[01-01-2024 15:43] shivani: using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.Cosmos;
using TaskAPI.DTO;
using Container = Microsoft.Azure.Cosmos.Container;

namespace TaskAPI.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class taskController : ControllerBase
    {
        public string URI = "https://localhost:8081";
        public string PrimaryKey = "C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==";
        public string DatabaseName = "TaskDB";
        public string ContainerName = "TaskManager";

        public Container container1;

        public taskController()
        {
            container1 = GetContainer();
        }

        [HttpPost]
        public async Task<IActionResult> AddEmployee(empDTO empDTO)
        {
            try
            {
                Employee employeeEntity = new Employee();

                employeeEntity.TaskName = empDTO.TaskName;
                employeeEntity.TaskDescription = empDTO.TaskDescription;

                employeeEntity.Id = Guid.NewGuid().ToString();
                employeeEntity.UId = employeeEntity.Id;
                employeeEntity.DocumentType = "Employee";

                employeeEntity.CreatedOn = DateTime.Now;
                employeeEntity.CreatedByName = "shivani";
                employeeEntity.CreatedBy = "shivani’s UId";

                employeeEntity.UpdateOn = DateTime.Now;
                employeeEntity.UpdateByName = "shivani";
                employeeEntity.UpdateBy = "shivani’s UId";

                employeeEntity.Version = 1;
                employeeEntity.Active = true;
                employeeEntity.Archieved = false;

```
            Employee resposne = await container1.CreateItemAsync(employeeEntity);

            // Reverse MApping
            empDTO.TaskName = resposne.TaskName;
            empDTO.TaskDescription = resposne.TaskDescription;



            return Ok(empDTO);
        }
        catch (Exception ex)
        {

            return BadRequest("Data Adding Failed" + ex);
        }
    }

    [HttpPut]
    public async Task<IActionResult> UpdateItem(string uId, string name, string taskDesc)
    {

        Employee existingTask = container1.GetItemLinqQueryable<Employee>(true).Where(q => q.Docu
mentType == "Employee" && q.UId == uId).AsEnumerable().FirstOrDefault();
        if (existingTask != null)
        {
            existingTask.TaskName = name;
            existingTask.TaskDescription = taskDesc;
            existingTask.Version++;

            try
            {
                var response = await container1.UpsertItemAsync(existingTask, new PartitionKey(uId));
                if (response.StatusCode == System.Net.HttpStatusCode.OK || response.StatusCode == Syst
em.Net.HttpStatusCode.Created)
                {
                    return Ok("Task Updated Successfully");
                }
                else
                {
                    return BadRequest("Failed to Update Task");
                }

            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }

        }
        return BadRequest();
    }
    [HttpGet]
    public IActionResult GetemployeeByUId(string uId)
    {
        try
```

```
        {
            Employee tasks = container1.GetItemLinqQueryable<Employee>(true).Where(q => q.Documen
tType == "Employee" && q.UId == uId).AsEnumerable().FirstOrDefault();

            var taskModel = new empDTO();
            taskModel.TaskName = tasks.TaskName;
            taskModel.TaskDescription = tasks.TaskDescription;
            return Ok(taskModel);
        }
        catch (Exception ex)
        {
            return BadRequest("Data Get Failed");
        }
    }
    [HttpGet]
    public IActionResult GetAllEmployee()
    {
        try
        {

            var listresponse = container1.GetItemLinqQueryable<Employee>(true).AsEnumerable().ToList()
;
            return Ok(listresponse);

        }
        catch (Exception ex)
        {

            return BadRequest("Data Get Failed");
        }
    }

    [HttpDelete]
    public async Task DeleteTaskAsync(string uId)
    {
        await container1.DeleteItemAsync<Employee>(uId, new PartitionKey(uId));
    }
    private Container GetContainer()
    {
        CosmosClient cosmoscClient = new CosmosClient(URI, PrimaryKey);
        Database database = cosmoscClient.GetDatabase(DatabaseName);
        Container container = database.GetContainer(ContainerName);
        return container;
    }

  }


}
```

[01-01-2024 15:44] shivani:
```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.Cosmos;
using TaskAPI.DTO;
```

```
using Container = Microsoft.Azure.Cosmos.Container;



namespace TaskAPI.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class taskController : ControllerBase
    {
        public string URI = "https://localhost:8081";
        public string PrimaryKey = "C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqo
bD4b8mGGyPMbIZnqyMsEcaGQy67Xlw/Jw==";
        public string DatabaseName = "TaskDB";
        public string ContainerName = "TaskManager";

        public Container container1;

        public taskController()
        {
            container1 = GetContainer();
        }

        [HttpPost]
        public async Task<IActionResult> AddEmployee(empDTO empDTO)
        {
            try
            {
                Employee employeeEntity = new Employee();

                employeeEntity.TaskName = empDTO.TaskName;
                employeeEntity.TaskDescription = empDTO.TaskDescription;



                employeeEntity.Id = Guid.NewGuid().ToString();
                employeeEntity.UId = employeeEntity.Id;
                employeeEntity.DocumentType = "Employee";


                employeeEntity.CreatedOn = DateTime.Now;
                employeeEntity.CreatedByName = "shivani";
                employeeEntity.CreatedBy = "shivani’s UId";

                employeeEntity.UpdateOn = DateTime.Now;
                employeeEntity.UpdateByName = "shivani";
                employeeEntity.UpdateBy = "shivani’s UId";

                employeeEntity.Version = 1;
                employeeEntity.Active = true;
                employeeEntity.Archieved = false;

                Employee resposne = await container1.CreateItemAsync(employeeEntity);

                // Reverse MApping
```

```
            empDTO.TaskName = resposne.TaskName;
            empDTO.TaskDescription = resposne.TaskDescription;



            return Ok(empDTO);
        }
        catch (Exception ex)
        {

            return BadRequest("Data Adding Failed" + ex);
        }
    }

    [HttpPut]
    public async Task<IActionResult> UpdateItem(string uId, string name, string taskDesc)
    {

        Employee existingTask = container1.GetItemLinqQueryable<Employee>(true).Where(q => q.Docu
mentType == "Employee" && q.UId == uId).AsEnumerable().FirstOrDefault();
        if (existingTask != null)
        {
            existingTask.TaskName = name;
            existingTask.TaskDescription = taskDesc;
            existingTask.Version++;

            try
            {
                var response = await container1.UpsertItemAsync(existingTask, new PartitionKey(uId));
                if (response.StatusCode == System.Net.HttpStatusCode.OK || response.StatusCode == Syst
em.Net.HttpStatusCode.Created)
                {
                    return Ok("Task Updated Successfully");
                }
                else
                {
                    return BadRequest("Failed to Update Task");
                }

            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }

        }
        return BadRequest();
    }
    [HttpGet]
    public IActionResult GetemployeeByUId(string uId)
    {
        try
        {
            Employee tasks = container1.GetItemLinqQueryable<Employee>(true).Where(q => q.Documen
tType == "Employee" && q.UId == uId).AsEnumerable().FirstOrDefault();
```

```
            var taskModel = new empDTO();
            taskModel.TaskName = tasks.TaskName;
            taskModel.TaskDescription = tasks.TaskDescription;
            return Ok(taskModel);
        }
        catch (Exception ex)
        {
            return BadRequest("Data Get Failed");
        }
    }
    [HttpGet]
    public IActionResult GetAllEmployee()
    {
        try
        {

            var listresponse = container1.GetItemLinqQueryable<Employee>(true).AsEnumerable().ToList()
;
            return Ok(listresponse);

        }
        catch (Exception ex)
        {

            return BadRequest("Data Get Failed");
        }
    }

    [HttpDelete]
    public async Task DeleteTaskAsync(string uId)
    {
        await container1.DeleteItemAsync<Employee>(uId, new PartitionKey(uId));
    }
    private Container GetContainer()
    {
        CosmosClient cosmoscClient = new CosmosClient(URI, PrimaryKey);
        Database database = cosmoscClient.GetDatabase(DatabaseName);
        Container container = database.GetContainer(ContainerName);
        return container;
    }

  }


}
[01-01-2024 15:46] shivani: using Newtonsoft.Json;
using System.Text.Json.Serialization;


namespace TaskAPI.Entity
{
  public class Emp
  {
    [JsonProperty(PropertyName = "active", NullValueHandling = NullValueHandling.Ignore)]
```

```
        public bool Active { get; set; }

        [JsonProperty(PropertyName = "archieved", NullValueHandling = NullValueHandling.Ignore)]
        public bool Archieved { get; set; }

        [JsonProperty(PropertyName = "version", NullValueHandling = NullValueHandling.Ignore)]
        public int Version { get; set; }

        [JsonProperty(PropertyName = "id", NullValueHandling = NullValueHandling.Ignore)]
        public string Id { get; set; }

        [JsonProperty(PropertyName = "uId", NullValueHandling = NullValueHandling.Ignore)]
        public string UId { get; set; }

        [JsonProperty(PropertyName = "taskName", NullValueHandling = NullValueHandling.Ignore)]
        public string TaskName { get; set; }

        [JsonProperty(PropertyName = "taskDescription", NullValueHandling = NullValueHandling.Ignore)]
        public string TaskDescription { get; set; }

        [JsonProperty(PropertyName = "updateBy", NullValueHandling = NullValueHandling.Ignore)]
        public string UpdateBy { get; set; }


        [JsonProperty(PropertyName = "updateByName", NullValueHandling = NullValueHandling.Ignore)]
        public string UpdateByName { get; set; }

        [JsonProperty(PropertyName = "updateOn", NullValueHandling = NullValueHandling.Ignore)]
        public DateTime UpdateOn { get; set; }

        [JsonProperty(PropertyName = "createdBy", NullValueHandling = NullValueHandling.Ignore)]
        public string CreatedBy { get; set; }


        [JsonProperty(PropertyName = "documentType", NullValueHandling = NullValueHandling.Ignore)]
        public string DocumentType { get; set; }

        [JsonProperty(PropertyName = "createdByName", NullValueHandling = NullValueHandling.Ignore)]
        public string CreatedByName { get; set; }

        [JsonProperty(PropertyName = "createdOn", NullValueHandling = NullValueHandling.Ignore)]
        public DateTime CreatedOn { get; set; }



    }
}
```