API Tasks

Create a new API Solution

1. Create an ASP.NET Core Web Application
    a. Choose the API template
    b. Don’t configure Https or Docker support

Add Swagger tooling (Open API Specification)

1. Add nuget package => Swashbuckle.AspNetCore
2. Configure Swagger Services => services.AddSwaggerGen(c => { c.SwaggerDoc("v1", new Info { Title = "Lists API", Version = "v1" });});
3. Enable Swagger => app.UseSwagger();
4. Configure Swagger UI => app.UseSwaggerUI(c => { c.SwaggerEndpoint("/swagger/v1/swagger.json", "Lists API V1"); });
5. Check Swagger UI => {root}/swagger/index.html

Links

Install Swagger https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-2.2&tabs=visual-studio

Create DocumentDB Repository

1. Add nuget package => Microsoft.Azure.DocumentDB.Core
2. Copy IDocumentDBRepository and DocumentDBRepository classes from quickstart UI solution
3. Copy Item.cs model from quick start UI solution
4. Configure IOC for Repo - services.AddSingleton<IDocumentDBRepository<Item>>(new DocumentDBRepository<Item>());

Create Items Controller

1. Create an Empty API Controller named ItemsController
2. Add Constructor to ItemsController

```
private readonly IDocumentDBRepository<Item> Respository;
public ItemsController(IDocumentDBRepository<Item> Respository)
{
   this.Respository = Respository;
}
```

3. Add REST Methods

```
[HttpGet]
public async Task<IEnumerable<Item>> GetAll()
{
   var items = await Respository.GetItemsAsync(d => !d.Completed);
```

```
    return items;
}

[HttpGet("{id}")]
public async Task<ActionResult> GetItem(string id)
{
    var items = await Respository.GetItemsAsync(x => x.Id == id);
    var item = items.FirstOrDefault();
    if (item == null)
        return NotFound();
    return Ok(item);
}

[HttpPost]
public async Task<ActionResult> Post([FromBody] Item value)
{
    var item = await Respository.CreateItemAsync(value);
    return Ok(item.Id);
}

[HttpPut("{id}")]
public async Task<ActionResult> Put(string id, [FromBody] Item value)
{
    await Respository.UpdateItemAsync(id, value);
    return Ok();

}

[HttpDelete("{id}")]
public async Task<ActionResult> Delete(string id)
{
    await Respository.DeleteItemAsync(id);
    return Ok();
}
```

4. Delete the Values Controller

Switch to using App Settings

1. Add settings to appsettings.json

```
 "CosmosDB": {
    "AccountEndpoint": "YOURENDPOINTHERE",
    "AccountKeys": "YOURKEYHERE",
    "Database": "ToDoList",
    "Collection": "Items"
  }
```

2. Change DocumentDBRepository constructor to take in the Cosmos Settings

```
public DocumentDBRepository(string endpoint, string key, string database, string collection)
{
   Endpoint = endpoint;
   Key = key;
   DatabaseId = database;
   CollectionId = collection;
```

3. Adjust IOC to pass in the cosmos settings into the constructor of the Repo from the Config (App Settings)

```
services.AddSingleton<IDocumentDBRepository<Item>>(new
DocumentDBRepository<Item>(Configuration["CosmosDB:AccountEndpoint"],
Configuration["CosmosDB:AccountKeys"],
          Configuration["CosmosDB:Database"], Configuration["CosmosDB:Collection"]));
```

5. Clear the defaults on the local variables of DocumentDBRepository

```
private readonly string Endpoint = "";
private readonly string Key = "";
private readonly string DatabaseId = "";
private readonly string CollectionId = "";
```

Publish the API

1. Create a new Web App in Azure for the API – Use the same APP Service Plan
2. Publish the API from Visual Studio

<u>UI Tasks – DON'T start this until your API is working in AZURE</u>

Switch application to use a new IRemoteRepository instead of the existing IDocumentDBRepository

1. Create a new IRemoteRepository

```
public interface IRemoteRepository<T> where T : class
{
   Task<string> CreateItemAsync(T item);
   Task<bool> DeleteItemAsync(string id);
   Task<T> GetItemAsync(string id);
   Task<IEnumerable<T>> GetItemsAsync(Expression<Func<T, bool>> predicate);
   Task<bool> UpdateItemAsync(string id, T item);
}
```

1. Create an Implementation of IRemoteRepository

```
public class RemoteRepository<T> : IRemoteRepository<T> where T : class
{
    public string BaseAPIUri { get; set; }
    public RemoteRepository(string baseUrl)
    {
        BaseAPIUri = baseUrl;
    }

    public async Task<string> CreateItemAsync(T item)
    {
        HttpContent requestContent = new StringContent(JsonConvert.SerializeObject(item),
Encoding.UTF8, "application/json");
        using (HttpClient client = new HttpClient())
        using (HttpResponseMessage res = await client.PostAsync(BaseAPIUri + "items",
requestContent))
        using (HttpContent content = res.Content)
        {
            if (res.IsSuccessStatusCode)
            {
                string id = await content.ReadAsStringAsync();
                return id;
            }
            else
                return "";
        }
    }

    public async Task<bool> DeleteItemAsync(string id)
    {
        using (HttpClient client = new HttpClient())
        using (HttpResponseMessage res = await client.DeleteAsync(BaseAPIUri + "items/" + id))
        {
            return res.IsSuccessStatusCode;
        }
    }

    public async Task<T> GetItemAsync(string id)
    {
        using (HttpClient client = new HttpClient())
        using (HttpResponseMessage res = await client.GetAsync(BaseAPIUri + "items/" + id))
        using (HttpContent content = res.Content)
        {
            string data = await content.ReadAsStringAsync();
            var item = JsonConvert.DeserializeObject<T>(data);
            return item;
        }
    }
    public async Task<IEnumerable<T>>
GetItemsAsync(System.Linq.Expressions.Expression<Func<T, bool>> predicate)
    {
        using (HttpClient client = new HttpClient())
        using (HttpResponseMessage res = await client.GetAsync(BaseAPIUri + "items"))
        using (HttpContent content = res.Content)
        {
            string data = await content.ReadAsStringAsync();
            var items = JsonConvert.DeserializeObject<IEnumerable<T>>(data);
            return items;
        }
    }

    public async Task<bool> UpdateItemAsync(string id, T item)
    {
        HttpContent requestContent = new StringContent(JsonConvert.SerializeObject(item),
Encoding.UTF8, "application/json");
        using (HttpClient client = new HttpClient())
        using (HttpResponseMessage res = await client.PutAsync(BaseAPIUri + "items/" + id,
requestContent))
        {
            return res.IsSuccessStatusCode;
        }
    }
}
```