AF Function setup

|  |  |
| --- | --- |
| Install NPM | <https://nodejs.org/en/download/> |
| Install .net core | <https://dotnet.microsoft.com/download/dotnet-core/thank-you/sdk-3.1.402-windows-x64-installer> |
| Install postman | <https://www.postman.com/> |
| Install VS Code | <https://code.visualstudio.com/Download> |
| Default settings are OK |  |
| Install azure function core tools | <https://go.microsoft.com/fwlink/?linkid=2135274> |
| Install Azure Functions extension |  |
| Install c# extension |  |
| Close vs code | App closes |
| Test Policy (optional) |  |
| Test policy – from PowerShell | Get-ExecutionPolicy -List |
| Set policy – from PowerShell | Set-ExecutionPolicy Unrestricted -Scope CurrentUser -Force |
|  |  |
|  |  |

Build data store

|  |  |
| --- | --- |
| Search for Cosmos |  |
| Select your resource group |  |
| Create account name |  |
| Assure sql core |  |
| Let it cook @ 350 for 10 min |  |
|  |  |
|  |  |

Create Project and test

|  |  |
| --- | --- |
| Open cmd |  |
| Go to root | Cd\ |
| Make projects | Md Projects |
| Change dir to projects | Cd projects |
| Make backend folder | Md CN |
| Change dir to backend | Cd CN |
| Open vs code | code . |
| Open Command pallet |  |
| Create new project |  |
| Select your newly created folder |  |
| Select your language choice (C#) |  |
| Select http trigger |  |
| Enter “Bike” |  |
| Can leave company name |  |
| Leave as “Function” |  |
| Run | Press F5 |
| Get URL to Bike |  |
| Open postman |  |
| Cut and paste url in postman and run |  |

Create container

|  |  |
| --- | --- |
| Navigate to your cosmos instance |  |
| Navigate to Data explorer |  |
| Create new container/database |  |
| Settings |  |

Setup cosmos

|  |  |
| --- | --- |
| Open VS Code |  |
| Open your project file |  |
| Add the cosmos package to the itemgroup | <PackageReference Include="Microsoft.Azure.Cosmos"   Version="3.14.0" /> |
| Create a new file name DAL.cs |  |
| Add references to system and cosmos | using System;  using Microsoft.Azure.Cosmos;  using System.Threading.Tasks;  using System.Collections.Generic; |
| Create a DAL class | public class DAL {} |
| Add variables to hold settings | public const string COSMOS\_CONTAINER = "bike";   public const string COSMOS\_BIKE\_TABLE = "bike";   private const string COSMOS\_CONNECTION\_STRING = "YOURS"; |
| Initialize the lazy loaded static cosmos client | private static Lazy<CosmosClient> m\_lcdb = new Lazy<CosmosClient>(InitializeCosmosClient);      private static CosmosClient m\_cdb => m\_lcdb.Value;      private static CosmosClient InitializeCosmosClient()      {          return new CosmosClient(DAL.COSMOS\_CONNECTION\_STRING);      } |
| Create bike method | public static async Task CreateBikeAsync(Models.Bike Bike)      {          try          {              Container c = m\_cdb.GetContainer(DAL.COSMOS\_CONTAINER, DAL.COSMOS\_BIKE\_TABLE);              await c.CreateItemAsync<Models.Bike>(Bike);          }          catch (Exception ex)          {          }      } |
| Create a new folder for Models |  |
| Create new file Bike.cs **In models folder** |  |
| Create Bike model | using Newtonsoft.Json;  namespace Models  {    public  class Bike      {          [JsonProperty(PropertyName = "id")]          public string ID { get; set; }          public string Model {get;set;}          public string Make {get;set;}          public double Price {get;set;}          public int Quantity {get;set;}      }  } |

Test create bike

|  |  |
| --- | --- |
| Open Bike.cs app | Bike.cs |
| Add create bike code (toward top of code, below logging information) | Models.Bike b = new Models.Bike();  b.ID = Guid.NewGuid().ToString();  b.Make = "Specialized";  b.Model = "FSR";  b.Price = 10000;  b.Quantity =10;  await DAL.CreateBikeAsync(b); |
| Open postman |  |
| Run project then call in postman |  |
| Open cosmos data explorer from Azure portal |  |
| Expand container |  |
| Should have an item |  |

Complete Bike service

|  |  |
| --- | --- |
| Add Get method | public static async Task<Models.Bike>  GetBikeByIDAsync(string ID, string Partition)      {          Models.Bike bkReturn = null;            try          {              Container c =  m\_cdb.GetContainer(DAL.COSMOS\_CONTAINER, DAL.COSMOS\_BIKE\_TABLE);                bkReturn =  await c.ReadItemAsync<Models.Bike>(ID, new PartitionKey(Partition));          }          catch (Exception ex)          {          }          return bkReturn;      } |
| Add Get All method | public static async Task<Models.Bike[]> GetAllBikesAsync()  {      List<Models.Bike> abk = new List<Models.Bike>();      try      {          Container c = m\_cdb.GetContainer(DAL.COSMOS\_CONTAINER, DAL.COSMOS\_BIKE\_TABLE);          var sqlQueryText = "SELECT \* FROM c";          QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);          FeedIterator<Models.Bike> queryResultSetIterator = c.GetItemQueryIterator<Models.Bike>(queryDefinition);          while (queryResultSetIterator.HasMoreResults)          {              FeedResponse<Models.Bike> currentResultSet  = await queryResultSetIterator.ReadNextAsync();              foreach (Models.Bike b in currentResultSet)                  abk.Add(b);          }      }      catch (Exception ex)      {      }      return abk.ToArray();  } |
| Add Save method | public static async Task SaveBikeAsync(Models.Bike Bike)  {      try      {          Container c = m\_cdb.GetContainer(DAL.COSMOS\_CONTAINER, DAL.COSMOS\_BIKE\_TABLE);          await c.ReplaceItemAsync<Models.Bike>(Bike, Bike.ID);      }      catch (Exception exError)      { }  } |
| Complete Bike Service (replace all code) | switch (req.Method.ToString())  {      case "GET":          {              Models.Bike bk = new Models.Bike();              try              {                  string strID = req.Query["ID"];                  string strPartition                                = req.Query["Partition"];                  bk = await DAL.GetBikeByIDAsync(strID, strPartition);              }              catch (Exception exError)              {                  return new StatusCodeResult(500);              }              return new JsonResult(bk);          }      case "POST":          {              try              {                  string strBody = await new StreamReader(req.Body).ReadToEndAsync();                  Models.Bike bk = JsonConvert.DeserializeObject<Models.Bike>(strBody);                  if (bk.ID == null || bk.ID == string.Empty)                  {                      bk.ID = Guid.NewGuid().ToString();                      await DAL.CreateBikeAsync(bk);                  }                  else                      await DAL.SaveBikeAsync(bk);              }              catch (Exception exError)              {                  return new StatusCodeResult(500);              }              break;          }  }  return new OkResult(); |

Create Bikes service

|  |  |
| --- | --- |
| Create Bikes.cs http function |  |
| Implement service by replacing all code | Models.Bike[] abk;  try  {      abk = await DAL.GetAllBikesAsync();  }  catch (Exception exError)  {      return new StatusCodeResult(500);  }  return new JsonResult(abk); |

Test Service

|  |  |
| --- | --- |
| Open postman |  |
| Test Get Bike |  |
| Test Post Bike with ID (update) |  |
| Test Post Bike no ID (create) |  |
| Test Get Bikes |  |

Deploy Service

|  |  |
| --- | --- |
| In VS Code Login to azure with azure sign in with command palette |  |
| Create function app in azure with command palette |  |
| Select subscription |  |
| Enter a unique name (very unique) | \*\*\*\* |
| Select region |  |
| Deploy to function app in command pallet | * Select subscription * Select App created above |
| Open the azure pane |  |
| Navigate to the Bike and Bikes function and copy the function URL for both services |  |
| Test | You can now run the tests from the “Test Service” section with those URLS. Please note the ?code=XXX. You need to make sure you include that token. |

Enable Frontend

|  |  |
| --- | --- |
| Open Azure portal, navigate to your function app |  |
| Select Platform features |  |
| Select CORS |  |
| Add <http://localhost:8080> and save |  |