***// each of these import declarations should be commented –***

***// - what are we importing from each module?***

***// what do each of the imported functions do, in one phrase?***

***//***

using CPH\_IVT.Models;

using CPH\_IVT.Services.MongoDB.Repository;

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

using System.Threading;

using System.Threading.Tasks;

***// in what follows, the getters in the repository access routines call await. Will this be an issue? What about timeouts?***

***//***

namespace CPH\_IVT.Services.MongoDB.Init

{

public class InitializerAsync

{

private static readonly string CurrentPath = Environment.CurrentDirectory;

private static readonly string DataDirectory = Path.GetFullPath(Path.Combine(CurrentPath, "..", "..", "data", "indicators"));

***..***

***// can this be recast in C# as iteration over a pair of (left hand side, string) arrays??***

***// the repetition here is ugly.***

***//***

private static List<string> CountiesCSVData = File.ReadAllLines(Path.GetFullPath(Path.Combine(CurrentPath, "Counties.csv"))).ToList();

private static List<string> StatesCSVData = File.ReadAllLines(Path.GetFullPath(Path.Combine(CurrentPath, "States.csv"))).ToList();

private static List<string> CensusDivisionsCSVData =

File.ReadAllLines(Path.GetFullPath(Path.Combine(CurrentPath, "CensusDivisions.csv"))).ToList();

private static List<string> CensusRegionsCSVData =

File.ReadAllLines(Path.GetFullPath(Path.Combine(CurrentPath, "CensusRegions.csv"))).ToList();

IHealthIndicatorRepository HealthIndicatorRepository;

ICountyRepository CountyRepository;

IStateRepository StateRepository;

ICensusDivisionRepository CensusDivisionRepository;

ICensusRegionRepository CensusRegionRepository;

***// Ick! Hard-coded 100000 element list. 100000 should be parameterized.***

***//***

List<HealthIndicator> buffer = new List<HealthIndicator>(100000);

ConcurrentQueue<HealthIndicator> IndicatorsQueue = new ConcurrentQueue<HealthIndicator>();

ConcurrentQueue<County> CountiesQueue = new ConcurrentQueue<County>();

ConcurrentQueue<State> StatesQueue = new ConcurrentQueue<State>();

***// I told the group to update the name FuckingSendIt. They didn’t. We need to.***

***// And what’s being sent, anyway? Aren’t we receiving from CHR at this juncture? Or are we sending to Mongo? This name is unclear.***

***//***

public async void FuckingSendIt(

IHealthIndicatorRepository healthIndicatorRepository,

ICountyRepository countyRepository, IStateRepository stateRepository,

ICensusDivisionRepository censusDivisionRepository,

ICensusRegionRepository censusRegionRepository)

{

HealthIndicatorRepository = healthIndicatorRepository;

CountyRepository = countyRepository;

StateRepository = stateRepository;

CensusDivisionRepository = censusDivisionRepository;

CensusRegionRepository = censusRegionRepository;

try

{

// Creating a thread that will gather the health indicators and a Task that will send those in bulk with a buffer

var healthIndicatorThreadGather = new Thread(GatherHealthIndicators);

healthIndicatorThreadGather.Start(); ***// Where is the corresponding wait on this thread?***

***// in what follows, do we really need to send the health indicators, counties, states, census divisions, and census regions in sequence?***

***// can any of this content be sent concurrently?***

var healthIndicatorTaskSender = Task.Run(() => HealthIndicatorSender(healthIndicatorThreadGather));

healthIndicatorTaskSender.Wait(); ***// I don’t like this at all – a Wait on a task without a timeout.***

// TODO: Probably implement some kind of buffer for this like with health indicators since this will be a large amount of data

var countiesThread = Task.Run(() => CreateCounties());

countiesThread.Wait(); ***// I don’t like this at all – a Wait on a task without a timeout.***

await CountyRepository.CreateBulkAsync(CountiesQueue.ToList()); ***// another await without a timeout.***

CountiesQueue.Clear();

// TODO: The writing should probably be taken care of within the thread since these are so few but pretty large

var statesTask = Task.Run(() => CreateStates());

statesTask.Wait(); ***// I don’t like this at all – a Wait on a task without a timeout.***

await StateRepository.CreateBulkAsync(StatesQueue.ToList());

CountiesQueue.Clear();

// The data writing is taken care of within the thread since these are so few but extremely large

var censusDivisionTask = Task.Run(() => CreateCensusDivision());

censusDivisionTask.Wait(); ***// I don’t like this at all – a Wait on a task without a timeout.***

// The data writing is taken care of within the thread since these are so few but extremely large

***// TODO: Seems to be issue where we are not writting the last region***

***// I think there could be an error in how the parallel tasks’ parallel for statements are coded.***

***// See below for comments.***

var censusRegionsTask = Task.Run(() => CreateCensusRegions());

censusRegionsTask.Wait(); ***// I don’t like this at all – a Wait on a task without a timeout.***

}

catch (Exception ex)

{

var e = ex.Message;  ***// so, why aren’t we logging errors, or are we??***

}

}

private async Task<Task> HealthIndicatorSender(Thread work)

{

while (work.IsAlive || IndicatorsQueue.Count != 0)  ***// is this just spinning while work.IsAlive?? if so, ugly code***

{

if (IndicatorsQueue.Count > 0)

{

HealthIndicator healthIndicator = null;

IndicatorsQueue.TryDequeue(out healthIndicator);

if (healthIndicator != null) buffer.Add(healthIndicator);

if (buffer.Count == buffer.Capacity)

{

await HealthIndicatorRepository.CreateBulkAsync(buffer); ***// another await without a timeout.***

buffer.Clear();

}

}

}

await HealthIndicatorRepository.CreateBulkAsync(buffer); ***// another await without a timeout.***

buffer.Clear();

return Task.CompletedTask;

}

***// need a comment about input format – what assumptions is this code making?***

***//***

private void GatherHealthIndicators(object parameters)

{

Parallel.ForEach(Directory.GetDirectories(DataDirectory), directory =>

{

string pattern = @"\d{4}";

string year = Regex.Match(directory, pattern).ToString();

Parallel.ForEach(Directory.GetFiles(directory), path =>

{

var reader = new StreamReader(path);

string indicatorName = reader.ReadLine().Split(',')[4];

while (!reader.EndOfStream)

{

var nextLine = reader.ReadLine().Split(',');

var healthIndicator = new HealthIndicator

{

Name = indicatorName,

Year = year,

Value = double.Parse(nextLine[4]),

CountyId = string.Concat(nextLine[0], nextLine[1])

};

IndicatorsQueue.Enqueue(healthIndicator);

}

});

});

}

***// need a comment about input format – what assumptions is this code making?***

***//***

private Task CreateCounties()

{

CountiesCSVData.RemoveAt(0);

Parallel.ForEach(CountiesCSVData, county =>

{

var countyData = county.Split(',');

string stateFips = countyData[0];

string countyFips = countyData[1];

string countyId = string.Concat(stateFips, countyFips);

var newCounty = new County(countyData[0], countyData[1])

{

***// so, why is this in its own scope? Is this erroneous? And why does this code differ from the code for the other Tasks***

Name = countyData[2]

};

newCounty.Indicators = HealthIndicatorRepository.GetAllByCountyIdAsync(newCounty.CountyId).Result;

CountiesQueue.Enqueue(newCounty);

});

***// is there a need to wait for the CreateAsync’s to complete? How could a parallel for detect the completions?***

return Task.CompletedTask;

}

***// need a comment about input format – what assumptions is this code making?***

***//***

private Task CreateStates()

{

StatesCSVData.RemoveAt(0);

Parallel.ForEach(StatesCSVData, state =>

{

var stateData = state.Split(',');

var newState = new State

{

FIPS = stateData[0],

Abbreviation = stateData[1],

Name = stateData[2],

CensusDivisionNumber = stateData[3]

};

newState.Counties = CountyRepository.GetAllCountiesByStateFIPSAsync(newState.FIPS).Result;

StatesQueue.Enqueue(newState);

});

***// is there a need to wait for the CreateAsync’s to complete? How could a parallel for detect the completions?***

return Task.CompletedTask;

}

***// need a comment about input format – what assumptions is this code making?***

***//***

private Task CreateCensusDivision()

{

CensusDivisionsCSVData.RemoveAt(0);

Parallel.ForEach(CensusDivisionsCSVData, censusDivision =>

{

var censusDivisionData = censusDivision.Split(',');

var newCensusDivision = new CensusDivision

{

Number = censusDivisionData[0],

Name = censusDivisionData[1],

CensusRegionNumber = censusDivisionData[2]

};

newCensusDivision.States = StateRepository.GetAllStatesByDivisionNumberAsync(newCensusDivision.Number).Result;

CensusDivisionRepository.CreateAsync(newCensusDivision);

});

***// is there a need to wait for the CreateAsync’s to complete? How could a parallel for detect the completions?***

return Task.CompletedTask;

}

***// need a comment about input format – what assumptions is this code making?***

***//***

private Task CreateCensusRegions()

{

CensusRegionsCSVData.RemoveAt(0);

Parallel.ForEach(CensusRegionsCSVData, censusRegion =>

{

var censusRegionData = censusRegion.Split(',');

var newCensusRegion = new CensusRegion

{

Number = censusRegionData[0],

Name = censusRegionData[1]

};

newCensusRegion.CensusDivisions = CensusDivisionRepository.GetAllDivisionsByRegionNumberAsync(newCensusRegion.Number).Result;

CensusRegionRepository.CreateAsync(newCensusRegion);

});

***// is there a need to wait for the CreateAsync’s to complete? How could a parallel for detect the completions?***

return Task.CompletedTask;

}

}

}