פרויקט חצי קורס פולסטאק ציונט

מערכת לקידום קמפיינים ברשת הטוויטר עבור חברות ללא מטרות רווח

הערות חיים (13/1) + תוספות שלי

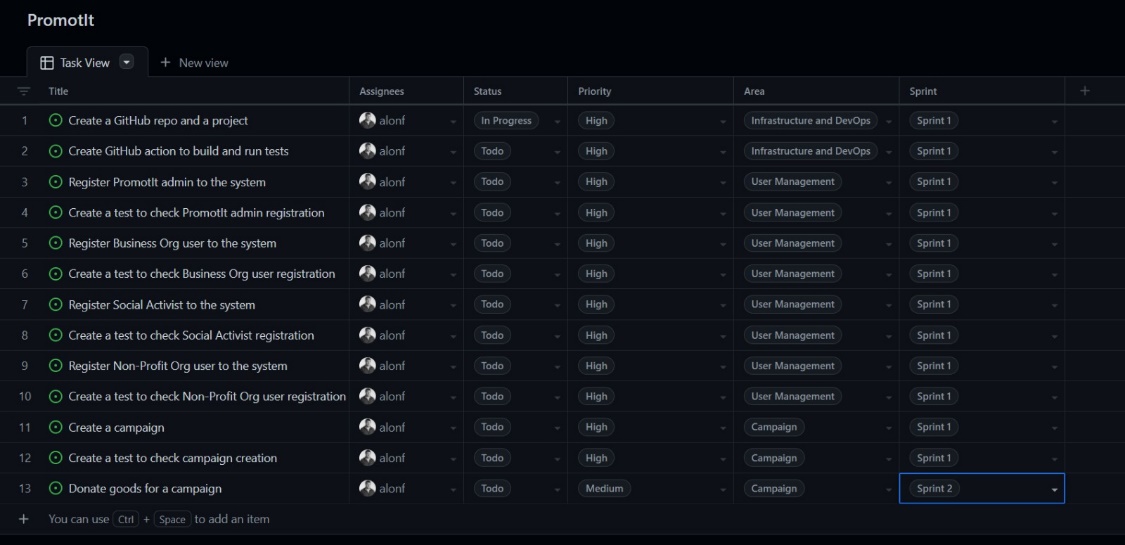
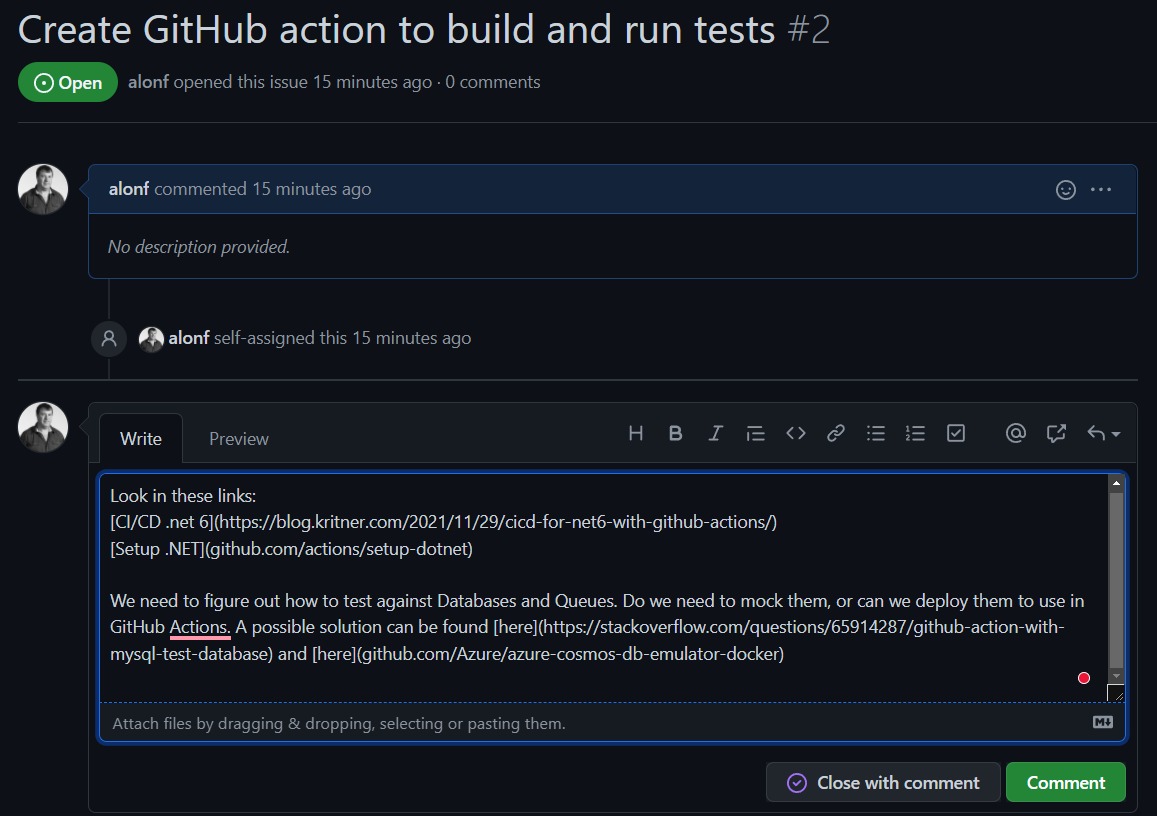
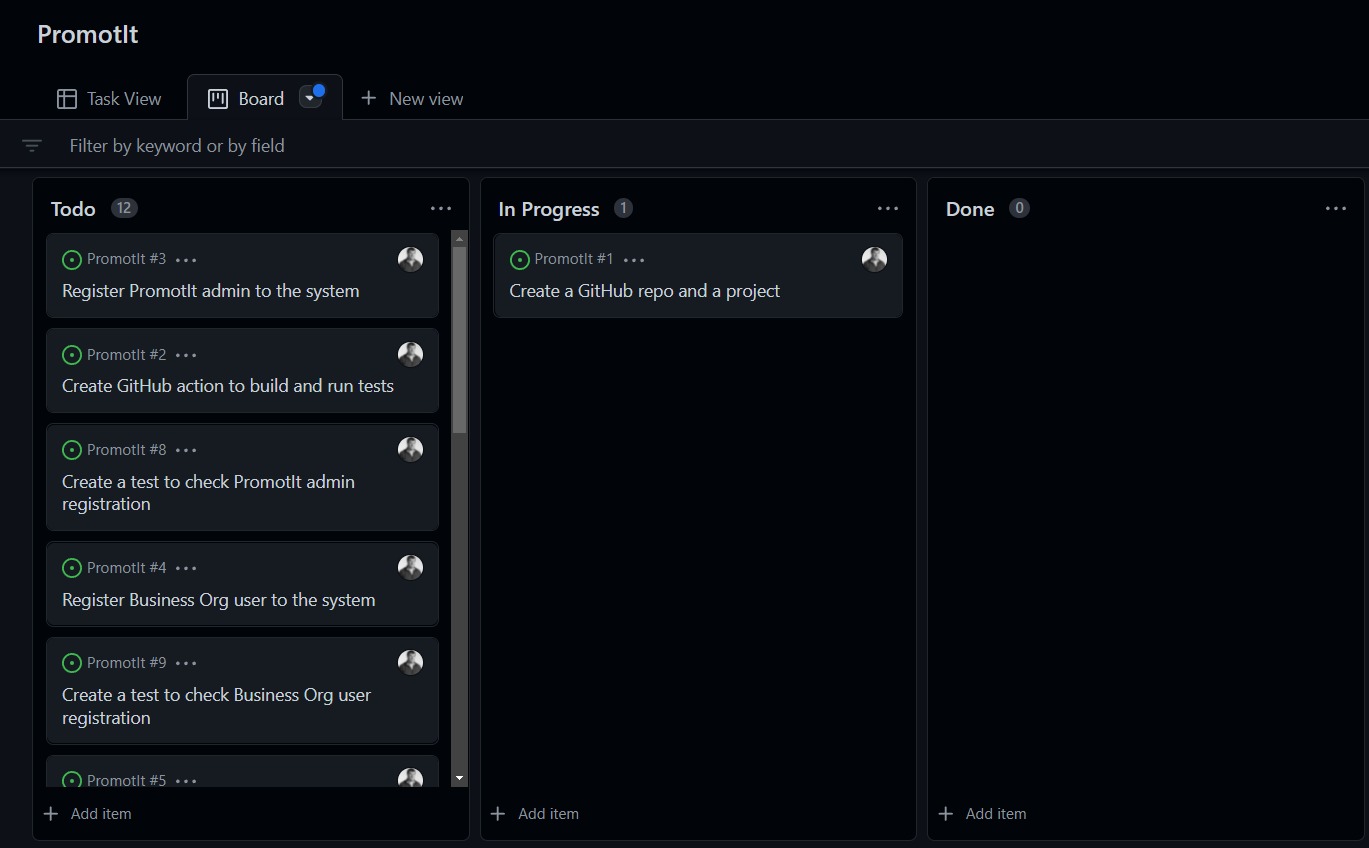
* לבנות דאטה בייס
* לפתח באזור פאנקשיין
* לבנות winform
* 4 סוגים של משתמשים
  + **ProLobby –** האדמין של המערכת
    - אדמין בעל שם משתמש וסיסמה
    - המערכת תשמור כל טוויט שביצע הפעיל חברתי עם הטקסט והאשטאג
    - המערכת שולחת לכל פעיל חברתי דולר על כל טוויט או הייחסות
    - המערכת שולחת טוויט על כל רכישה שבוצע על ידי פעיל חברתי עם שם הפעיל ושם החברה התורמת
    - המערכת תדווח על קמפיינים , משתמשים , טוויטים
  + **Non-Profit Organization Representative –** הנציגים ללא מטרות רווח
    - רוצים לקדם אגנדות חברתיות
    - האירגון ללא מטרות רווח נרשם למערכת עם שם אירגון, מייל וקישור לאתר (סיסמה)?
    - הנציגים יכולים לייצר קמפיין מסוגים שונים עם קישור ואשטאג (חד חד ערכי)

* + **Business Company Representative –** נציגים של חברות מסחריות
    - מוכנים לתרום כסף או פרסים לעודד פעילים לעשות קמפיין
    - חברה מסחריות נרשמת למערכת עם שם חברה וסיסמה
    - חברה מסחריות יכולה לבצע תרומה עם שם מוצר, כמות נתרמת או
    - סכום כסף (בדולרים) וקמפיין שאליו הם רוצים לתרום
    - מוצר מכיל את שם המוצר ואת שווי שלו ומספר מזהה
    - חברה מסחריות מבצעת שליחה של מוצרים עם שם המוצר ומספר קנייה ועדכון התרומה
  + **Social Activists – פעילים חברתיים**
    - מתייגים ב# אגנדות חברתיות בטוויטר
    - פעיל חברתי נרשם עם מייל שם מלא כתובת מספר פלאפון סיסמה ואת היוזר של הטוויור פעיל חברתי גם בעל חשבון כסף
    - פעיל חברתי מקבל כסף על כל טוויט או הייחסות דולר
    - כאשר פעיל חברתי צובר מספיק כסף הוא יכול לבצע קנייה עם שם מוצר
* להשתמש בAPI של טוויטר (לפתוח יוזר של מפתח)
* הדאטה בייס יהיה בענן
* הקוד יהיה באזור פאנקשיין

הערות אלון (13/1)

* azure functions appמייצר שיכבה של services ה ui קורא אליהם
  + דוגמא מייצרים קמפיין חדש אז יהיה service שיהיה אחראי על קמפיינים עם create getlist ועוד
  + Service שמאזין לטוויטר ובוקד מידי פעם את האשטאש של הקמפיינים
  + Service של באלנס שמתעסק בכסף
* ליצור accessors שידברו עם הדאטה בייס
* להחליט אם להשתמש בראפראנס בקוד או לקרוא לו בrest
* ב services להשתמש בopen api (swagger)
  + ברגע ששמים atribute של open api מקבל services ויוצר קליינט
* להשתמשApi לדברים שלא משנים דאטה
* להשתמש בQueues לדברים שמשנים דאטה
  + הודעות דרך Que לתת להודעה identity
    - דוגמא ההודעה תכיל id של הטוויטר כדי לראות שההודעה בוצעה
* יעלה יותר מאינסטנס אחד של הקוד שקורא טוויטים
* ללמוד CI/CD pipeline
  + CI היגט אחרי הפולריקוסט לוקח מכונה ריקה מתקין את כלים של הקומפילציה מקמפל את הקוד ומריץ בדיקות ובונה את התוצר הכל בענן
  + CD פורס את הקוד בענן
* ApprovalTests.Net ספרייה מאפשרת לכתוב טסטים בצורה קלה
  + למשל כותבים לapi ומקבלים תשובה Approval בודק אם התוצאה דומה לריצה הקודמת
* לבנות פיפליין בגיטהב לדוט נט 6
* לבחור אם אקטייביסט מקבל כסף כללי על כל טוויט או לכל קמפיין קופה שונה
* לAzure Logic Apps יש חיבור לטוויטר עושה חיים קלים אבל צריך חשבון ענן
* מחפשים טוויטים לפי האשטאג בודקים מי היוזר ואפשר גם את הURL ואם קיים במערכת מגיע דולר
* כל ריטוויט שעושים לטוויט של אקטיביסט מקבל האקטיביסט גם דולר
* צריך לכתוב טסט שעושה קרייט 3 קמפיים ואז עושה גטאול קמפיינס ובודק אם יש 3 קמפיינים
* swagger מאשפר לעשות בנייה אוטומית של הקלינט מאפשר לקבל ui בשביל להפעיל את הקריאות מהקומד ליין מכניס את ה Service שבונה את swagger
* Swagger זה open api
* לחפש איך עובדים סטורג קיו באזור
* ליצור קלאסים ולכתוב את הקוד שם הapi יקרא לקוד מהם
* עדיף שהקלאסים יזרקו ל ICalculationService
* להתחיל מליצור משימות
* ליצור flow תרשים זרימה של המערכת
* להתחיל מליצור serviceאחד פשוט יוזר יוצר קמפיין
  + ליצור יוזר
  + לצור דאטה בייס ששומר קמפיינים
  + צריך accessors שמדבר עם הדאטה בייס
  + צריך service שיש לו API שהוא עושה קרייט קאמפיין או פוסט של קמפיין
  + קמפיין מכיל שם האשטאג url
  + ליצור קמפיין פוסט שמקבל את האוביקט קורא או לקלאס או לסרביס אחר שמדבר עם הדאטה בייס של קרייאט קמפיין ויוצר אותו בדאטה בייס
  + ליצור עכשיו גטקמפיין גטקמפיין byID גט אול קמפיין
  + לכתוב טסטים
* עדיף ש Accessors ימשוך מתוך תור
* טוויטר יכול להיות Service אחד מסתכל אם יש טווטים ובאיזה נושא וגם טוויט בעת מכירה
* עדיף ליצור כמה api שונים לכל סוג משתמש
* ליצור Service שאחראי על הכסף
* ליצור Service של מכירת
* בגיט אקשיין ליצור מוק דאטה בייס

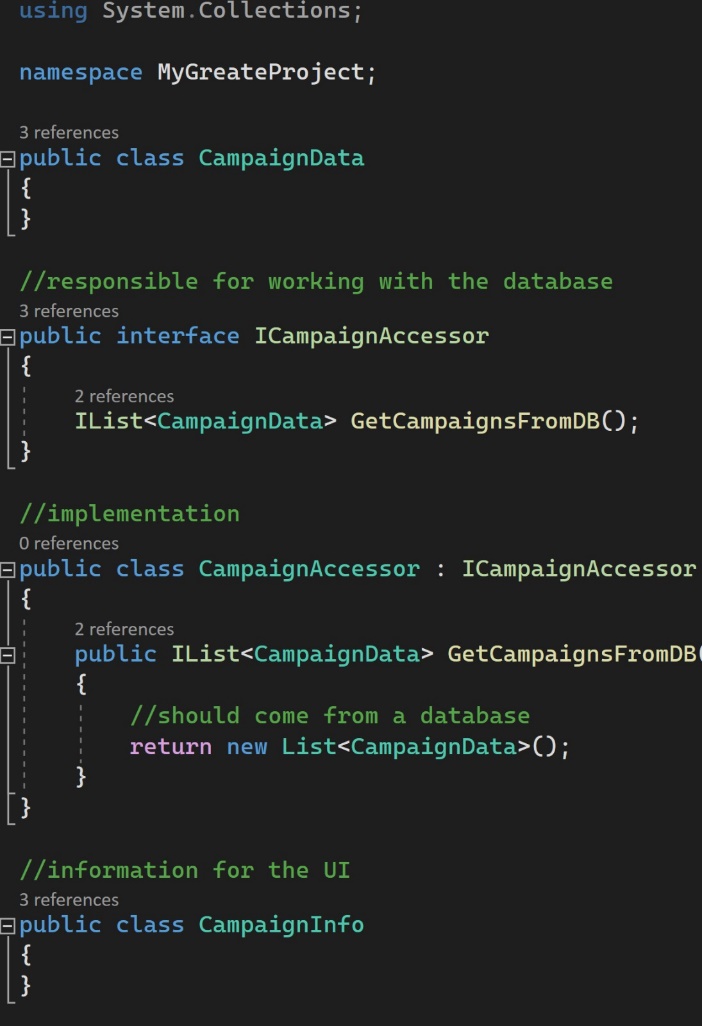
הערות אלון ווצאפ (14/1)

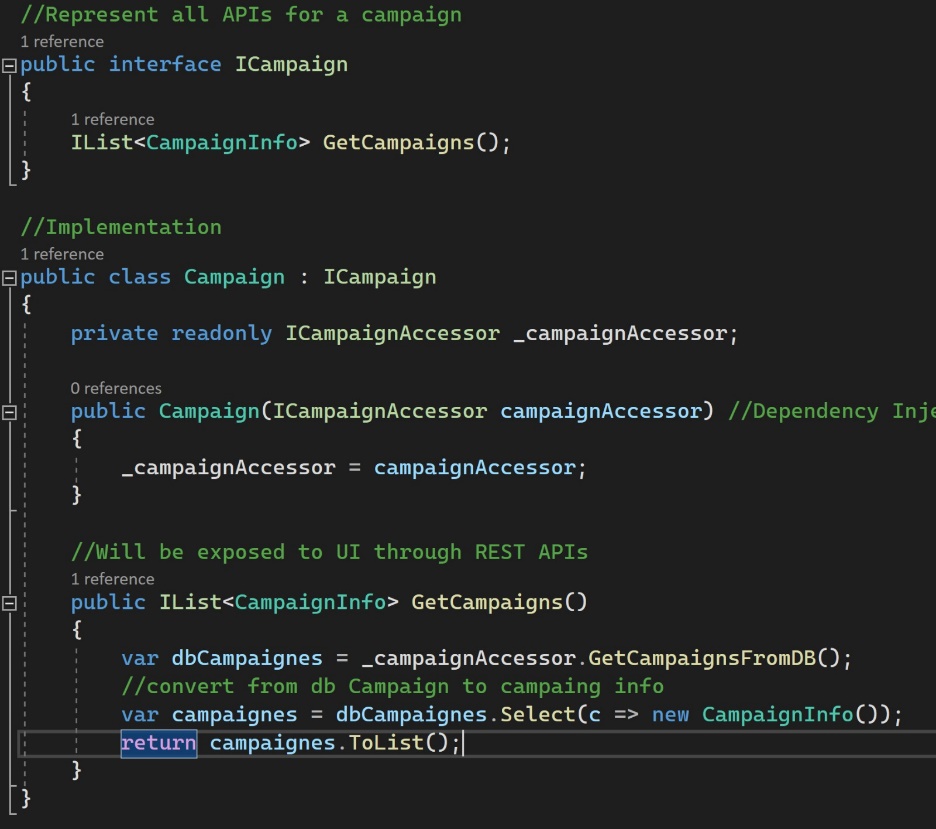
* כשאתם מתחילים לפתח את הפרויקט ומשתמשים ב GitHub, את קובץ הקונפיגורציה שמכיל את המפתחות של Twitter נא לא להעלות ל github. לשמור אותו על המחשב שלכם. תעבירו אותו בינכם. כמובן שאם היינו עובדים אל מול הענן, אז יש פתרון לשמירת נתונים סודיים: <https://azure.microsoft.com/en-us/services/key-vault/>
* לטובת ריצה של טסטים בגיטהאב, ניתן להשתמש במשתני סביבה בהתאם למה שכתוב פה: docs.github.com/en/actions/security-guides/encrypted-secrets
* ולכן אולי יהיה לכם נוח להחזיק את המפתחות כמשתני סביבה גם במחשב הפרטי שלכם.
* מי שרוצה (וכדאי) לנהל את משימות הפיתוח בצורה מסודרת בגיטהאב יכול לעשות את מה שמופיע פה: <https://github.com/features/issues>
* docs.github.com/en/issues/trying-out-the-new-projects-experience<blob:https://web.whatsapp.com/cbeec511-f76a-4737-b7bf-c682eb1b7f61>
* 
* כאשר לוחצים על כל משימה, נפתח מסך שבו ניתן להרחיב את התיאור של המשימה.
* 
* תייצרו גם Board View וכך תוכלו להעביר משימות מעמודה לעמודה
* 
* שימו לב לפרט חשוב. כאשר יוצרים Pull Request, צריך להגיד איזה Issues הוא מממש. זה מאפשר מעקב אחר הקוד שמימש את הדרישות, או קוד שפתר את הבאג. כמו כן זה מאפשר לדעת מי כתב את הקוד שמימש את הפיצ'ר או פתר את הבאג.
* ביום שני בשעה שש בערב נקיים פגישת הנחייה. תשרינו לפחות שלוש שעות. מומלץ מאוד עד הפגישה לבנות תוכנית משימות מסודרת. אם אתם לא מסתדרים עם יצירת פרוייקט ב GitHub אז ניתן בנתיים להשתמש באקסל ואני אראה לכם איך לעבוד עם GitHub Project. תכינו שאלות, כמובן שניתן גם לשאול שאלות פה, או בפרטי.

הערות אלון ווצאפ (18/1)

חברה, אני מבין שהפרויקט הוא רף גבוה, ואחרי אתמול הוא נראה עוד יותר מסובך. אני רוצה להדגיש שדרישות המינימום אינן מחייבות לרדת לכל הפרטים שהצגתי. ניתן לייצר פרויקט אחד שמחצין APIs לכל הפעולות ומשתמש ב Database יחיד. וכל המימוש של הדרישות נמצא במחלקות שונות בתוך הפרויקט הזה. הגישה הזו פשוטה משמעותית מהגישה שהראיתי אתמול. אפשר להתחיל ככה ואז לעבור לגישה המורכבת יותר, או להגיש את הפרויקט בצורה הזו. אז כדי לחדד, הנה הגדרה לפרויקט מינימאלי:

1. יצירת GitHub Repository על מנת שכל הקוד יהיה במקום אחד
2. יצרת פרויקט עם רשימת משימות – או ב GitHub Project או באקסל או בכל כלי נוח לניהול המשימות
3. חלוקה למחלקות (classes) לפי אחריות כגון:
   1. Campaign, Users, SocialNetwork, וכו'
   2. לכל כזו מחלקה צריך לייצר אינטרפייס כגון ICampaign שיכיל את הפונקציות שהמחלקה מחצינה, כגון GetCampainLists וכו'
4. כל קוד שצריך לקרוא לקוד במחלקה אחרת עושה זאת דרך האינטרפייס שהוגדר.
5. כדאי להשתמש ב IoC Container על מנת להעביר את האינטרפייסים לקוד שצריך אותם.
6. בצורה הזאת יבנה לכם פרויקט שהוא exe יחיד שמכיל את כל הקוד. בניגוד למה שהראיתי אתמול שכל שירות הוא Azure Functions או ASP.NET Web API.
7. הקוד שלכם צריך מצד אחד להשתמש ב Database, ומצד שני להחצין APIs לצד המשתמש.
8. ניתן לצורך הפשטות להשתמש ב Database יחיד בניגוד לגישת ה Microservices שבא לכל שירות יש Database בפני עצמו.
9. כדי להחצין את הפונקציות לצד המשתמש יש לייצר REST API, הדרך הפשוטה לעשות זאת היא להשתמש בפרויקט ASP.NET MVC API
   1. ז"א כלל המחלקות שיצרתם יתארחו בפרויקט הזה. ניתן לנצל את ה IoC Container המובנה של ASP.NET לטובת השימוש במחלקות השונות שיצרתם.
10. את הטסטים ניתן לכתוב אל מול המחלקות העיקריות, על ידי הוספת reference בטסט לפרויקט הראשי שלכם וקריאה לקוד של המחלקות, או קריאה דרך ה REST API





using System.Collections;

namespace MyGreateProject;

public class CampaignData

{

}

//responsible for working with the database

public interface ICampaignAccessor

{

IList<CampaignData> GetCampaignsFromDB();

}

//implementation

public class CampaignAccessor : ICampaignAccessor

{

public IList<CampaignData> GetCampaignsFromDB()

{

//should come from a database

return new List<CampaignData>();

}

}

//information for the UI

public class CampaignInfo

{

}

//Represent all APIs for a campaign

public interface ICampaign

{

IList<CampaignInfo> GetCampaigns();

}

//Implementation

public class Campaign : ICampaign

{

private readonly ICampaignAccessor \_campaignAccessor;

public Campaign(ICampaignAccessor campaignAccessor) //Dependency Injection of ICampaignAccessor

{

\_campaignAccessor = campaignAccessor;

}

//Will be exposed to UI through REST APIs

public IList<CampaignInfo> GetCampaigns()

{

var dbCampaignes = \_campaignAccessor.GetCampaignsFromDB();

//convert from db Campaign to campaing info

var campaignes = dbCampaignes.Select(c => new CampaignInfo());

return campaignes.ToList();

}

}

דרך אגב דוגמאת הקוד מתאימה גם למי שכן רוצה לפרק את המערכת למספר רב של שירותים, כי ה Manager צריך לדבר עם ה Engine ועם ה Accessor ובתוך הקוד שלו יהיו מחלקות שיבצעו את התקשורת והמחלקות האלו אמורות להיות מוזרקות ב Dependency Injection

גם איליה וגם אני לימדנו את הנושאים. אני אזכיר לך שדיברנו על ASP.NET Minimal API שהראתי ואפילו ביקשתי שתעשו שיעורי בית בנושא. בנוסף אני יודע מאיליה שהוא דיבר אתכם על Dependency Injection, אז כנראה אתה לא מקשר את זה עם IoC Containers.

בכל מקרה למי שצריך רענון:

https://www.c-sharpcorner.com/UploadFile/cda5ba/dependency-injection-di-and-inversion-of-control-ioc/

מי שרוצה בעברית:

<https://www.geektime.co.il/dependency-injection-101/>

וההסבר המתאים ביותר ל C#:

<https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio>

מכיוון שבדיקת יחידה - Unit Test - דורש שנוכל להפריד כל מחלקה מכל התלויות שלה, הנושא הזה תמיד עולה בהקשר הזה. אבל המהות של DI חשובה יותר מהצורך בבדיקות. למעשה זה מאפשר לבנות חלקי לגו ולאחר מכן לבנות קוד מחלקי הלגו.

למעשה שעוברים ל Micro Services, אז יש לנו הפרדה מכיוון שכל שירות מחצין גישה דרך ה REST API שלו, וכל שירות משתמש ב Database שלו או שקורא ל API של שירותים אחרים ולכן ברמת ה Integration Test זה אמור להספיק. אבל שנכנסים למימוש של כל שירות, אנחנו צריכים להשתמש ב Logger ובגישה למאבי ענן ועוד מנגנוניים פנימיים ולכן גם בקוד המימוש של כל שירות אנחנו משתמשים ב DI. זה מאפשר לנו בעתיד לשדרג את הקוד, להחליף חלקים מבלי לשנות את כל הקוד.

למי שרוצה להריץ Github Action עם הרצת טסטים אל מול האמולטור של Azure Storage ועם האמולטור של CosmosDB, אז הנה ה flow שעובד. הוא רץ על Windows Runner, בלינוקס עדיין יש לי בעיה. אבל זה מספיק בשביל בנייה וריצה אוטומטית של הטסטים בגיטהאב:

name: .NET Windows

on:

push:

branches: [ master ]

pull\_request:

branches: [ master ]

jobs:

build\_and\_test:

name: Build and test

runs-on: windows-latest

steps:

- uses: actions/checkout@v2

- name: Setup .NET

uses: actions/setup-dotnet@v1

with:

dotnet-version: 6.0.x

- name: Setup .NET SDK 6.0 LTS

uses: actions/setup-dotnet@v1

with:

dotnet-version: 6.0.x

- name: Restore dependencies

run: dotnet restore

- name: func 4.x

run: npm i azure-functions-core-tools@4 --unsafe-perm true

- name: Build

run: dotnet build --no-restore

- name: 'Install Cosmos Emulator'

uses: southpolesteve/cosmos-emulator-github-action@v1

- name: start Azurite

shell: bash

run: |

cd DevOps

npm install -g azurite

mkdir azurite

azurite --silent --location azurite &

- name: Start function apps

shell: pwsh

run: |

$rootDir = $pwd.Path

cd ./Services/HttpTriggerQueueOutput

Start-Process -NoNewWindow .\node\_modules\azure-functions-core-tools\bin\func.exe -ArgumentList "host","start","--port","7071"

Start-Sleep -s 40

cd $rootDir/Services/QueueTriggerCosmosOutput

Start-Process -NoNewWindow .\node\_modules\azure-functions-core-tools\bin\func.exe -ArgumentList "host","start","--port","7072"

Start-Sleep -s 40

cd $rootDir/Tests/HttpTriggerQueueOutputTest

dotnet test . -c Debug

cd $rootDir

בסקריפט האחרון תשנו את השמות של ה Function Apps ואת הנתיבים וכן את הנתיבים והשמות של הטסטים וגם את הפורטים של כל פונקציה.

אני שוב שם פה את הלינק לפרויקט הדוגמא בגיטהאב שכולל שתי פונקציות, עבודה מול Azure Storage Queue ועבודה מול CosmosDB כולל הבניה האוטומטית והרצת הבדיקות:

<https://github.com/alonf/AzureFunctionsMSADemo>

למי שלא מבין איך הקוד שהעלתי עובד, תעשו Fork בגיטהאב ואז תוכלו לשחק עם הקוד אצלכם. אפשר ללכת ל Actions ולהריץ את הבניה בצורה ידנית, זאת אומרת לבקש מגיטהאב להריץ את כל התהליך (Pipeline) או פשוט לשנות את הקוד וזה יקרה מעצמו.

https://github.com/features/actions

הערות אלון (17/1) זום

* מטודה idesign
  + שכבת המנהלים (manager) **מה עושים?**
    - סוג של מנהל לא כותבים בו קוד חדש אלא קוראים לקלאסים ופונקציות חדשות שיעשו את העבודה
    - המנהל מציג פלאו עסקי
    - המנהל קורא למנועים (engines) מבצעים את הקוד
  + שכבת המנועים (engine) **איך עושים?**
    - כל קוד שמבצע עבודה כלשהיא נמצא במנוע
  + שכבת המגשרים (accessors) **מי עושה?**
    - נותן גישה למנוע לדאטה
* מנהלים מדברים עם אנגנים ואם אין עבודה יכולים לדבר אקססורים
* אנגנים יכולים לדבר רק עם רק עם אקססורים
* אקסורים מדברים רק עם העולם מחיצוני (דאטה בייס)