannot access because he does not have authority for updating records.

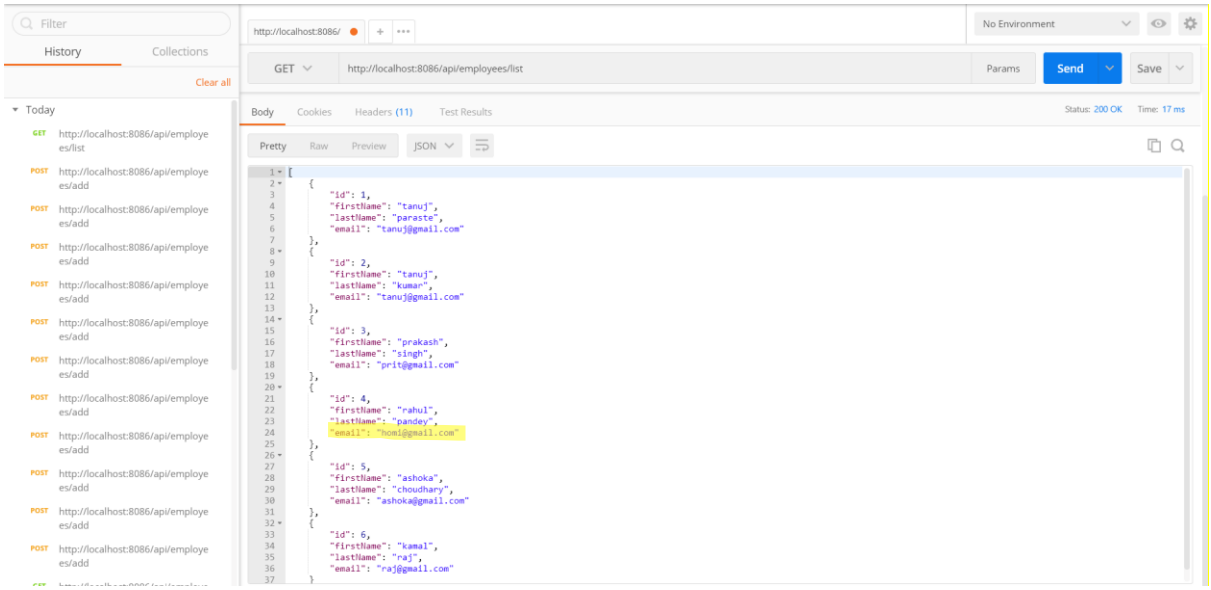
User prithvi(user) tries to update records.





As we can see, user prithvi has permission to update employee data.

Let us update data for an employee named rahul. See in the list below, the email id does not look fine.



Updating record for employee id = 4

{

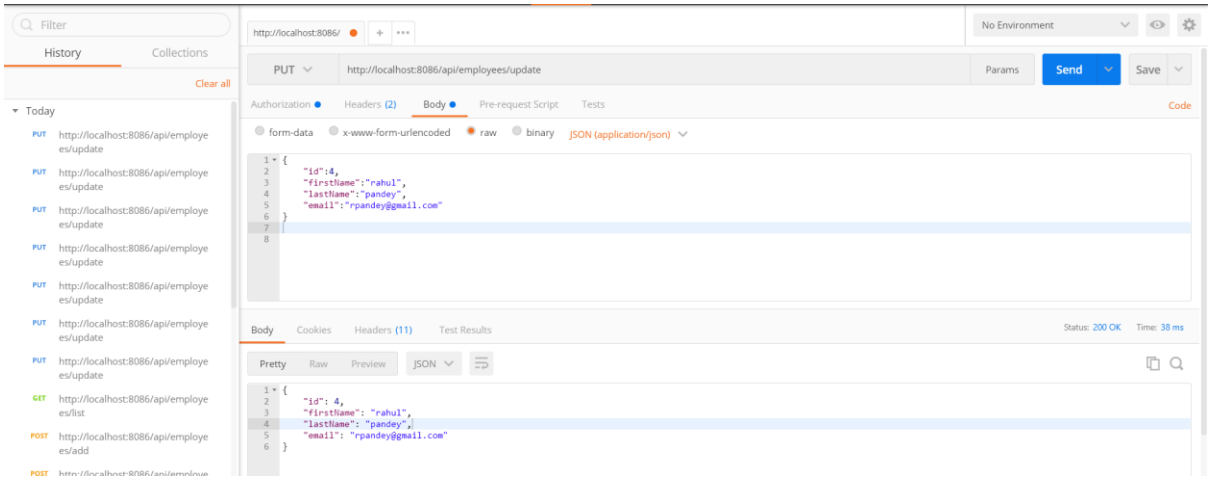
"id":4,

"firstName":"rahul",

"lastName":"pandey",

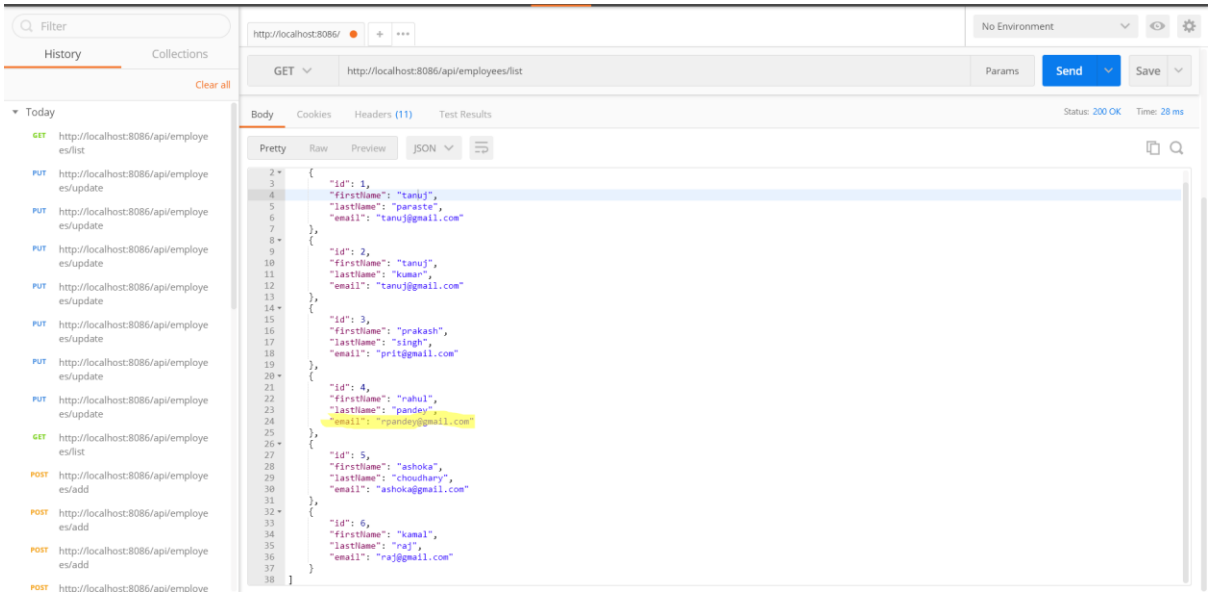
"email":"rpandey@gmail.com"

}



Updated the record.

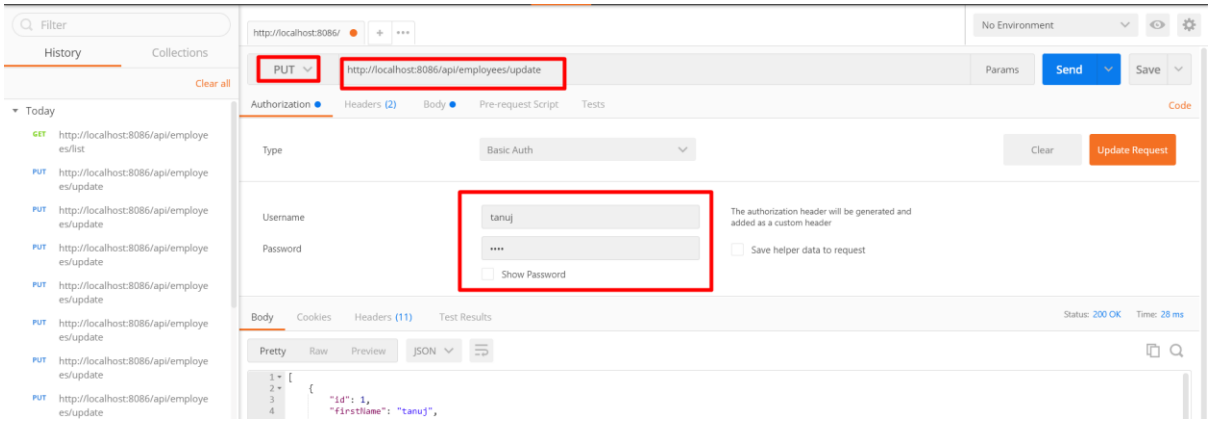
Let us look at employee list again.

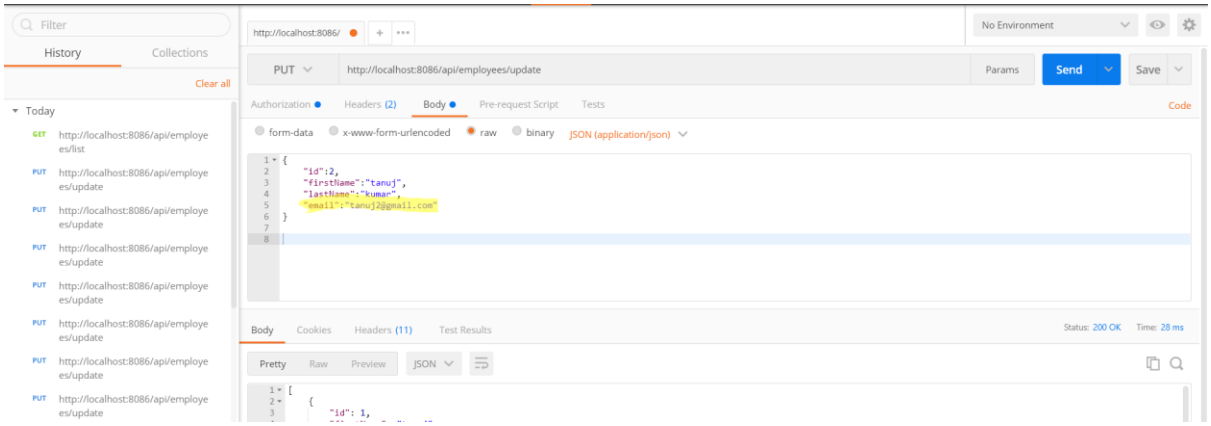


Email for employee id = 4 has been updated from *homi@gmail.com* to *rpandey@gmail.com*

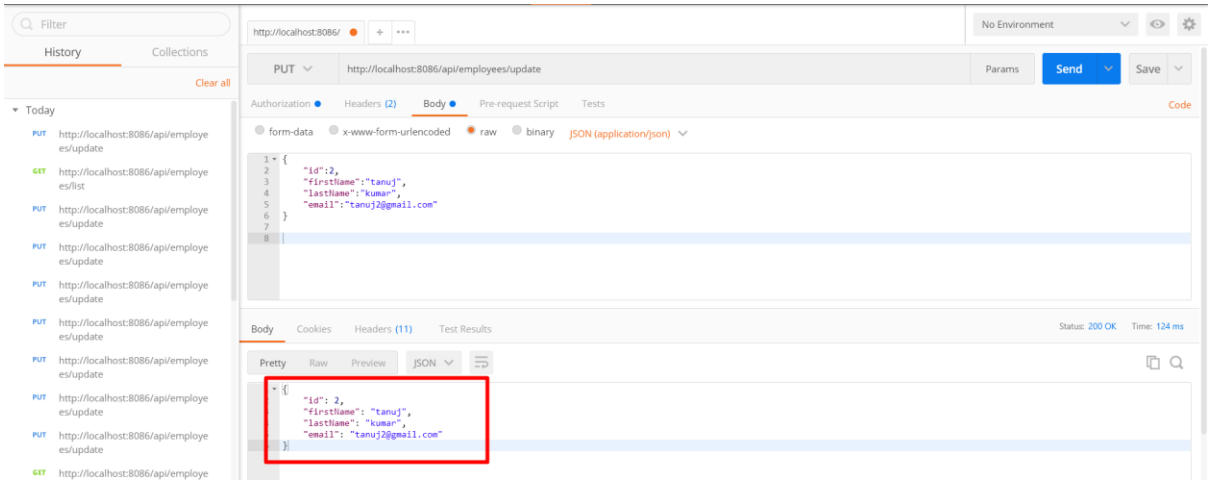
Now,

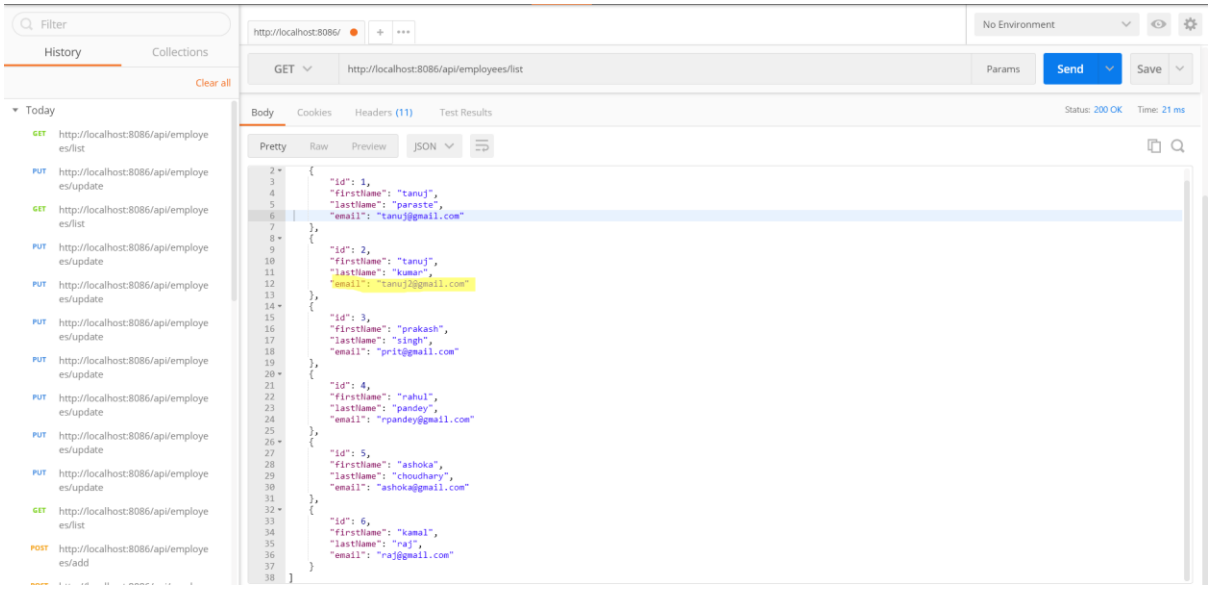
Use user tanuj(admin) to update email for id = 2 to *tanuj2@gmail.com*


Filter
History
Collections
Clear all
Today
http://localhost:8086/
No Environment
PUT
http://localhost:8086/api/employees/update
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Type
Basic Auth
Clear
Update Request
Username
tanuj
Password
Show Password
The authorization header will be generated and added as a custom header
Save helper data to request
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 28 ms
Pretty
Raw
Preview
JSON
"id": 1,
"firstName": "tanuj",


Filter
History
Collections
Clear all
Today
http://localhost:8086/
No Environment
PUT
http://localhost:8086/api/employees/update
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
"id":2,
"firstName":"tanuj",
"lastName":"kumar",
"email":"tanuj2@gmail.com"
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 28 ms
Pretty
Raw
Preview
JSON
"id": 1,

Press Send.


Filter
History
Collections
Clear all
Today
http://localhost:8086/
No Environment
PUT
http://localhost:8086/api/employees/update
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
"id":2,
"firstName":"tanuj",
"lastName":"kumar",
"email":"tanuj2@gmail.com"
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 124 ms
Pretty
Raw
Preview
JSON
"id": 2,
"firstName": "tanuj",
"lastName": "kumar",
"email": "tanuj2@gmail.com"

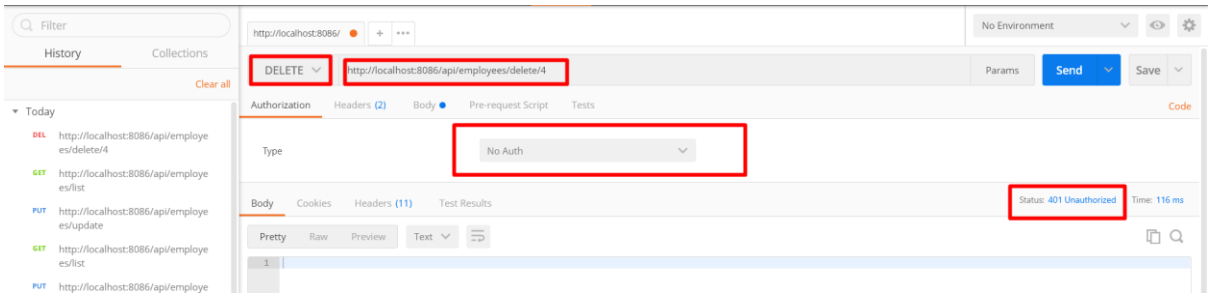


Record has been updated successfully.

Deleting an employee by id

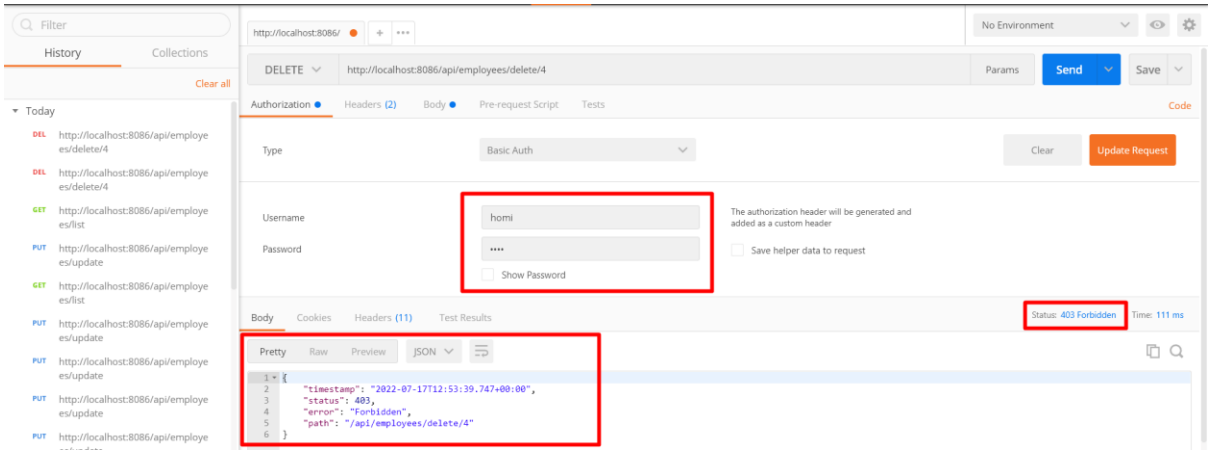
http://localhost:8086/api/employees/delete/1 (DELETE request)

Try deleting employee with id = 4 by using an unauthenticated user.



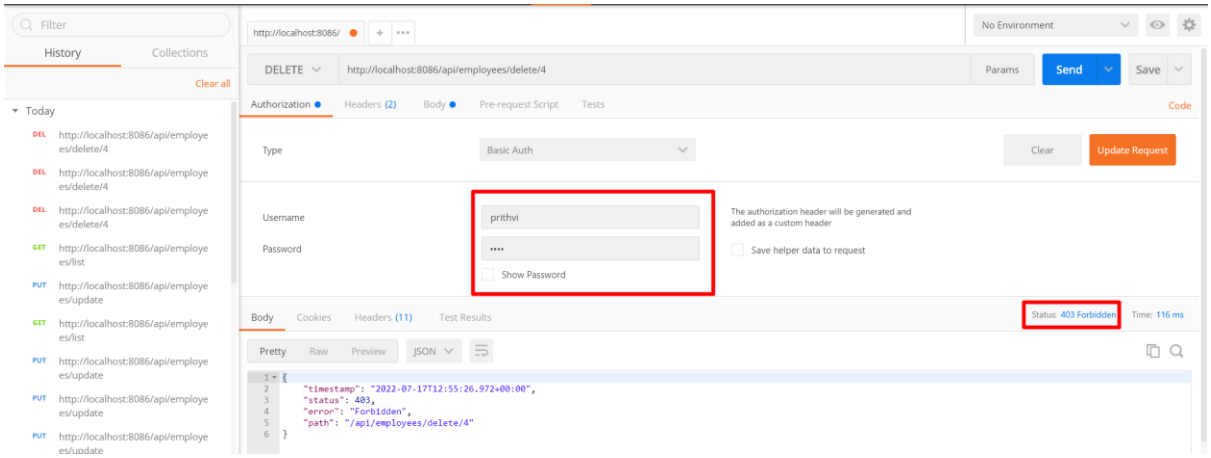
Unauthenticated users cannot access the endpoint.

Use homi(guest) tries to delete.



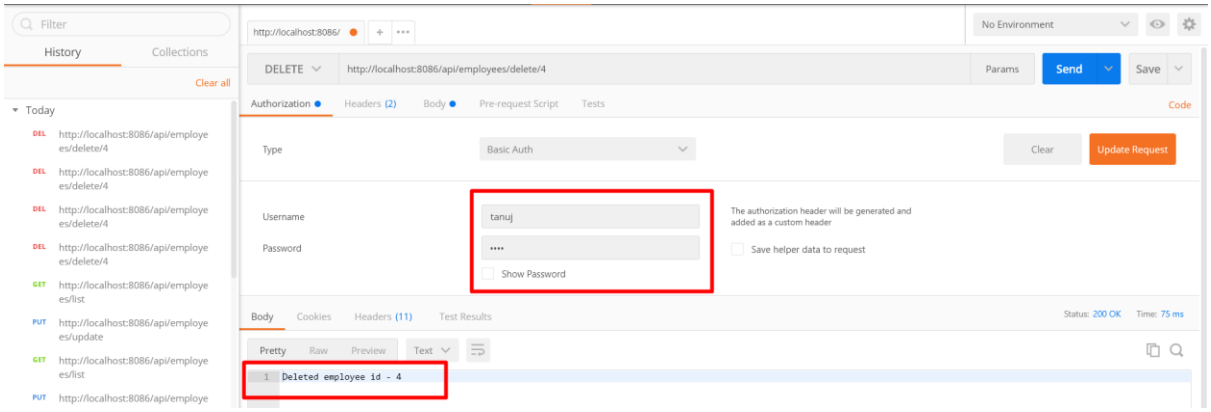
homi is authenticated but not authorized to access the resource.

User prithvi(user) tries to delete emp id = 4



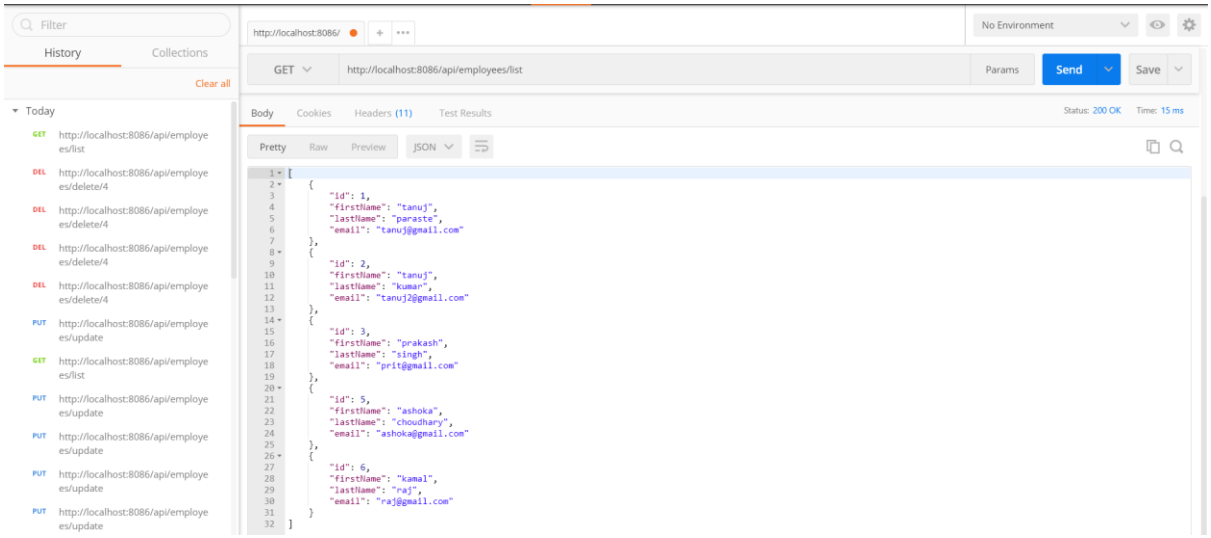
He is also a verified user but does not have access to delete endpoint.

User tanuj(admin) tries to delete the record.



tanuj could delete the record for emp id = 4 because only user with authority ADMIN can delete the record.

New employee list:

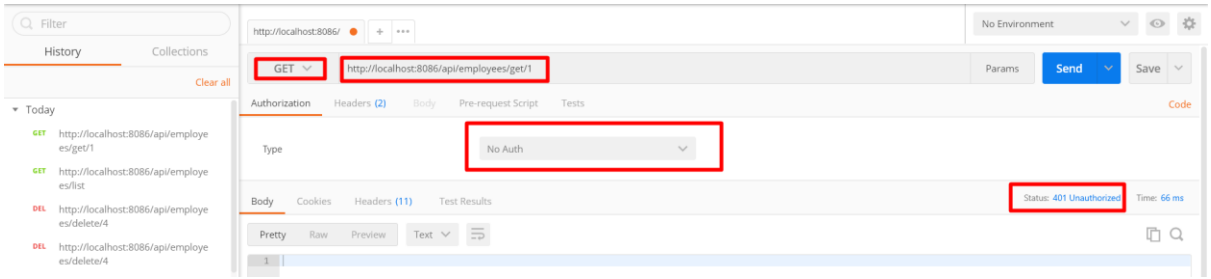


The list has 5 employees now.

Viewing an employee by id

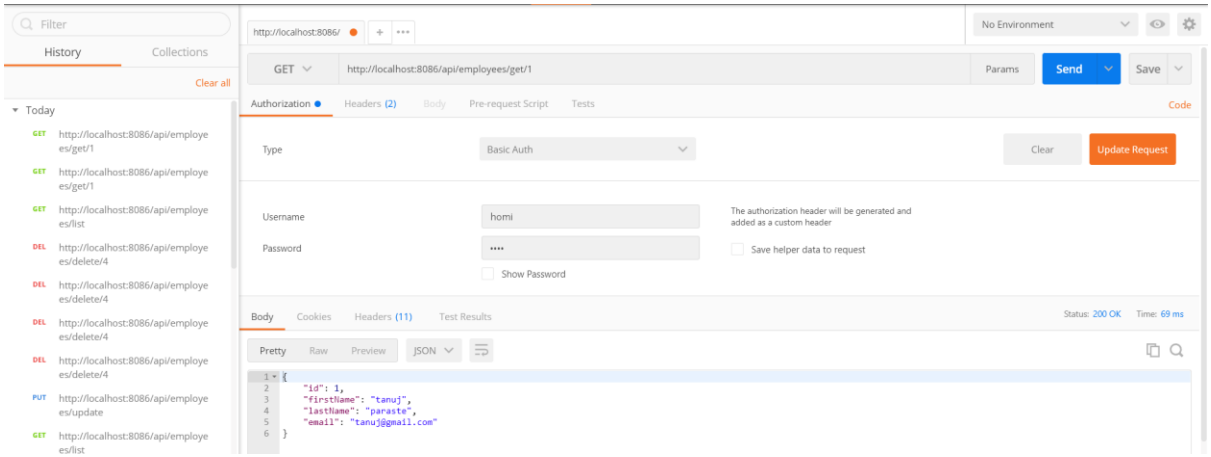
http://localhost:8086/api/employees/get/1 (GET request)

Unauthenticated user wants to view employee id = 1



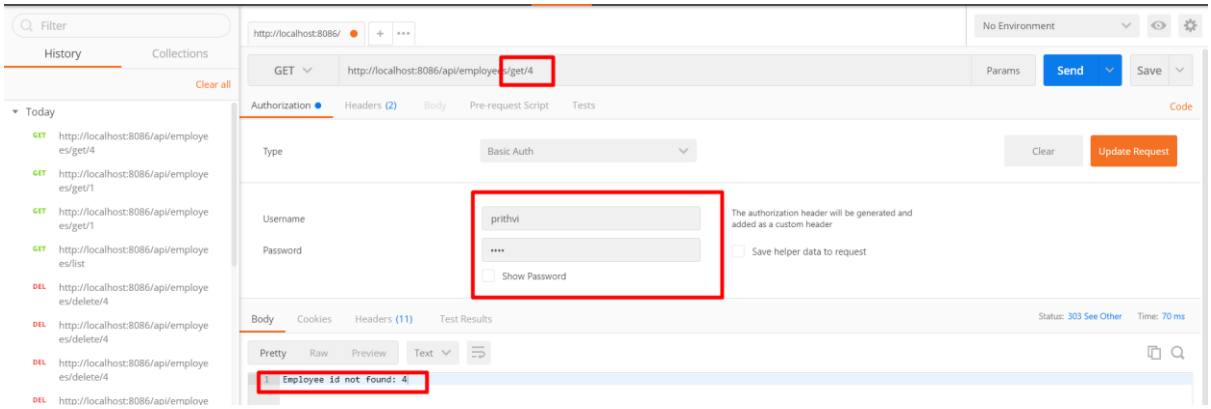
He is unauthorized to view.

User homi(guest) wants to view employee id = 1



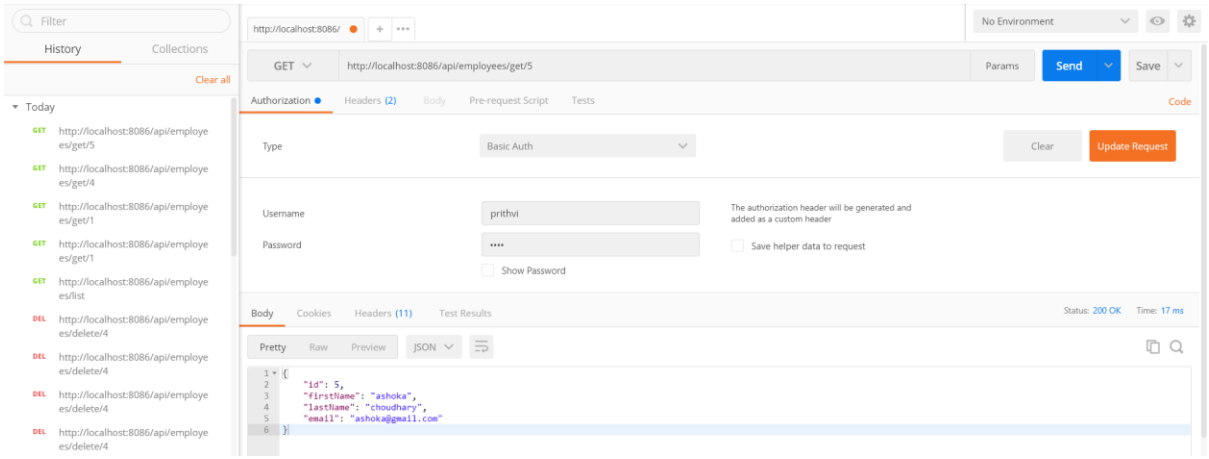
Only verified users have the authority to see an employee by id, hence homi could see employee id = 1.

User prithvi(user) wants to view employee id = 4



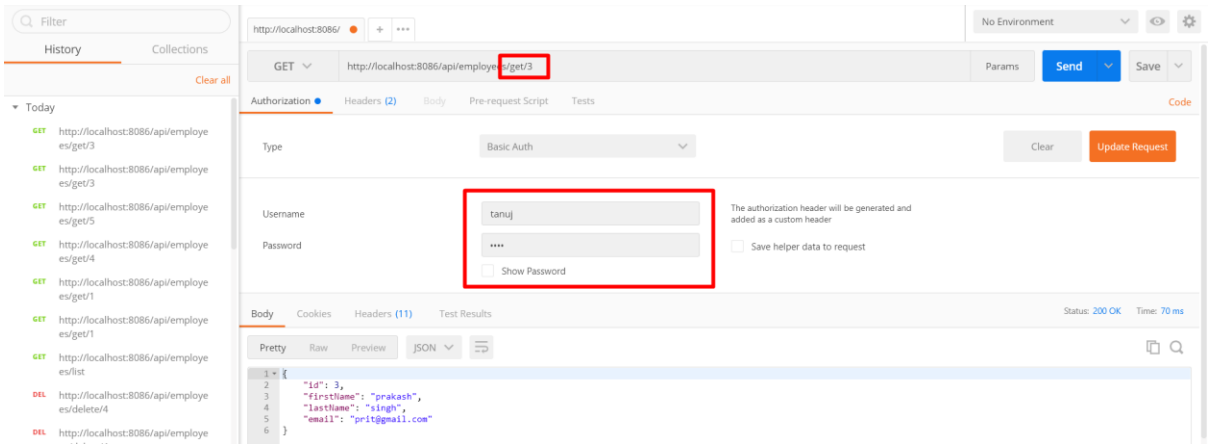
Seems employee id = 4 is not available.

Let us see employee id = 5



prithvi is authenticated hence he can fetch an employee's detail by id.

User tanuj(admin) would like to call employee id = 3 for a meeting but he does not know his name. He is searching the employee list.

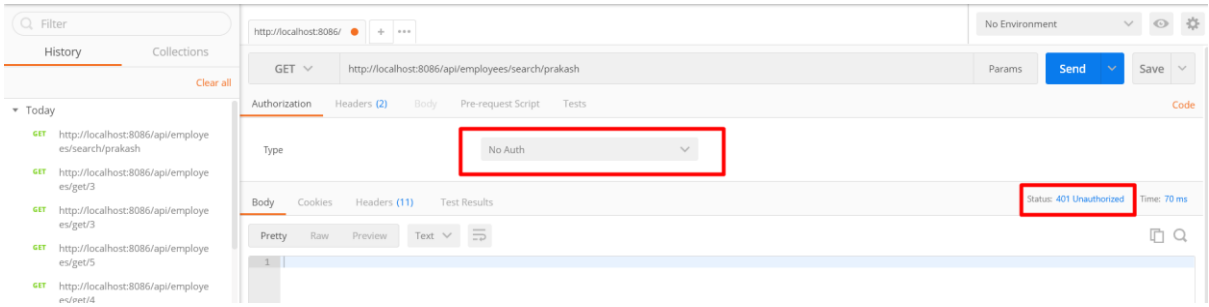


Now, he knows that he should call *prakash singh.*

Searching an employee by firstname

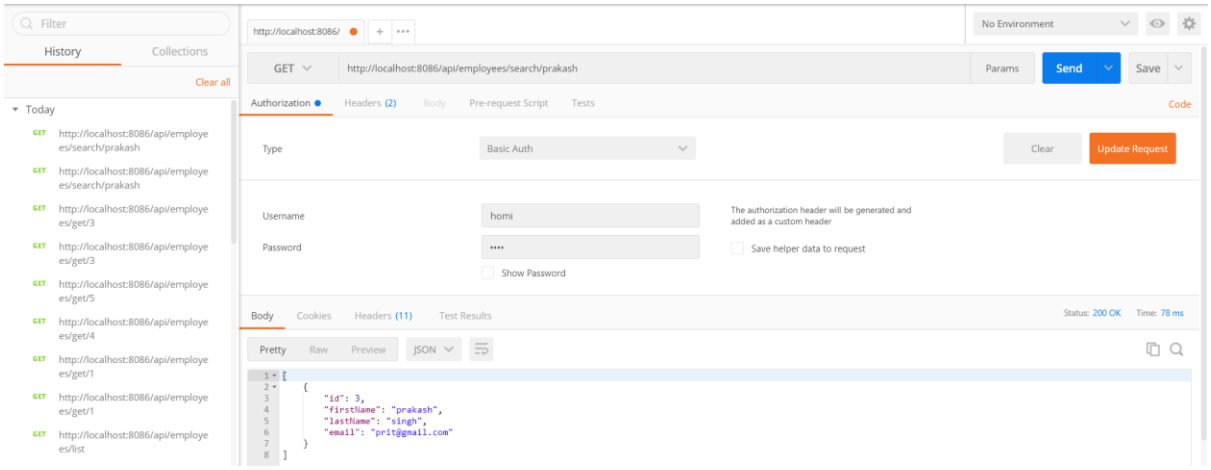
http://localhost:8086/api/employees/search/prakash (GET request)

Unauthenticated user tries to search for firstName *prakash*



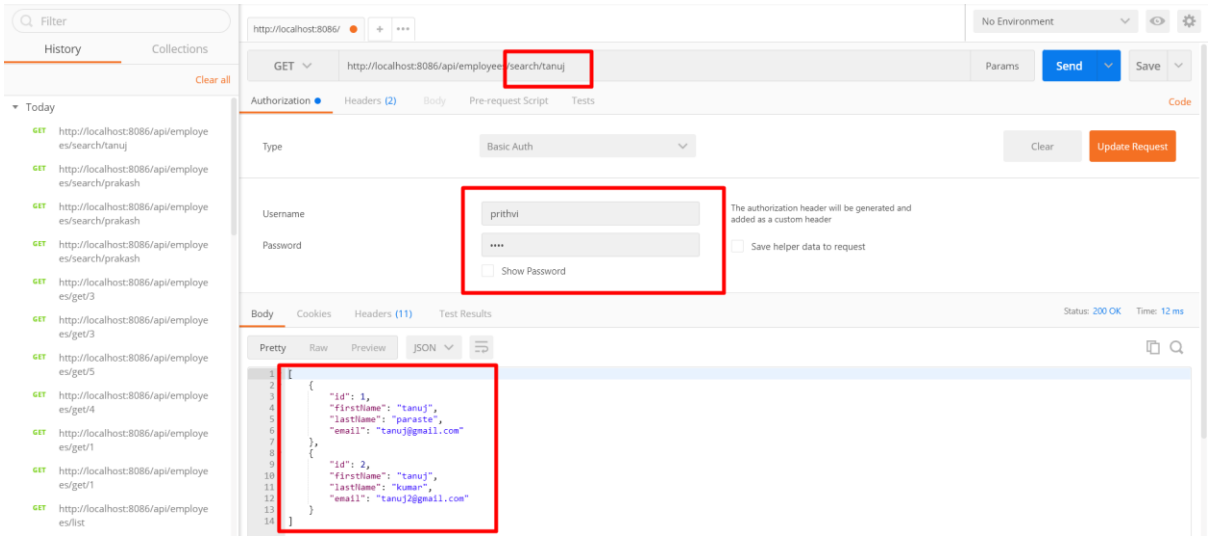
He is not permitted.

User homi(guest) searches for an employee having firstName *prakash*



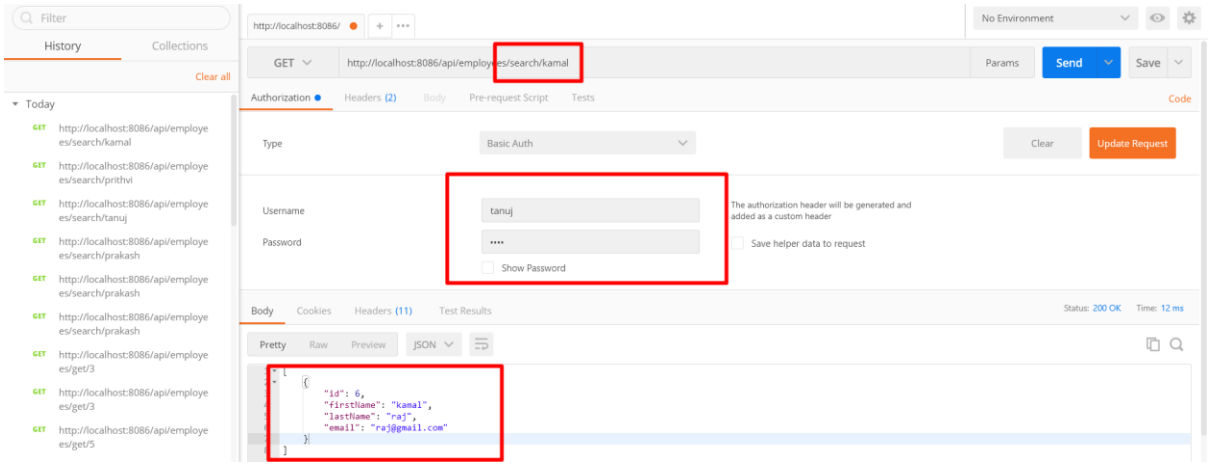
He can do so because all verified users are allowed to search for an employee based on firstname.

User prithvi(user) is looking for employees with firstname *tanuj*



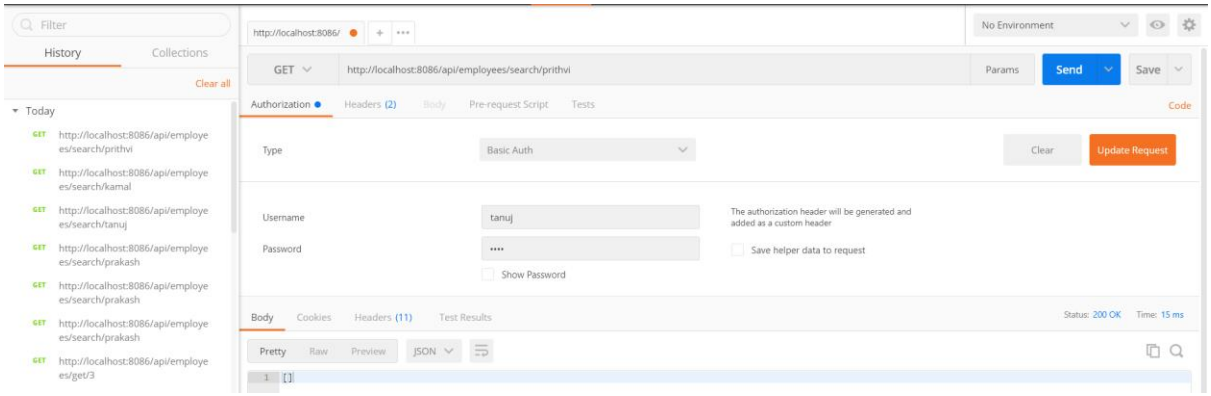
He found two employees with two records, GREAT.

User tanuj(admin) is looking for *kamal's employee id.*



He found; it is 6.

tanuj searches if there is any employee having firstname *prithvi.*



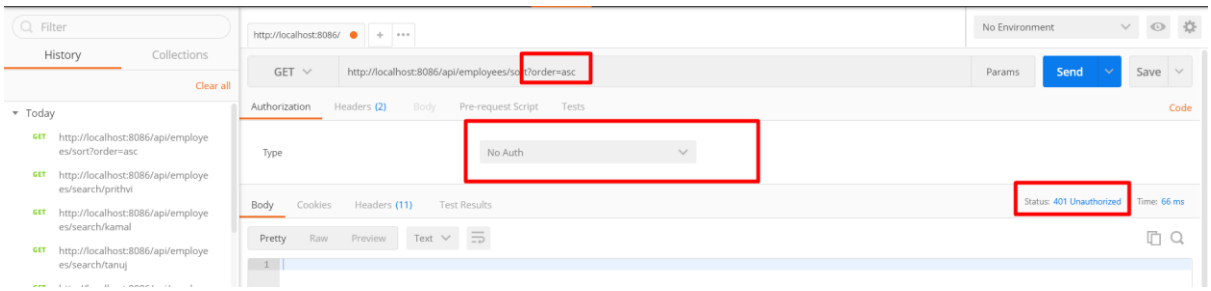
No there is no employee with that record.

Viewing ordered employee list

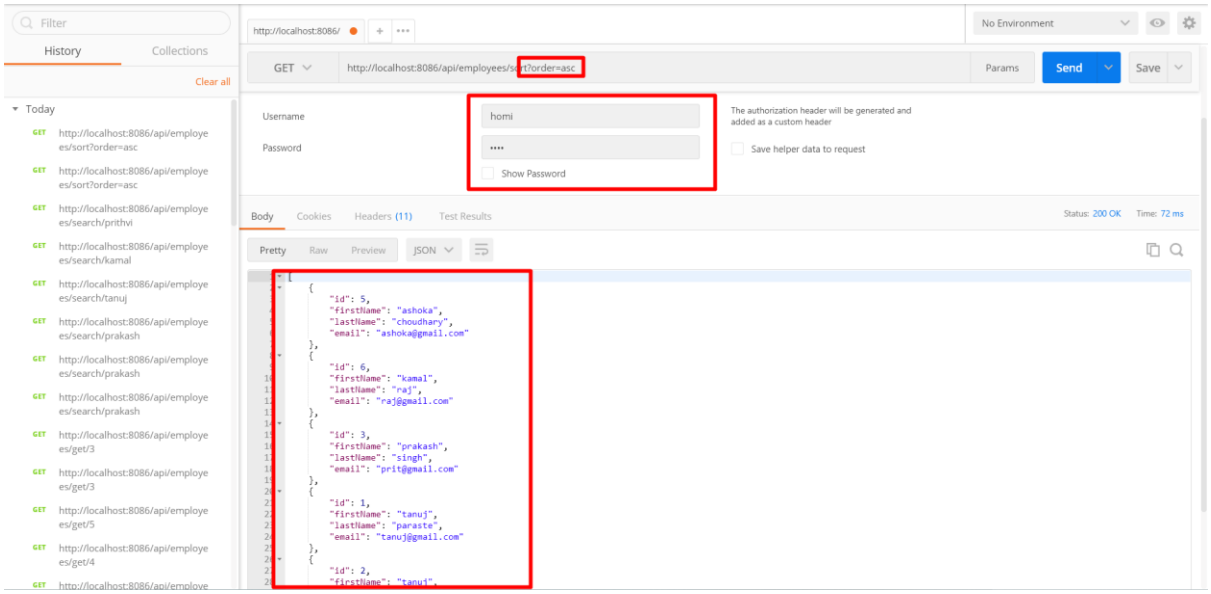
The employee list can be sorted on firstname in ascending or descending order.

http://localhost:8086/api/employees/sort?order=asc (GET request)

Unauthorized users cannot access this endpoint as can be seen in the screenshot below.

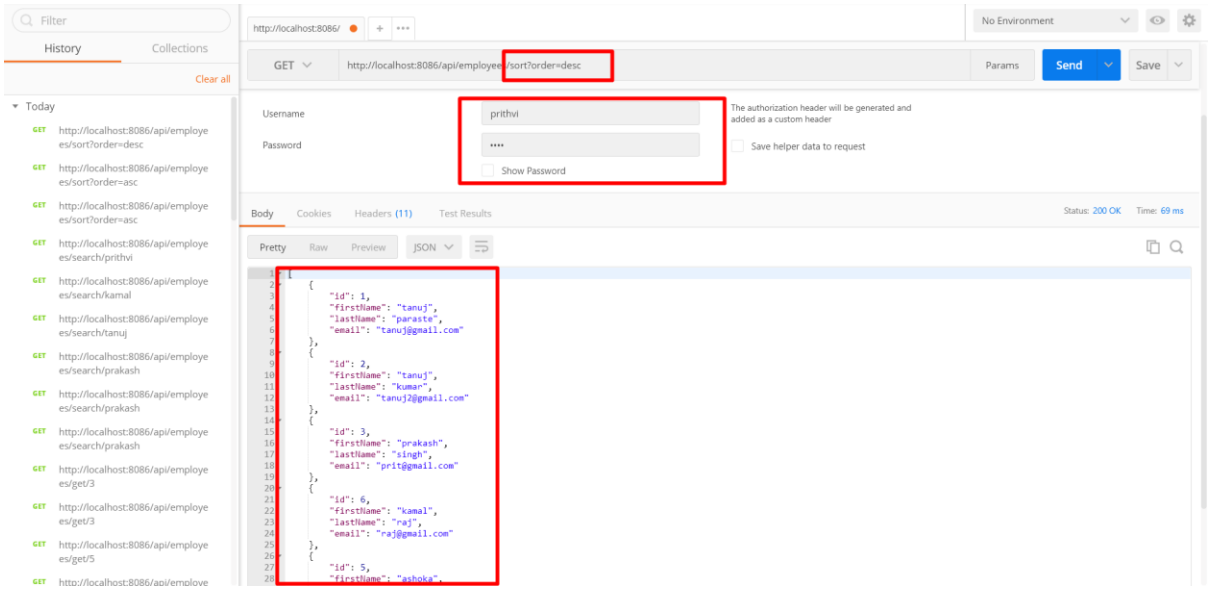


User homi(guest) wants to see the employee list in dictionary order.



He can get the list because he is authenticated and has permission.

User prithvi(user) wants the list in descending order.



He gets it.

Admin users are also allowed to sort the list.