

Task 1

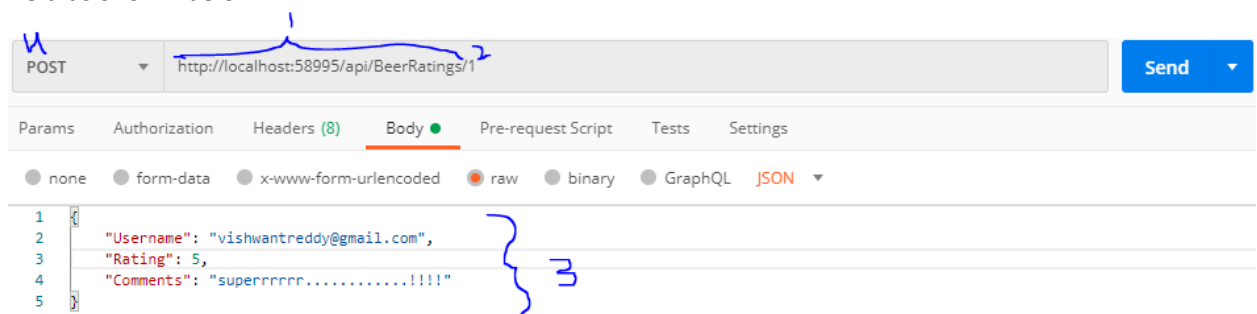
1. This endpoint should accept an id parameter and JSON request body which includes the following properties: username, rating, comments.
2. Add validation to ensure the id parameter is a valid beer id by querying the Punk API.
3. Add validation to ensure that the rating is a valid value in the range of 1 to 5.
4. If any of the validations fail an applicable error should be returned to the user.
5. All valid requests should append the JSON from the request body to a file called database.json stored within a solution folder of your choosing.

Steps to perform

1. Make sure the database.json file exists and have square brackets to append the ratings into it as an array.



- For testing, first run the application and open any API testing tool, like postman. And set the field as shown below.



1. Get the api endpoint by running the application and copy the endpoint from browser.
2. Set a valid beer ID (in this case I used 1).
3. Form the body of the request as shown above , containing (Username(valid string), rating (int), comments (string)).
4. Finally set the verb as POST and send the request.

Endpoint: <http://localhost:port/api/BeerRatings/{id}>

Method: POST

sample Request: {

```
"Username": "vishwantreddy@gmail.com",  
"Rating": 5,  
"Comments": "superrrrrr.....!!!!"  
}
```

Provided the rating is between 1 and 5, and Valid email address username.
Response would be

Response: Successfully rating added

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:58995/api/BeerRatings/1` with a 'Send' button. Below the bar, tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. The 'Body' tab is selected, showing a JSON payload:

```
{  
  "Username": "vishwantreddy@gmail.com",  
  "Rating": 5,  
  "Comments": "superrrrrr.....!!!!"  
}
```

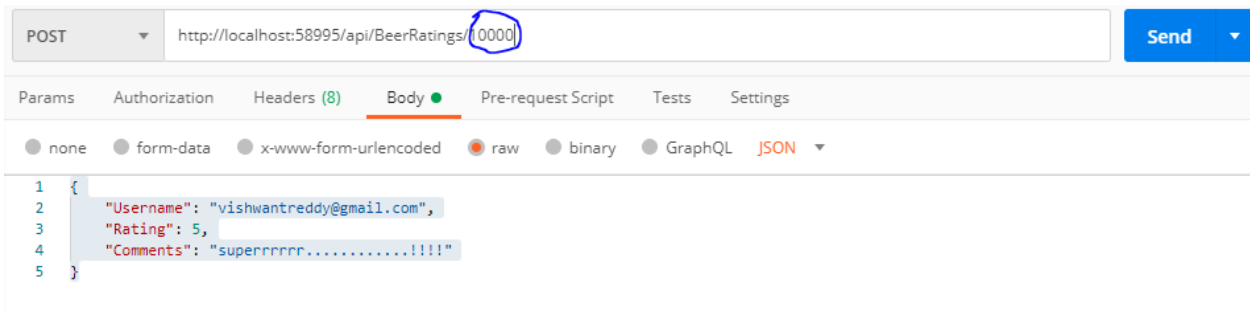
. Below the body, tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results' are shown. The 'Test Results' tab is selected, displaying a status of '200 OK', a time of '1996 ms', and a size of '248 B'. The response body is shown as 'Successfully rating added'.

Verification: now in database.json you can the see the rating added.

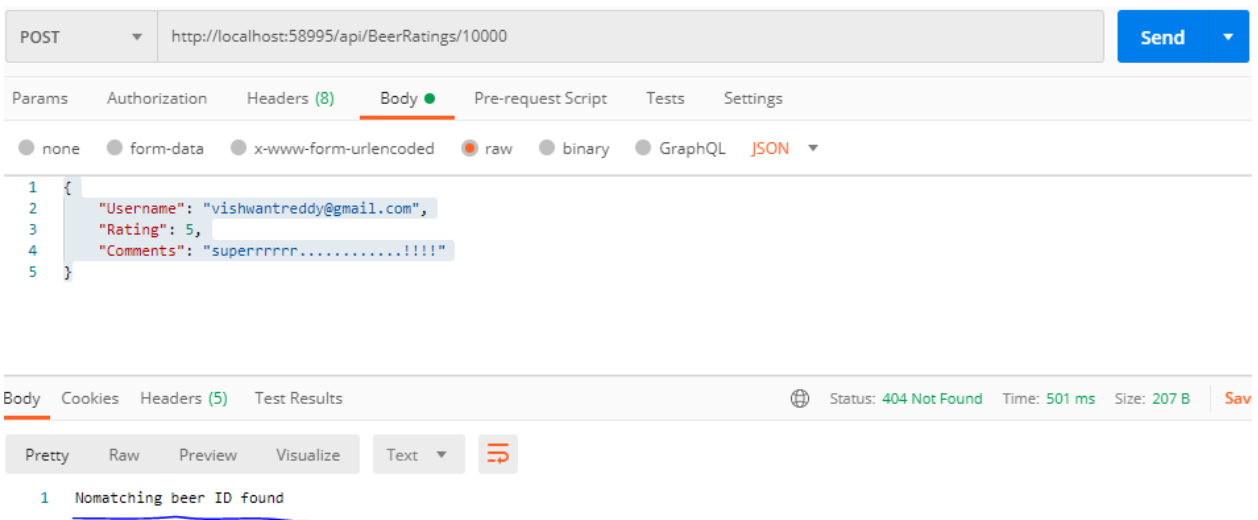
The screenshot shows a JSON viewer interface. The top bar indicates the file 'database.json' is open. Below the bar, a 'Schema' dropdown is set to '<No Schema Selected>'. The JSON content is displayed in a tree view, showing an array with one object:

```
[  
  {  
    "Username": "vishwantreddy@gmail.com",  
    "Rating": 5,  
    "Comments": "superrrrrr.....!!!!"  
  }  
]
```

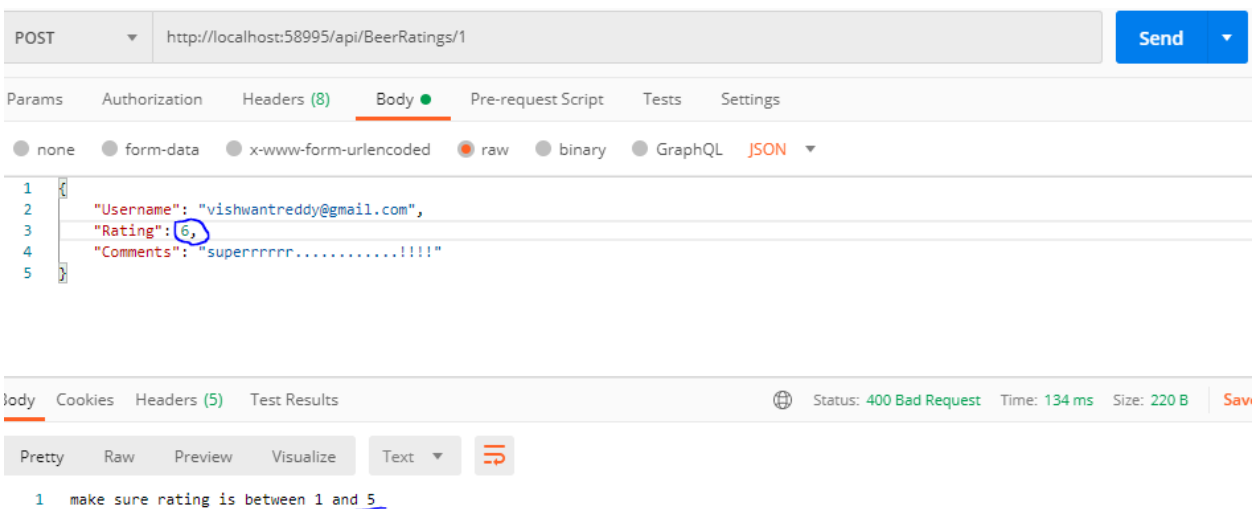
Now, let see if we send wrong Beer ID, what would happen. I used 10000 as beer ID.



Response: NO matching beer ID found is the response. (as there is no Beer with ID 10000 in punk API)



Now, Let's see what happens if we send invalid rating (below 1 or above 5).



As you see if we pass invalid rating the response is “make sure rating is between 1 and 5”.

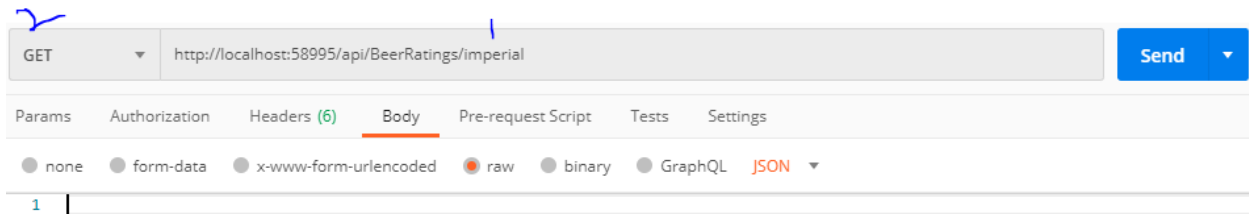
Now, we achieved as the requirements of task #1. Lets move on to Task #2.

Task 2.

1. This endpoint should accept one parameter in the query string of the request. The purpose of this parameter is to denote the name of the beer to search for.
The implementation of your REST endpoint will use the public available Punk API to retrieve all beers matching the search parameter described above.
2. After retrieving the search result, load the contents of the database.json file into an in-memory collection and write a Linq query to project an API response that follows the structure of the following example:

```
[
  {
    "id":1,
    "name":"Buzz",
    "description":"A light, crisp and bitter IPA brewed with English and American hops. A small batch brewed only once.",
    "userRatings":[
      {
        "username":"john.doe@fictitious.com",
        "rating":3,
        "comments":"Lorem ipsum dolor sit amet, consectetur adipiscing elit",
      },
      {
        "username":"max.power@fictitious.com",
        "rating":5,
        "comments":"sed do eiusmod tempor incididunt ut labore",
      }
    ]
  }
]
```

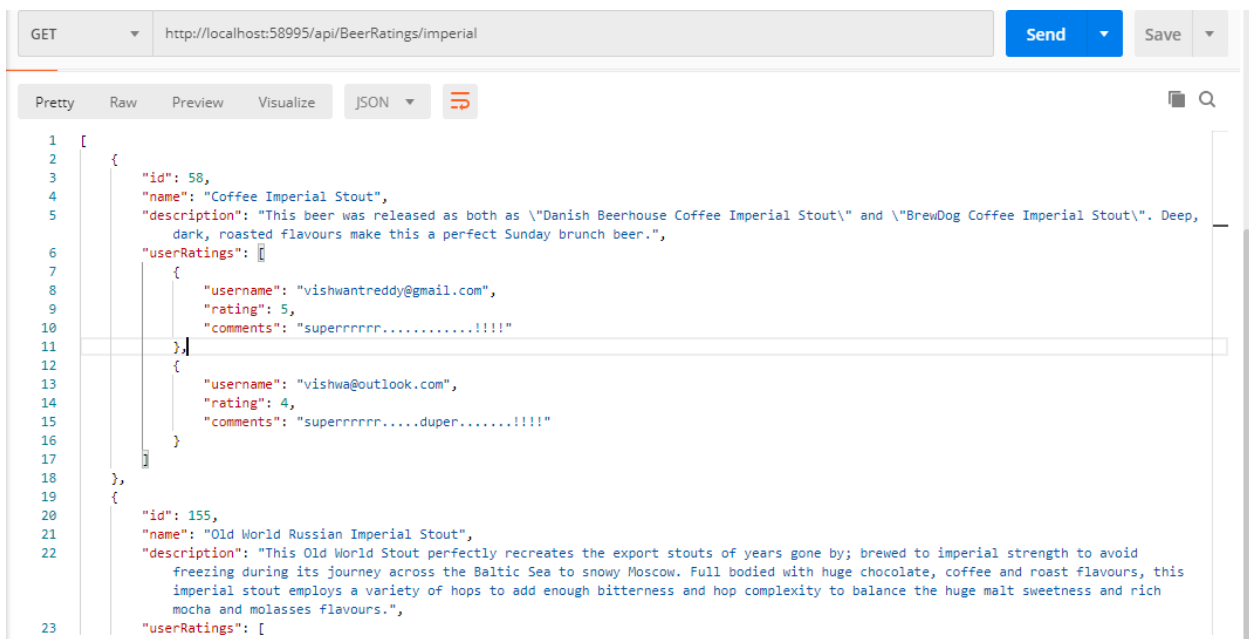
1. Now to find the all the beers with matching name. please follow the below steps.



1. Replace the beer ID from earlier endpoint with name of beer.
2. Change the Http method to GET.

Now replace the Beer ID with beer Name you want to search for. And change the method to GET and also remove the body.

Now we expect all the beers matching with name imperial with all the ratings added previously.



Response: As you can see we get all three beers matching the word “imperial” and also the appended with User ratings extracted from database.json file.

```
[
  {
    "id": 58,
    "name": "Coffee Imperial Stout",
    "description": "This beer was released as both as \"Danish Beerhouse Coffee Imperial Stout\" and \"BrewDog Coffee Imperial Stout\". Deep, dark, roasted flavours make this a perfect Sunday brunch beer.",
    "userRatings": [
      {
        "username": "vishwantreddy@gmail.com",
        "rating": 5,
        "comments": "superrrrrr.....!!!!!"
      },
      {
        "username": "vishwa@outlook.com",
        "rating": 4,
        "comments": "superrrrrr.....duper.....!!!!!"
      }
    ]
  },
  {
    "id": 155,
    "name": "Old World Russian Imperial Stout",
    "description": "This Old World Stout perfectly recreates the export stouts of years gone by; brewed to imperial strength to avoid freezing during its journey across the Baltic Sea to snowy Moscow. Full bodied with huge chocolate, coffee and roast flavours, this imperial stout employs a variety of hops to add enough bitterness and hop complexity to balance the huge malt sweetness and rich mocha and molasses flavours.",
    "userRatings": [
      {
        "username": "vishwantreddy@gmail.com",
        "rating": 4,
        "comments": "superrrrrr.....!!!!!"
      }
    ]
  }
]
```

```

        "rating": 5,
        "comments": "superrrrrr.....!!!!"
    },
    {
        "username": "vishwa@outlook.com",
        "rating": 4,
        "comments": "superrrrrr.....duper.....!!!!"
    }
]
},
{
    "id": 155,
    "name": "Old World Russian Imperial Stout",
    "description": "This Old World Stout perfectly recreates the export stouts of
years gone by; brewed to imperial strength to avoid freezing during its journey across
the Baltic Sea to snowy Moscow. Full bodied with huge chocolate, coffee and roast fla
vours, this imperial stout employs a variety of hops to add enough bitterness and hop
complexity to balance the huge malt sweetness and rich mocha and molasses flavours.",
    "userRatings": [
        {
            "username": "vishwantreddy@gmail.com",
            "rating": 5,
            "comments": "superrrrrr.....!!!!"
        },
        {
            "username": "vishwa@outlook.com",
            "rating": 4,
            "comments": "superrrrrr.....duper.....!!!!"
        }
    ]
},
{
    "id": 269,
    "name": "Small Batch: Imperial Pale Weizen",
    "description": "An amplified version of a style we first brewed with Weiheste
phan. A full-
on deluge of spice and citrus, the new world hops bring spiced orange, lemon peel and
fresh grassy and floral notes while the weizen base adds pepper, clove and banana.",
    "userRatings": [
        {
            "username": "vishwantreddy@gmail.com",
            "rating": 5,
            "comments": "superrrrrr.....!!!!"
        },
        {
            "username": "vishwa@outlook.com",
            "rating": 4,
            "comments": "superrrrrr.....duper.....!!!!"
        }
    ]
}
]
}
]

```

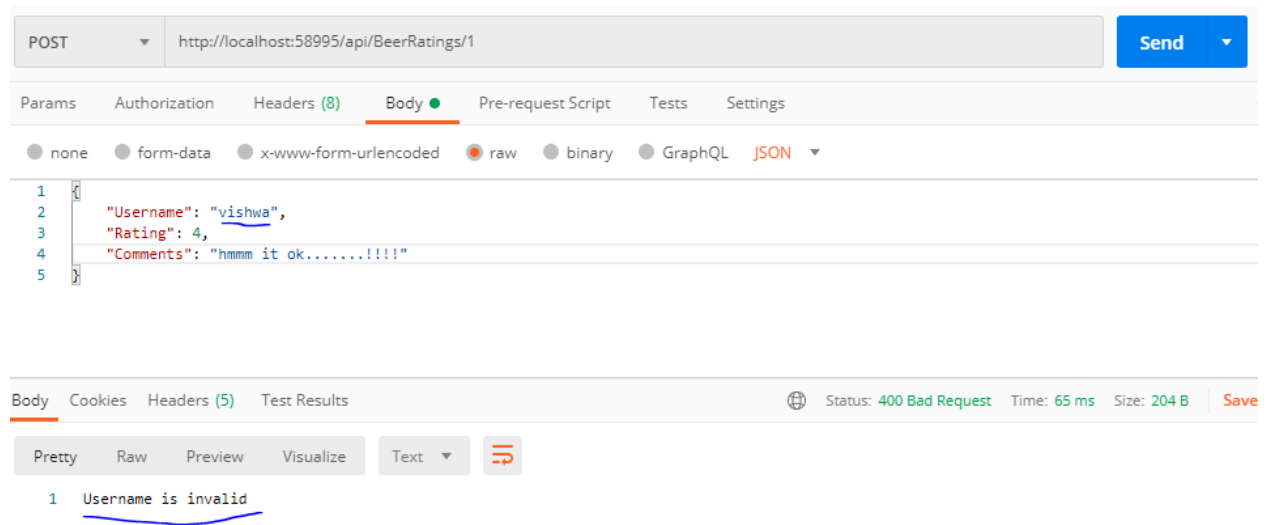
Now, we have achieved all the requirements of task #2. Let's move on task #3.

Task 3: Add a custom Web API Action Filter Attribute

1. This action filter attribute is for the purpose of validating the username supplied in the JSON body of the REST operation described in task 1. The validation should use a regular expression to ensure the username value is a valid formatted email address. If the username is not valid an applicable error should be returned by the API.

1. Steps to see if the username is valid or not (using action filter).

As we already saw in Task #1 that if we use a valid email as username in body we were able to add that body to database.json, but let see what happened if we the username in body with invalid email type.



As you can see when the username is not a valid email type is responds with message “username is invalid”. This was achieved through the use if Action Filter. Now let’s move on to Task #4.

Task 4: Add unit tests

Add applicable Unit tests to a routine of your choosing that formed part of the implementation of one of the above tasks.

1. I have written two simple unit tests to verify that if invalid rating or invalid beer id is submitted, does the systems respond correctly,

```

namespace BeerApiTest
{
    public class UnitTest1
    {
        [Fact]
        public async void TestInvalidRating()
        {
            var controller = new BeerRatingsController();
            var rating = new Ratings() { Username = "vishwant@gmail.com", Rating = 6, Comments = "superrrr!!!!!!!" };
            int beerId = 1;
            var result = await controller.AddRating(beerId, rating);

            var badRequestResult = Assert.IsType<BadRequestObjectResult>(result);
        }

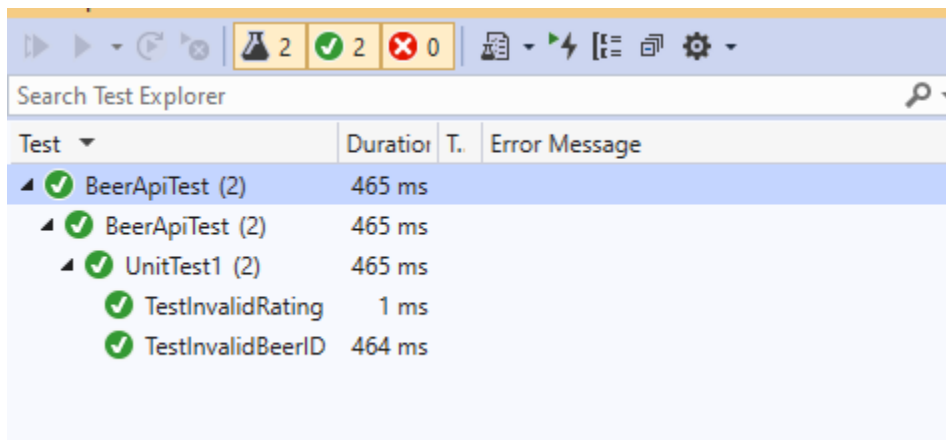
        [Fact]
        public async void TestInvalidBeerID()
        {
            var controller = new BeerRatingsController();
            var rating = new Ratings() { Username = "vishwant@gmail.com", Rating = 5, Comments = "superrrr!!!!!!!" };
            int invalidBeerId = 82738;
            var result = await controller.AddRating(invalidBeerId, rating);

            var badRequestResult = Assert.IsType<NotFoundObjectResult>(result);
        }
    }
}

```

When the rating is invalid the test expects a BadRequestObjectResult.

And when the beer id is invalid the test expects a NotFoundObjectResult, let see if the test cases pass.



Test	Duration	T.	Error Message
BeerApiTest (2)	465 ms		
BeerApiTest (2)	465 ms		
UnitTest1 (2)	465 ms		
TestInvalidRating	1 ms		
TestInvalidBeerID	464 ms		

As we can see the we got the expected responses and test cases passed.