

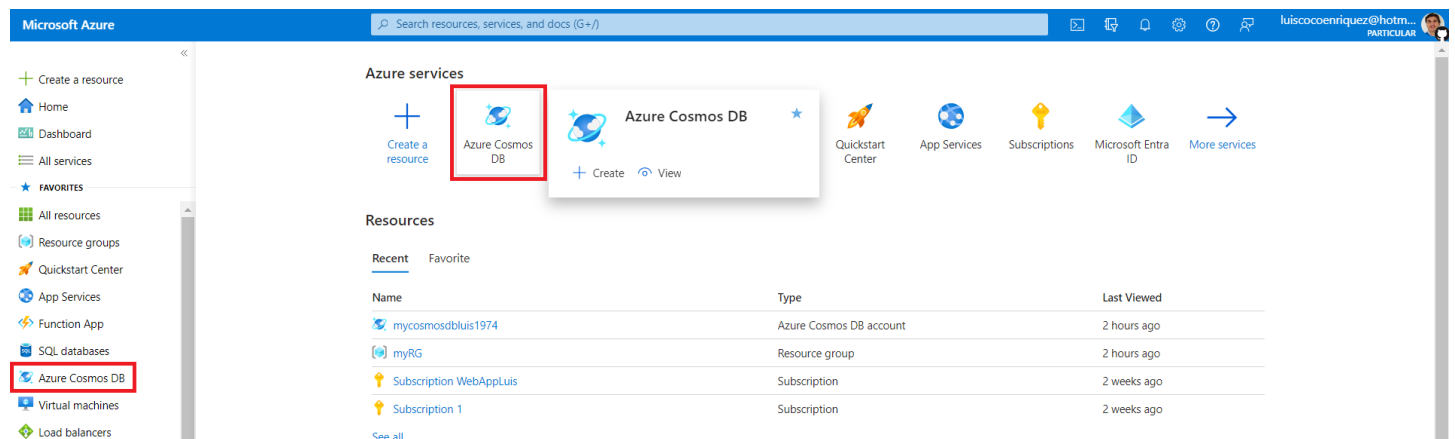
How to create a .NET8 WebAPI CRUD Azure CosmosDB Microservice

The code for this example is available in this github repo:

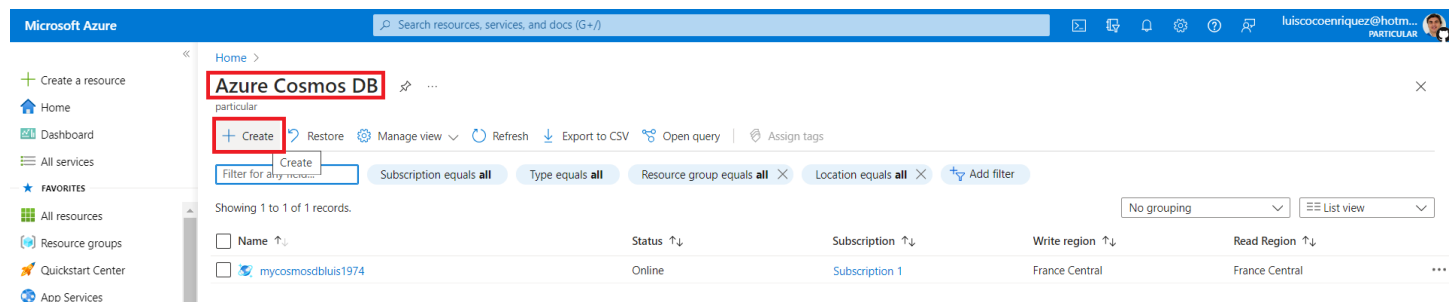
https://github.com/luisccoco/MicroServices_dotNET8_CRUD_WebAPI-AzureCosmosDB

1. Create Azure CosmosDB

We navigate to **Azure CosmosDB** service and we click in this option to create a new account:



We click on **Azure CosmosDB** account button



Now we select the option **Azure Cosmos DB for NoSQL**, and we press the **create** button

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create an Azure Cosmos DB account

Which API best suits your workload?

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. [Learn more](#)

To start, select the API to create a new account. The API selection cannot be changed after account creation.

Azure Cosmos DB for NoSQL

Azure Cosmos DB's core, or native API for working with documents. Supports fast, flexible development with familiar SQL query language and client libraries for .NET, JavaScript, Python, and Java.

[Create](#) [Learn more](#)

Azure Cosmos DB for PostgreSQL

Fully-managed relational database service for PostgreSQL with distributed query execution, powered by the Citus open source extension. Build new apps on single or multi-node clusters—with support for JSONB, geospatial, rich indexing, and high-performance scale-out.

[Create](#) [Learn more](#)

Azure Cosmos DB for MongoDB

Fully managed database service for apps written for MongoDB. Recommended if you have existing MongoDB workloads that you plan to migrate to Azure Cosmos DB.

[Create](#) [Learn more](#)

Azure Cosmos DB for Apache Cassandra

Fully managed Cassandra database service for apps written for Apache Cassandra. Recommended if you have existing Cassandra workloads that you plan to migrate to Azure Cosmos DB.

[Create](#) [Learn more](#)

Azure Cosmos DB for Table

Fully managed database service for apps written for Azure Table storage. Recommended if you have existing Azure Table storage workloads that you plan to migrate to Azure Cosmos DB.

[Create](#) [Learn more](#)

Azure Cosmos DB for Apache Gremlin

Fully managed graph database service using the Gremlin query language, based on Apache TinkerPop project. Recommended for new workloads that need to store relationships between data.

[Create](#) [Learn more](#)

In following screen we input the required data for creating the service

We create a new **ResourceGroup** name: myRG

We set the **account name**: mycosmosdbluis1974

We choose the service **location**: France Central

Capacity mode: **serverless**

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Azure Cosmos DB >

Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL

Basics Global Distribution Networking Backup Policy Encryption Tags Review + create

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource Group * [Create new](#)

Instance Details

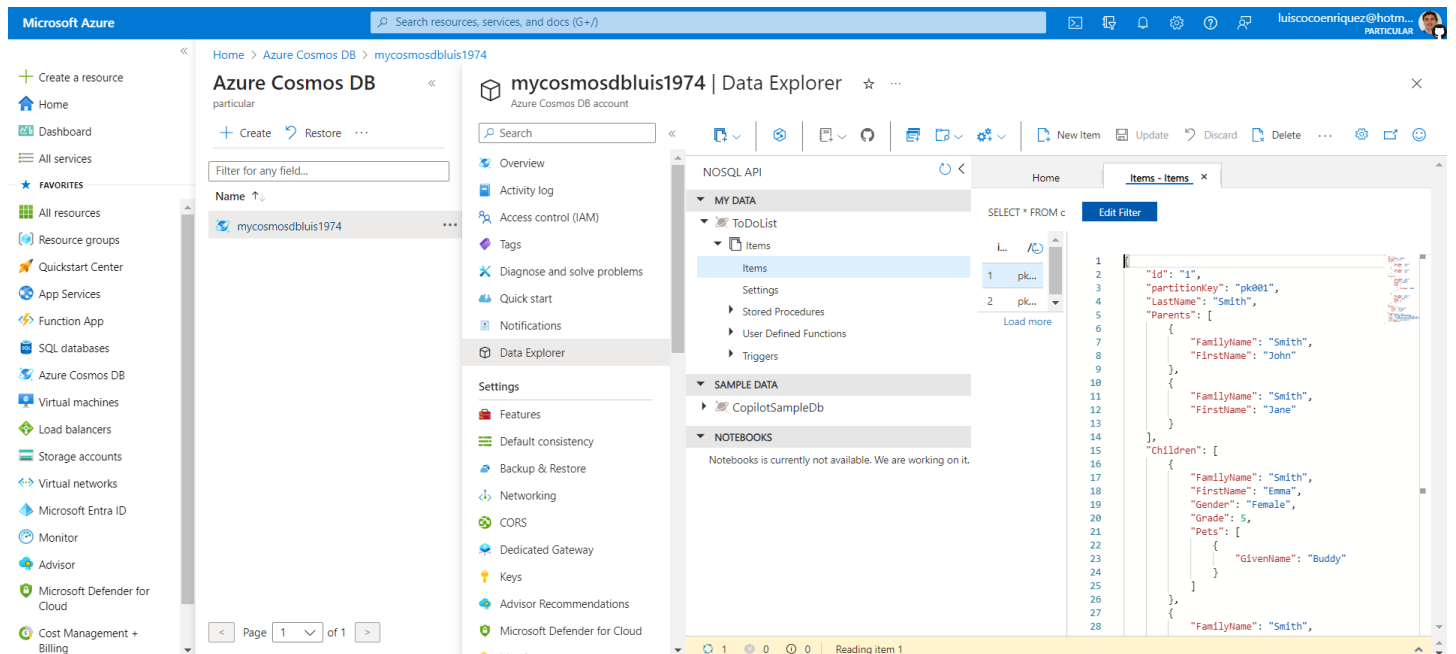
Account Name * ✓

Location *

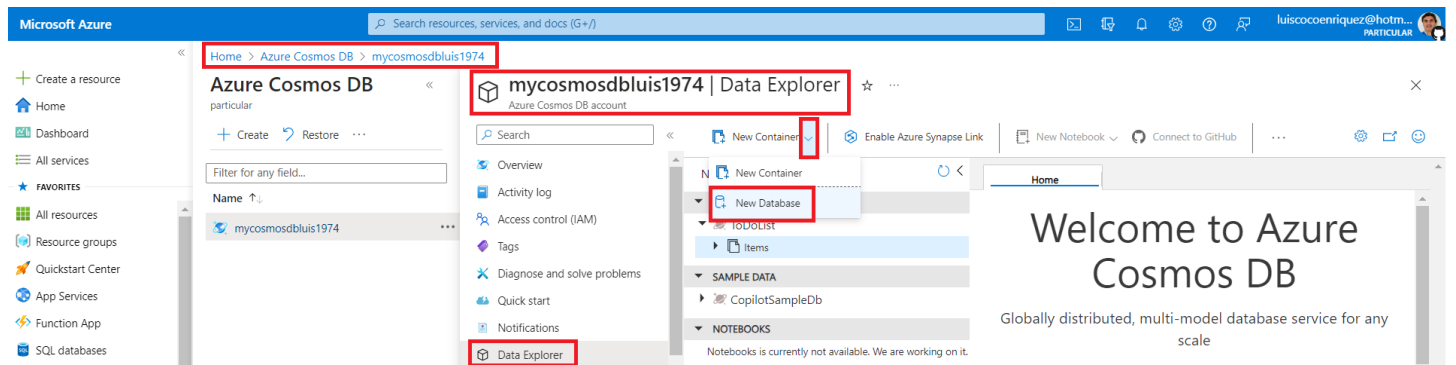
Capacity mode ⓘ ☐ Provisioned throughput ☒ Serverless [Learn more about capacity mode](#)

[Review + create](#) [Previous](#) [Next: Global Distribution](#)

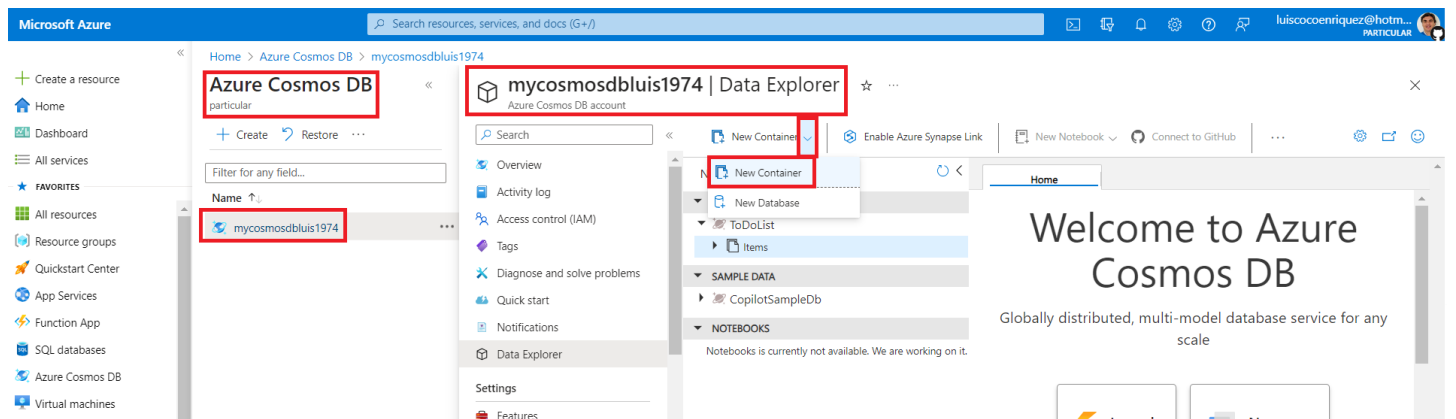
We navigate to the **Data Explorer** page and we create a **New Database** and a **New Container**



We first create a New Database. We input the Databaseld



We also create a New Container



2. Insert the new items in the Azure CosmosDB

This is the new item json file:

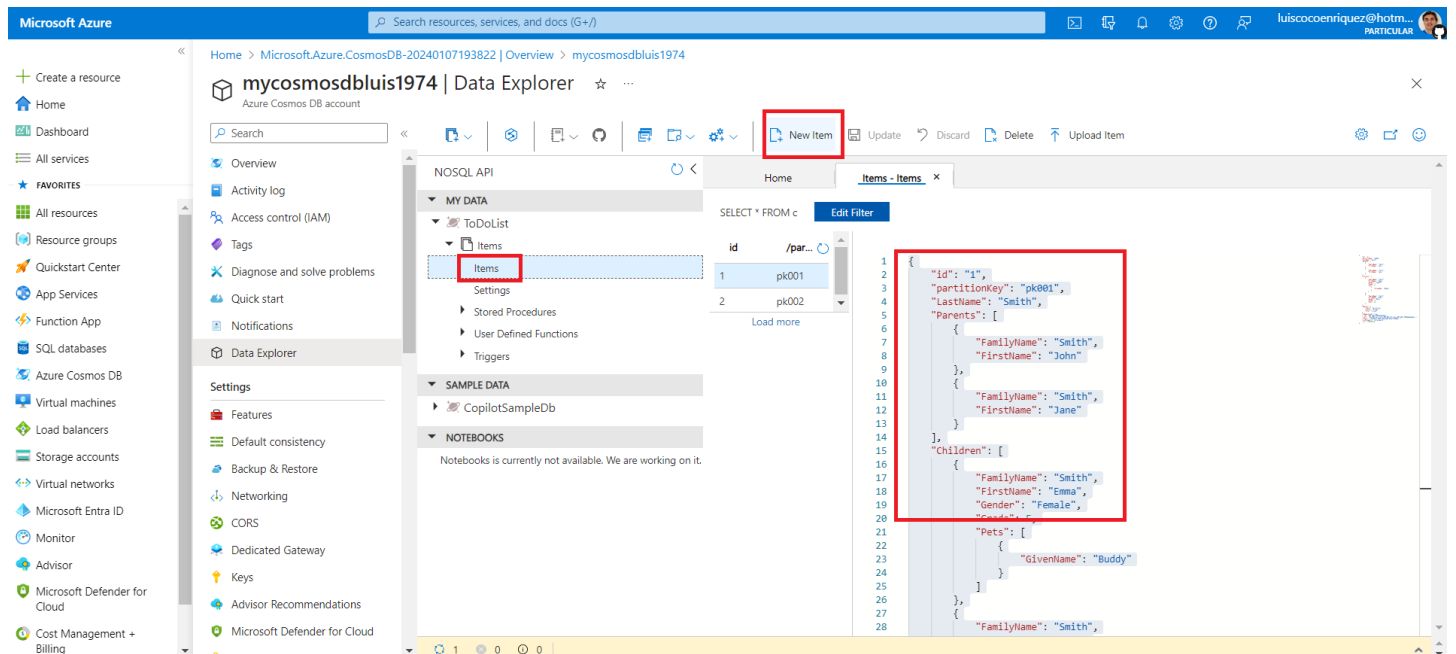
```

{
  "id": "1",
  "partitionKey": "pk001",

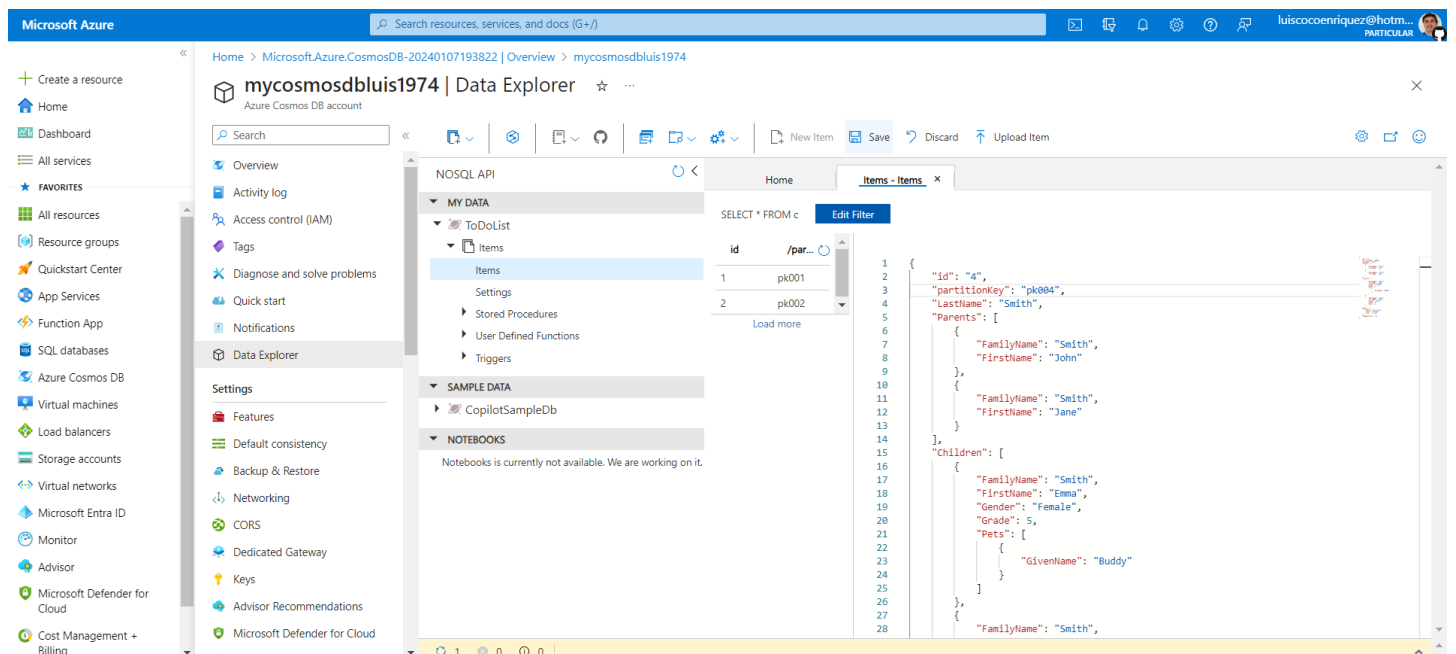
```

```
"LastName": "Smith",
"Parents": [
  {
    "FamilyName": "Smith",
    "FirstName": "John"
  },
  {
    "FamilyName": "Smith",
    "FirstName": "Jane"
  }
],
"Children": [
  {
    "FamilyName": "Smith",
    "FirstName": "Emma",
    "Gender": "Female",
    "Grade": 5,
    "Pets": [
      {
        "GivenName": "Buddy"
      }
    ]
  },
  {
    "FamilyName": "Smith",
    "FirstName": "Mike",
    "Gender": "Male",
    "Grade": 8,
    "Pets": []
  }
],
"Address": {
  "State": "California",
  "County": "Orange",
  "City": "Irvine"
},
"IsRegistered": true
}
```

We click in **Items** and then **New Item**



Then we copy and paste the json content and press Save button



3. appsettings.json

We copy the Databaseld and ContainerId

Databaseld: ToDoList

ContainerId: Items

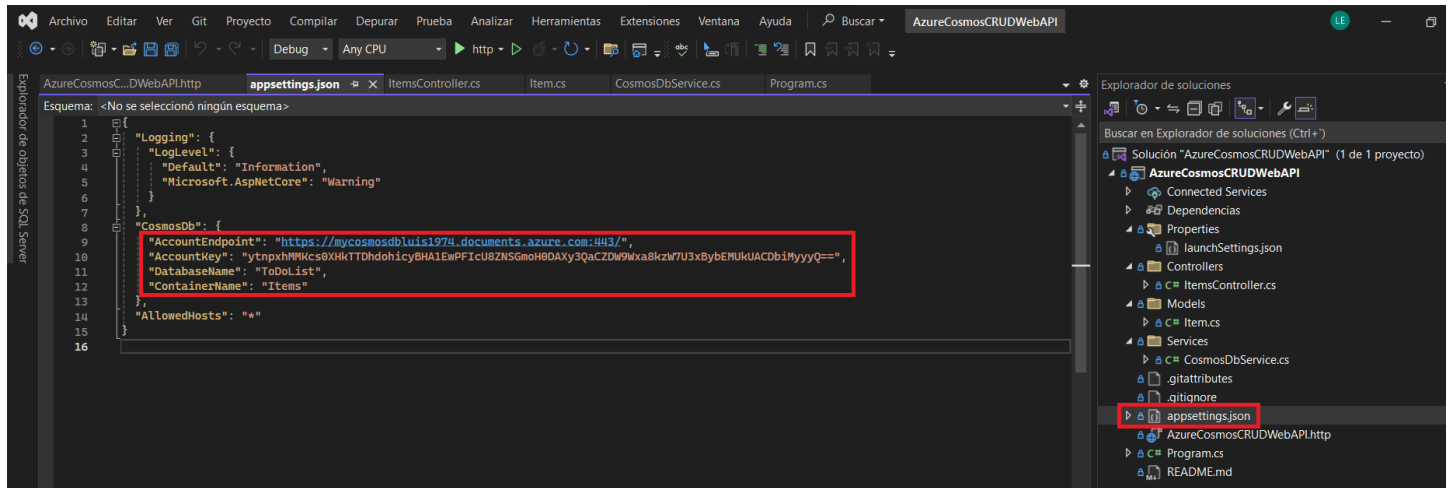
The screenshot shows the Microsoft Azure portal interface. On the left, the 'Data Explorer' tab is selected for the 'mycosmosdbluis1974' Azure Cosmos DB account. The 'Items' folder is expanded, and a sample item is shown. A red box highlights the 'id' and 'partitionKey' fields in the sample item, which are 'pk001' and 'pk001' respectively. The sample item also includes fields for 'LastName', 'FirstName', 'Parents', 'Children', and 'Pets'.

We copy the URI and Primary Key

The screenshot shows the 'Keys' page for the 'mycosmosdbluis1974' Azure Cosmos DB account. The 'URI' field is highlighted with a red box, showing the value 'https://mycosmosdbluis1974.documents.azure.com:443/'. The 'PRIMARY KEY' field is also highlighted with a red box, showing a long alphanumeric string. The 'Read-write Keys' and 'Read-only Keys' tabs are visible at the top.

This screenshot is identical to the previous one, showing the 'Keys' page for the 'mycosmosdbluis1974' Azure Cosmos DB account. The 'URI' and 'PRIMARY KEY' fields are highlighted with red boxes.

This is the appsettings.json file



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "CosmosDb": {
    "AccountEndpoint": "https://mycosmosdbbluis1974.documents.azure.com:443/",
    "AccountKey": "ytnpxhMMKcs0XHkTTDhdohicyBHA1EwPFicU8ZNSGmoH0DAXy3QaCZDW9Wxa8kzW7U3xBybEMUK",
    "DatabaseName": "ToDoList",
    "ContainerName": "Items"
  },
  "AllowedHosts": "*"
}
```

4. Program.cs

```
using Microsoft.Azure.Cosmos;
using AzureCosmosCRUDWebAPI.Services;
using Microsoft.Extensions.Configuration;
using Microsoft.AspNetCore.Diagnostics;
using Newtonsoft.Json;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();

// Cosmos DB Configuration
var cosmosDbConfig = builder.Configuration.GetSection("CosmosDb");
builder.Services.AddSingleton<CosmosClient>(s =>
    new CosmosClient(cosmosDbConfig["AccountEndpoint"], cosmosDbConfig["AccountKey"]));
builder.Services.AddSingleton<CosmosDbService>(s =>
    new CosmosDbService(s.GetRequiredService<CosmosClient>(), cosmosDbConfig["DatabaseName"],
```

```
// Add other necessary services like Swagger if needed
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseExceptionHandler(a => a.Run(async context =>
{
    var exceptionHandlerPathFeature = context.Features.Get<IExceptionHandlerPathFeature>();
    var exception = exceptionHandlerPathFeature?.Error;

    var result = JsonConvert.SerializeObject(new { error = exception?.Message });
    context.Response.ContentType = "application/json";
    await context.Response.WriteAsync(result);
})));

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

5. Models

Item.cs

```
using Newtonsoft.Json;

namespace AzureCosmosCRUDWebAPI.Models
{
    public class Family
    {
        [JsonProperty(PropertyName = "id")]
        public string Id { get; set; }
        [JsonProperty(PropertyName = "partitionKey")]
        public string PartitionKey { get; set; }
        public string LastName { get; set; }
        public Parent[] Parents { get; set; }
        public Child[] Children { get; set; }
        public Address Address { get; set; }
        public bool IsRegistered { get; set; }
    }
}
```



```
        public override string ToString()
        {
            return JsonConvert.SerializeObject(this);
        }
    }

    public class Parent
    {
        public string FamilyName { get; set; }
        public string FirstName { get; set; }
    }

    public class Child
    {
        public string FamilyName { get; set; }
        public string FirstName { get; set; }
        public string Gender { get; set; }
        public int Grade { get; set; }
        public Pet[] Pets { get; set; }
    }

    public class Pet
    {
        public string GivenName { get; set; }
    }

    public class Address
    {
        public string State { get; set; }
        public string County { get; set; }
        public string City { get; set; }
    }
}
```

6. CosmosDbService.cs

Pay attention we set the **PartitionKey** `"/Id"`

```
using Microsoft.Azure.Cosmos;
using AzureCosmosCRUDWebAPI.Models;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Linq;
using Newtonsoft.Json;

namespace AzureCosmosCRUDWebAPI.Services
{
    public class CosmosDbService
    {

```

```

private Container _container;

public CosmosDbService(CosmosClient dbClient, string databaseName, string containerName)
{
    this._container = dbClient.GetContainer(databaseName, containerName);
}

public async Task AddItemAsync(Family item)
{
    try
    {
        await this._container.CreateItemAsync(item, new PartitionKey(item.PartitionKey)
    }
    catch (CosmosException ex)
    {
        Console.WriteLine($"Cosmos DB error in AddItemAsync. Status code: {ex.StatusCode}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in AddItemAsync: {ex.Message}, StackTrace: {ex.StackTrace}");
        throw;
    }
}

// Read an item by id
public async Task<Family> GetItemAsync(string id, string partitionKeyValue)
{
    try
    {
        ItemResponse<Family> response = await this._container.ReadItemAsync<Family>(id, partitionKeyValue);
        return response.Resource;
    }
    catch (CosmosException ex) when (ex.StatusCode == System.Net.HttpStatusCode.NotFound)
    {
        return null;
    }
}

// Update an existing item
public async Task UpdateItemAsync(string id, Family item)
{
    await this._container.UpsertItemAsync(item, new PartitionKey(id));
}

// Delete an item
public async Task DeleteItemAsync(string id, string partitionKeyValue)
{
    await this._container.DeleteItemAsync<Family>(id, new PartitionKey(partitionKeyValue));
}

// List all items
public async Task<IEnumerable<Family>> GetItemsAsync(string queryString)

```

```

    {
        var query = this._container.GetItemQueryIterator<Family>(new QueryDefinition(query));
        List<Family> results = new List<Family>();
        while (query.HasMoreResults)
        {
            var response = await query.ReadNextAsync();
            results.AddRange(response.ToList());
        }
        return results;
    }
}
}

```

7. Controller

ItemsController.cs

```

using Microsoft.AspNetCore.Mvc;
using AzureCosmosCRUDWebAPI.Models;
using AzureCosmosCRUDWebAPI.Services;
using System.Threading.Tasks;

namespace AzureCosmosCRUDWebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class FamilyController : ControllerBase
    {
        private readonly CosmosDbService _cosmosDbService;

        public FamilyController(CosmosDbService cosmosDbService)
        {
            _cosmosDbService = cosmosDbService;
        }

        // POST api/family
        [HttpPost]
        public async Task<IActionResult> Create([FromBody] Family family)
        {
            if (family == null)
            {
                return BadRequest("Family cannot be null");
            }

            await _cosmosDbService.AddItemAsync(family);
            return CreatedAtAction(nameof(Get), new { id = family.Id }, family);
        }

        // GET api/family/{id}
        [HttpGet("{id}")]
    }
}

```

```
public async Task<IActionResult> Get(string id, string partitionKeyValue)
{
    var family = await _cosmosDbService.GetItemAsync(id, partitionKeyValue);
    if (family == null)
    {
        return NotFound();
    }

    return Ok(family);
}

// PUT api/family/{id}
[HttpPut("{id}")]
public async Task<IActionResult> Update(string id, [FromBody] Family family)
{
    if (family == null || family.Id != id)
    {
        return BadRequest();
    }

    await _cosmosDbService.UpdateItemAsync(id, family);
    return NoContent();
}

// DELETE api/family/{id}
[HttpDelete("{id}")]
public async Task<IActionResult> Delete(string id, string partitionKeyValue)
{
    await _cosmosDbService.DeleteItemAsync(id, partitionKeyValue);
    return NoContent();
}

// GET api/family
[HttpGet]
public async Task<IActionResult> GetAll()
{
    var families = await _cosmosDbService.GetItemsAsync("SELECT * FROM c");
    return Ok(families);
}
}
```

8. Application verification

Swagger UI interface showing the API definition for **AzureCosmosCRUDWebAPI** (1.0 OAS3). The API is titled **Family**. The endpoints listed are:

- POST** /api/Family
- GET** /api/Family
- GET** /api/Family/{id}
- PUT** /api/Family/{id}
- DELETE** /api/Family/{id}

Swagger UI interface showing the execution of the **GET /api/Family** endpoint. The response body is displayed as a JSON object:

```
{
  "id": "1",
  "partitionKey": "pk001",
  "lastName": "Smith",
  "parents": [
    {
      "familyName": "Smith",
      "firstName": "John"
    }
  ]
}
```

The image shows the Swagger UI interface for a .NET8 WebAPI. The endpoint is `GET /api/Family/{id}`. The parameters are:

- `id` (required, string, path): `1`
- `partitionKeyValue` (string, query): `pk001`

The response is a JSON object representing a family member:

```
{
  "id": "1",
  "partitionKey": "pk001",
  "lastName": "Smith",
  "parents": [
    {
      "familyName": "Smith",
      "firstName": "John"
    }
  ]
}
```