(Demo prep)

Integrating the repository with the controller

1. Add Extensions.cs:

```
public static class Extensions
{
    public static ItemDto AsDto(this Item item)
    {
        return new ItemDto(item.Id, item.Name, item.Description, item.Price, item.CreatedDate);
    }
}
```

2. Update ItemsController (CLOSE THE NAVIGATION PANE):

```
[ApiController]
[Route("items")]
public class ItemsController : ControllerBase
{
    private readonly ItemsRepository itemsRepository = new();

    [HttpGet]
    public async Task<ActionResult<IEnumerable<ItemDto>>> GetAsync()
    {
        var items = (await itemsRepository.GetAllAsync())
                    .Select(item => item.AsDto());

        return Ok(items);
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<ItemDto>> GetByIdAsync(Guid id)
    {
        var item = await itemsRepository.GetAsync(id);

        if (item == null)
        {
            return NotFound();
        }

        return item.AsDto();
    }
```

```
[HttpPost]
public async Task<ActionResult<ItemDto>> PostAsync(CreateItemDto createItemDto)
{
    var item = new Item
    {
        Name = createItemDto.Name,
        Description = createItemDto.Description,
        Price = createItemDto.Price,
        CreatedDate = DateTimeOffset.UtcNow
    };

    await itemsRepository.CreateAsync(item);

    return CreatedAtAction(nameof(GetByIdAsync), new { id = item.Id }, item.AsDto());
}

[HttpPut("{id}")]
public async Task<IActionResult> PutAsync(Guid id, UpdateItemDto updateItemDto)
{
    var existingItem = await itemsRepository.GetAsync(id);

    if (existingItem == null)
    {
        return NotFound();
    }

    existingItem.Name = updateItemDto.Name;
    existingItem.Description = updateItemDto.Description;
    existingItem.Price = updateItemDto.Price;

    await itemsRepository.UpdateAsync(existingItem);

    return NoContent();
}

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteAsync(Guid id)
{
    var item = await itemsRepository.GetAsync(id);

    if (item == null)
    {
        return NotFound();
```

```
            }

            await itemsRepository.RemoveAsync(item.Id);

            return NoContent();
        }
    }
```

3. Our REST API is ready to use our new ItemsRepository to store and query items in a MongoDB database. However, we still don't have a MongoDB database, or even a MongoDB server. That's a problem we can easily solve with Docker.

In the next lesson we will learn about Docker and how we can use it to run the infrastructure components needed by our microservices.