Bazar.com

D.Samer alarndi

Student1: Mohammad Mughrabi.

Student2: Jehad Abu Raed.

Student 1 ID:11940945

Student 2 ID:

Micro services are a well-known architecture style, having advantages and dis advantages, we were asked to build Bazor.com the world smallest book store, to experience how to build an application using it.

I choose to build this application using asp.net core where I'm familiar with it, I made three solutions each one representing a server as described in the description of the application figure 1.

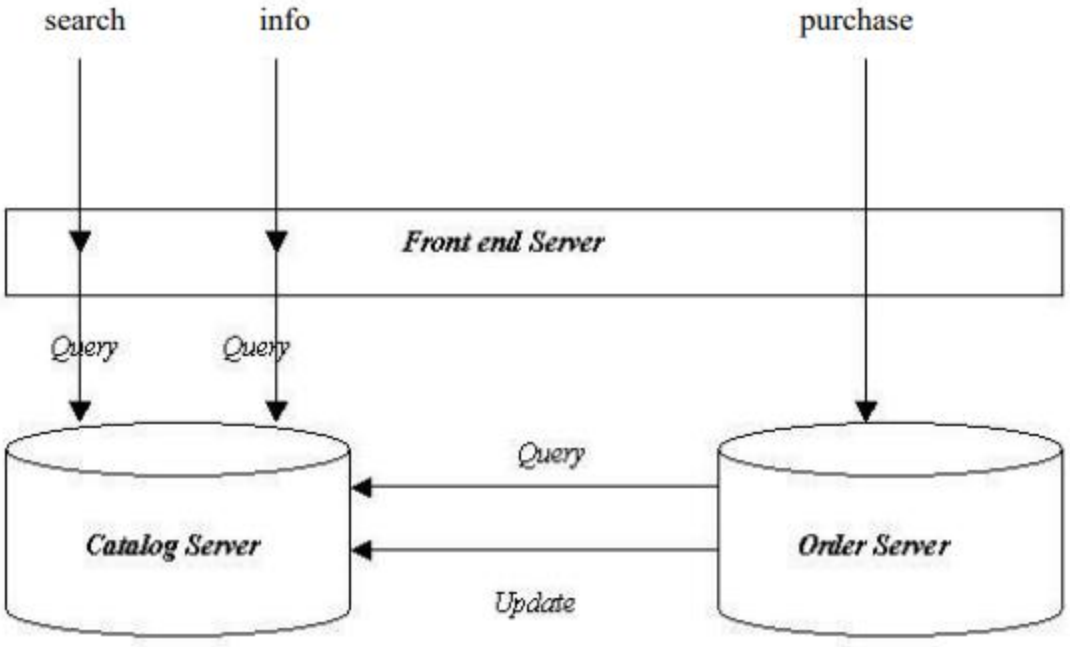

Figure 1

The three services communicate with each other using HTTP, where the frontend server acts as an API gateway as shown in figure 2, it takes the user request then forwards it to the appropriate server, order server if the user is ordering a book or the catalog server if the user Is searching about a book using the topic or the ID.

```
[HttpGet("{bookId}/info")]
0 references
public async Task<ActionResult<BookDto>> GetBookById(int bookId)
{
    try
    {
        HttpResponseMessage response = await _httpClient.GetAsync($"https://localhost:7093/api/catalog/{bookId}/info");

        if (response.IsSuccessStatusCode)
        {
            string responseBody = await response.Content.ReadAsStringAsync();

            return Ok(responseBody);
        }
        else
        {
            return StatusCode((int)response.StatusCode, "Error occurred while calling the external API.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred: {ex.Message}");
    }

}
```

Figure 2

If the user requests a search operation, the request is forwarded to the catalog server as mentioned above where it will be handled, then the response will be returned to the frontend server, then to the user.

If the user requests an order operation, the request is forwarded to the order server, where it will be handled no response will be returned, and the order server tells the catalog server about the purchase so it would decrease the quantity of the book purchased as in figure 3.

```
[HttpPost]
0 references
public async Task<ActionResult> Order(int bookId)
{
    var IsOrdered = await _orderRepository.PurchaseBook(bookId);
    if (IsOrdered)
    {

        try
        {
            HttpResponseMessage response = await _httpClient.PostAsync($"https://localhost:7093/api/catalog/{bookId}/purchase",null);

            if (response.IsSuccessStatusCode)
            {
                return Ok();
            }
            else
            {
                return StatusCode((int)response.StatusCode, "Error occurred while calling the external API.");
            }
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"An error occurred: {ex.Message}");
        }

    }
    else
    {
        return BadRequest("this book is out of stock.");
    }
```

Figure 3

this application is not complete where there are some points that I think would make it better.

Adding more data to the database, tracking the purchase operation and adding users to know who purchased what is create, it will give the user access to his purchase history, and it will give the owner of the application additional information helping him manage the application better.

Figure4 shows the search by topic request, figure5 shows the get book by id request, and figure6 shows the order request.

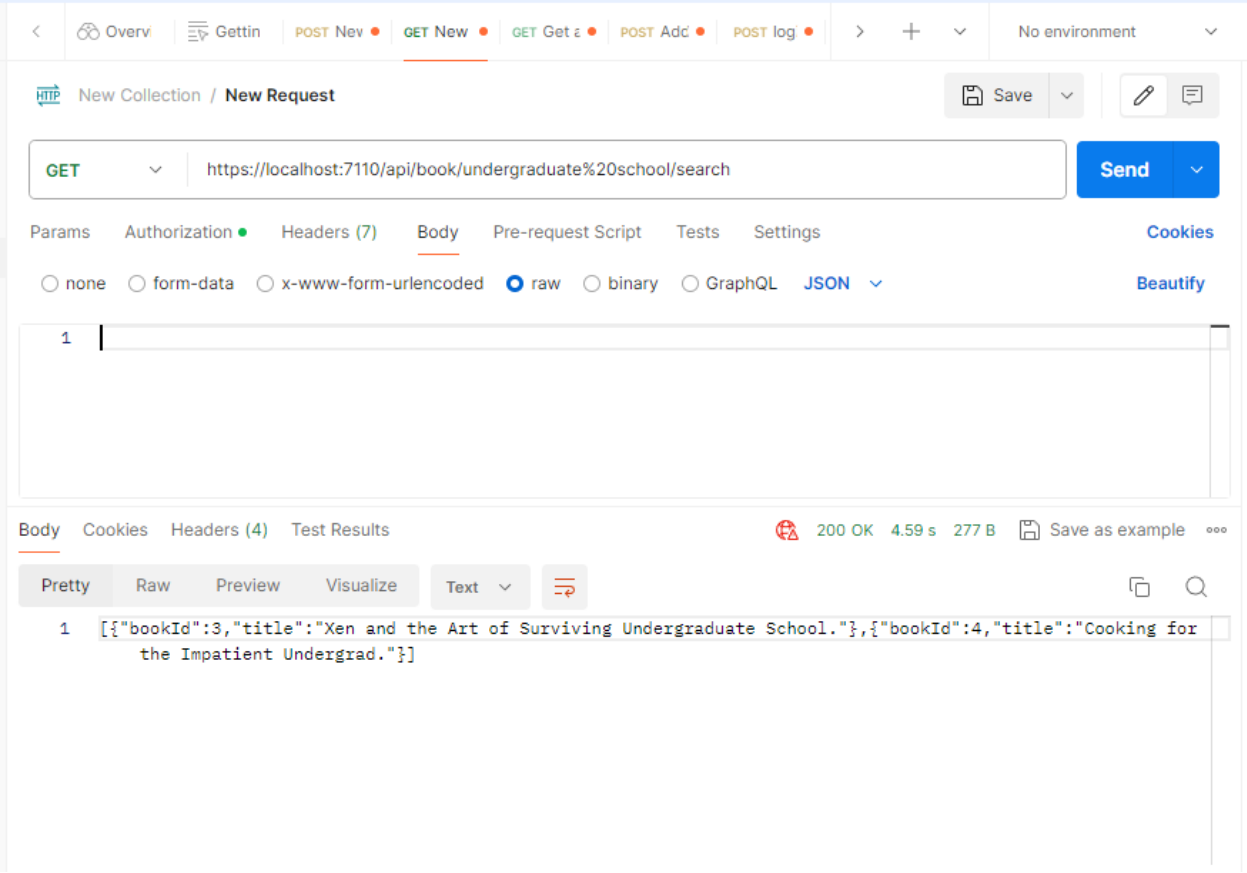

Figure 4

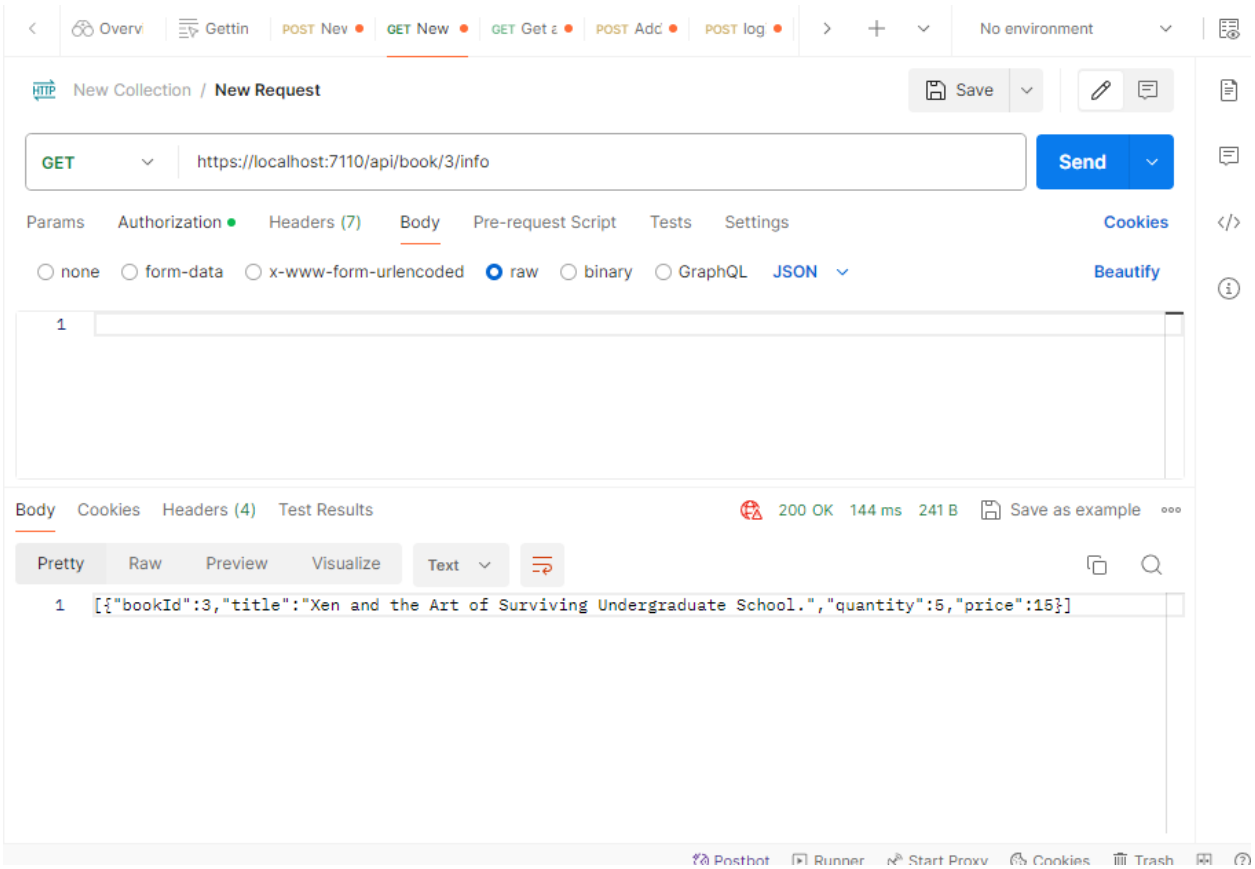

Figure 5

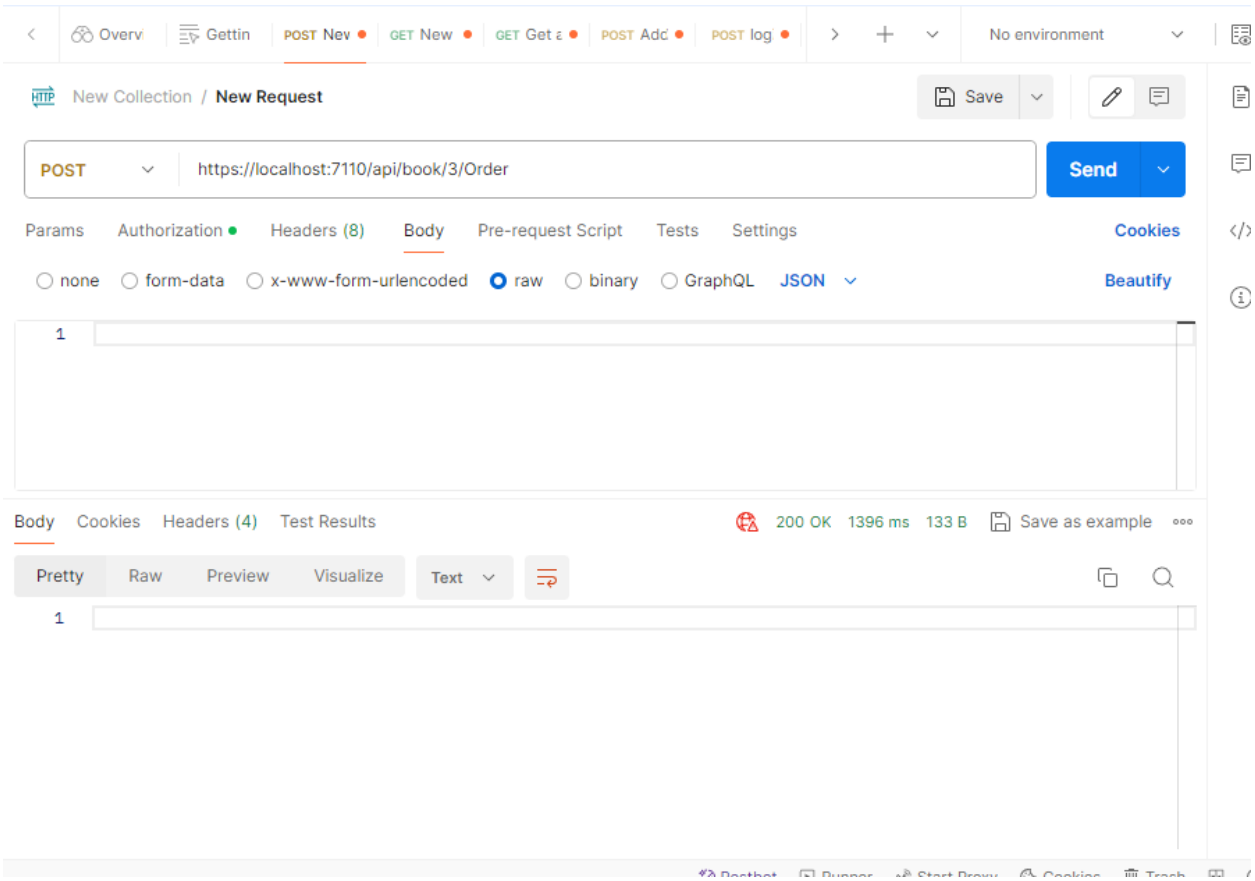

Figure 6