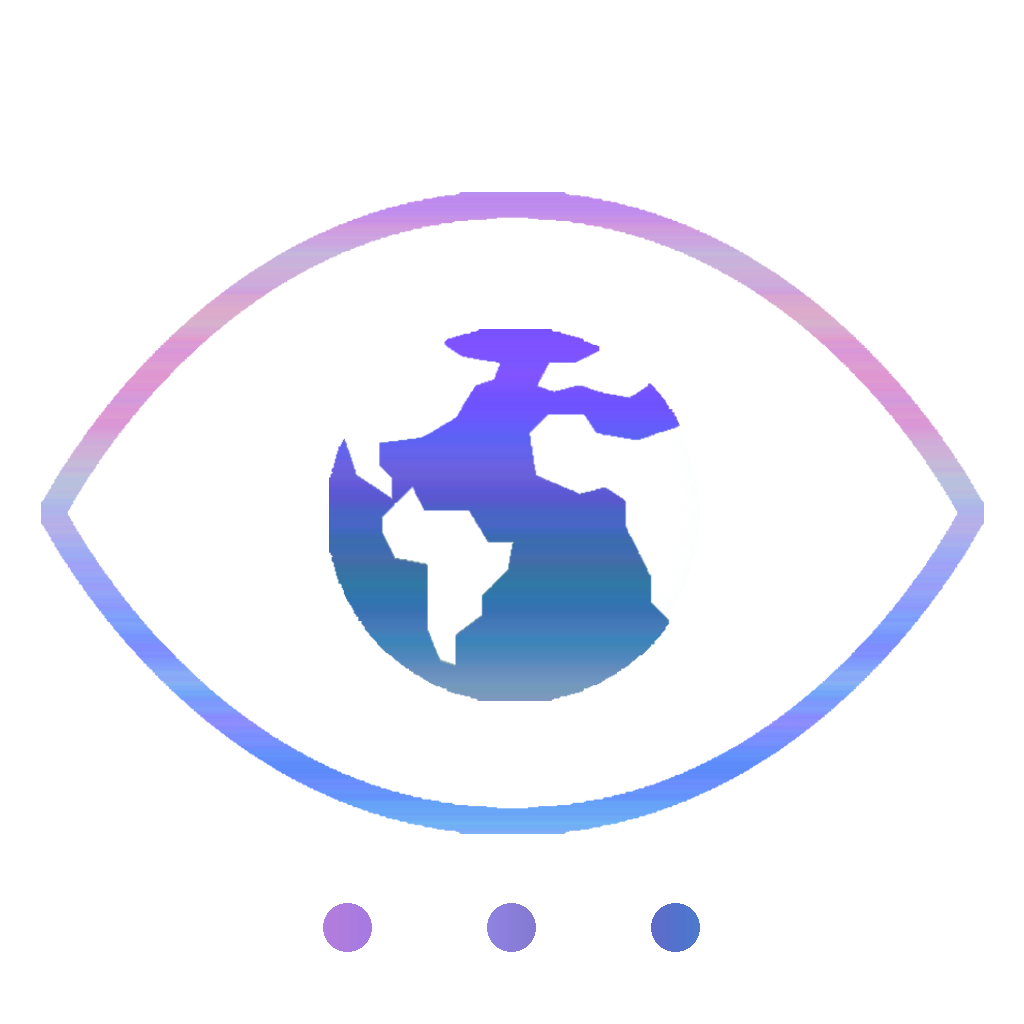
**mARk It - יומן מסע**

****

mARk It היא אפליקציה המאפשרת למשתמש לשתול חותם וירטואלי בצורת שלט עם טקסט במיקומו הגיאוגרפי באמצעות טכנולוגיית Augmented Reality.

המשתמש יכול לצפות ולדרג חותמים של משתמשים אחרים המתאימים לתחומי העניין שלו הנמצאים במיקומו הגיאוגרפי, על גבי מסך המצלמה שבמכשירו.

**mARk It - Video**

<https://drive.google.com/open?id=1Ixu1ZtF4MtXZalcvh6I_VIIzXBwURiFe>

**תוכן עניינים:**

* תחילת העבודה 3
* חיפוש ספריית שירותי AR 4
* האפליקציה 5
* Xamarin.Native 6
* Augmented Reality 7
* Authentication 8
* פיתוח צד-שרת 10
* נוטיפיקציות 13
* מסכי האפליקציה 15

**תחילת עבודה:**

* ישבנו לראשונה באופן מסודר כדי לדון ברעיונות. לאחר מס' שעות של סיעור מוחין העלינו בידינו 4 רעיונות לסדנה:

1. אפליקציית AR שבה המשתמשים יוכלו לשתול בעולם תוכן שמשתמשים אחרים יאתרו באמצעות GPS.
2. תוכנת מחשב התסיע לגיימרים למצוא חברים לשחק איתם לפי קריטריונים כמו יכולת במשחק, קרבה, שפה וכו'. בנוסף התוכנה תאפשר לתקשר בעת משחק ע"י voice chat.
3. אפליקציה לתיעוד ושיתוף מסלולי טיולים המבוססת על GPS.
4. אפליקציה שמזהה הבדלים בין מצבו החיצוני של רכב מסויים בזמן A למצבו של אותו רכב בזמן B. ה-UseCase העיקרי הוא לסייע לסוכנויות השכרת רכב ולקוחותיהן לאמת האם לקוח ביצע נזק כלשהו ברכב או שמא הנזק כבר היה קיים לפני שהלקוח שכר את הרכב.

* לאחר זמן מה הצגנו את הרעיונות לגיא. עקב המפגש פסלנו את רעיונות 2 ו-4 הנ"ל, וקיבלנו הנחיות וחידודים לגבי בדיקת ישימות של אפליקציית ה-AR.

בנוסף, גיא נתן לנו רעיון נוסף - לפתח אפליקציית פרומפטר/קריוקי אשר תצלם את המשתמש תוך כדי שהוא קורא/שר לפי הטקסט (והאקורדים) שמופיע על המסך. החידוש באפליקציה יהיה שהטקסט יתקדם לפי האזנה למשתמש.

* נפגשנו שוב על מנת לבחור באופן סופי את הרעיון לסדנה - לאור מחקר שביצענו אודות ספריית AR העובדת עם XAMARIN בחרנו באפליקציית ה-AR אשר כולנו התלהבנו מהרעיון שלה.
* לאחר כמה ימים נפגשנו כדי ליצור את מסמך האפיון של האפליקציה. תוך כדי המפגש יישרנו קו וקיבלנו החלטות לגבי אופן השימוש בה. בין היתר החלטנו שהאפליקציה תיקרא " Leave a m-AR-k".

הוספנו למסמך האפיון את מסכי האפליקציה כפי שאנו מניחים שיראו מבחינת UI. נעזרנו באתר MockFlow לשם כך.

* לבסוף הצגנו את מסמך האפיון הראשוני בפני גיא, אשר נתן לנו אור ירוק והצעות לשיפור. בנוסף החלטנו במפגש זה לקצר את השם של האפליקציה ל " mARk It".

**חיפוש ספריית שירותי AR:**

* מצאנו ספרייה נרחבת המספקת שירותי AR הנקראת Wikitude.

https://www.wikitude.com/

חיפשנו באינטרנט SDK לשימוש ב AR והגענו לבלוג הבא:

<https://thinkmobiles.com/blog/best-ar-sdk-review/>

לאחר בדיקה מעמיקה, ראינו כי Wikitude הוא אכן SDK שהכי מתאים לצרכינו שכן הוא מתאים לאפליקציות AR אשר משתמשות ב-GPS. יש בו אפשרות לשימוש חינמי וכן ישנה אפשרות לשלב אותו עם Xamarin וכך למעשה נוכל לממש את האפליקציה כ Cross Platform כפי שאנו רוצים.

* הורדנו מהאתר של הספרייה, סולושיין של VS ב- Xamarin המציג יכולות של הספרייה. לאחר מכן הרצנו את אותה אפליקציה על מכשיר אייפון וראינו כי אכן הספרייה הזאת מספקת לנו את השירותים הנדרשים של AR, GPS כפי שאנו צריכים.

https://www.wikitude.com/download-wikitude-sdk-for-xamarin/

**האפליקציה**

**Xamarin Forms**

תחילה פיתחנו את האפליקציה כאפליקציית Xamarin forms .

נעזרנו בקורס הבא על מנת ללמוד את הבסיס של ארכיטקטורת אפליקציית forms, יצירת מסכים בסיסית, הוספת מפה, מעבר בין מסכים.

<https://www.udemy.com/complete-xamarin-developer-course-ios-and-android/>

**החלטה על מעבר מ-Xamarin Forms ל-Xamarin Native:**

ההחלטה נבעה בעיקר מתאימות לא מלאה בין Wikitude ל Xamarin forms-

כבר בתחילת העבודה על הפרויקט ובדיקת הישימות, גילינו ש-wikitude לא מספקת תמיכה מלאה בforms אלא מתמקדים ב xamarin native .

אך בעזרת שאלת הפורום:

<https://support.wikitude.com/support/discussions/topics/5000078635>

והפתרון המוצע שם ע"י אחד מהלקוחות:

<https://github.com/mkoehne/WikitudeExample>

הצלחנו לשלב את מסך ה- AR ב Xamarin forms אך זה הצריך לשנמך את גרסת ה-wikitude בפרויקט אך הפונקציונליות הדרושה לנו עדיין עבדה.

נוצרו מכך 2 בעיות שהתגלו רק לאחר מספר חודשים:

א. Wikitude שדרגו גרסה , דבר הגרם לחלק מהפונקציונליות שעבדה בבדיקתנו הקודמת לא לעבוד יותר. זה טרם גרם לנו לעבור ל-native מהסיבה שהפונקציונאליות שהיינו צריכים עדיין עבדה לאחר שביצענו שינויים בקוד. אבל זה העלה חשש שבעתיד הקרוב יתכנו עוד בעיות כאלו, מכך שאין תמיכה לforms.

ב. לאחר שלב בדיקת הישימות כאשר רצינו להתחיל לממש את האפליקציה בצורתה המוגמרת, אחד הדברים שהיו חשובים לנו מבחינת UI היה שעמוד ה שAR יהיה העמוד הראשי באפליקציה שיהיה מוצג כטאב. נתגלתה בעיה להציג את העמוד כך באנדרואיד, לאחר ניסיונות וחיפוש הבעיה באינטרנט נתקלנו בפוסט הבא: : <https://support.wikitude.com/support/discussions/topics/5000077568>

וגילינו שבגרסת ה-wikitude שאנחנו משתמשים בה (ישנה בשל תאימות עם forms ) אין

אפשרות להציג את מסך ה AR כ-Fragment (נדרש על מנת להציג את המסך בטאב

באנדרואיד), והדרך היחידה לממש זאת היא ע"י שדרוג גרסת ה wikitude. (דבר שלא היה

אפשרי ב forms ).

לסיכום, המעבר ל-Xamarin Native נבע בעיקר משיקולי UI ומניעת תקלות אפשריות בעתיד.

**Xamarin Native**

כאשר בוחרים לבנות את האפליקציה ב-xamarin native צריכים לכתוב את רוב ה-UI בנפרד עבור android ו-iOS.

יתרון בפיתוח ב xamarin native לעומת xamarin.forms הוא הדמיון בפיתוח מול יצירת אפליקציות native: בiOS ( שפות swift או objective-c שהן השפות הקלאסיות ליצירת אפליקציות iOS ) וב-android ( שפת java שהיא השפה הקלאסית ליצירת אפליקציות android ) , מהדמיון הרב, ניתן להיעזר בפתרונות לבעיות רבות בעזרת פתרונות בשפות אלו (ורק לבצע שינויים קטנים כך שהקוד יהיה c#) כך כמות המידע לפתרון בעיות גודל באופן משמעותי.

על יצירת אפליקציות Xamarin.iOS ו-Xamarin.android, הוספת מסכים בסיסיים, הוספת רכיבים, טאבים, מפה למדנו בעיקר מהקורס:

[https://www.udemy.com/complete-xamarin-developer-course-ios-and-android/](https://www.udemy.com/complete-xamarin-developer-course-ios-and-android/learn/lecture/8437976?start=0#overview)

Xamarin.iOS

כאשר מפתחים ב Xamarin.iOS ניתן לעצב את כל המסכים בעזרת designer נוח הקיים ב visual studio for mac. את כל המסכים מעצבים במסך אחד הנקרא Main.storyboard בו מגדירים את כל המסכים (רכיבים, מעברים בין מסכים ועוד).

ביצירת מסך חדש ב Main.storyboard, כאשר מעניקים למסך שם של מחלקה, אוטומטית נוצרת מחלקת View Controller עבור המסך המקושרת כבר לאלמנטי ה-UI, שם אפשר לממש את לוגיקת המסך.

אתגרים:

1. עיצוב המסכים כך שיתאימו לכל גדלי המסכים- בדיזיינר ניתן למקם את הרכיבים בצורה חופשית על גבי מסך אייפון בגודל ספציפי, אך כאשר מריצים את האפליקציה על מכשיר מסוג אחר, מיקום הרכיבים משתנה, את הבעיה הנ"ל אפשר לפתור בעזרת הוספת constraints על הרכיבים (מרחק בין רכיבים, מרחק מגבולות המסך וכו).

2. הוספת רכיבים שלא קיימים בדיזיינר- ניתן להוסיף רכיבים נוספים ע"י יצירת מחלקות ב-c# כך:

<https://docs.microsoft.com/en-us/xamarin/ios/user-interface/designer/ios-designable-controls-walkthrough>

באפליקציה שלנו היינו זקוקים ל radio buttons וכוכבי דירוג, שניהם לא קיימים בדיזיינר.

מצאנו אתר המספק ספריות המספקות את הרכיבים האלו (מספק גם רכיבים ל Xamarin.Android, Xamarin.Forms ) :

<https://www.syncfusion.com/>

השירות בתשלום, אך קיימת אפשרות לחשבון סטודנט חינמי.

Xamarin.Android

כאשר מפתחים ב Xamarin.android כותבים את עיצוב המסכים בשפת התגיות axml, ניתן להשתמש ב-designer הקיים בvisual studio אבל הוא מוגבל מאוד, לכן נעזרנו בתוכנה **Android Studio** על מנת לעצב את דפי האנדרואיד בצורה נוחה יותר שכן הכתיבה של ה-UI ב-Xamarin.android זהה לכתיבה UI עבור אפליקציות Android רגילות. קבצים אלו ממוקמים בתיקייה Resources/layout.

על מנת לממש את לוגיקת המסכים יש ליצור מחלקות c# מתאימות ולקשרן לעמוד הaxml ולרכיבים בו בצורה ידנית.

לדוגמה עבור מסך רגיל יש ליצור מחלקת Activity ועבור מסך טאב יש ליצור מחלקת Fragment.

**Augmented Reality**

על מנת לממש את מסך ה AR השתמשנו בשירות Wikitude .

השימוש ב-wikitude sdk דורש רישיון. ניתן לבקש רישיון עבור סטודנטים:

<https://www.wikitude.com/wikitude-academy/>

על מנת ללמוד איך להשתמש ב sdk השתמשנו באפליקציית הדוגמה שהם מספקים:

<https://github.com/Wikitude/wikitude-xamarin-examples>

וכן בדוקומנטציה:

<https://www.wikitude.com/external/doc/documentation/6.1/Reference/JavaScript%20API/>

[https://www.wikitude.com/external/doc/documentation/latest/xamarin/gettingstartedxamarin.htm](https://www.wikitude.com/external/doc/documentation/latest/xamarin/gettingstartedxamarin.html)l

את לוגיקת מסך ה AR מימשנו ב javaScript אבל הפעלת המסך מתבצעת בקוד c# בצורה שונה ב-android וב-iOS.

אתגרים:

1. תקשורת בין קוד c# לקוד javaScript בשני הכיוונים – נדרשנו להפעיל מתודות c# מתוך קוד ה javaScript (תקשורת מול השרת, מעבר למסך הדירוג עקב לחיצה על mARk...), ולהפעיל פונקציות javascript מקוד ה-c# (החזרת נתונים מהשרת).

גילינו ש wikitude מספקת פתרונות לכך:

א. בקוד ה-javaScript ניתן לבצע קריאה לפונקציה AR.platform.sendJSONObject

המעבירה לקוד ה- c# אובייקט json (בקוד ה c# מחליטים לאיזו מתודה האובייקט יגיע).

ב. על מנת להפעיל פונקציות javascript מתוך קוד ה-c# יש להשתמש ב architectView.CallJavaScript(“function call”).

2. הצגת טקסט על אובייקט במיקום גיאוגרפי מסוים - (על מנת לממש שלט ועליו טקסט) תחילה ניסינו להציג טקסט על אובייקט תלת מימדי, לאחר שלא הצלחנו (הצלחנו למקם אובייקט תלת מימדי על משטח אך לא להצמיד לו טקסט), החלטנו על מעבר לאובייקט דו מימדי של שלט על מנת להציג עליו טקסט. את האובייקט הדו מימדי יצרנו בעזרת המחלקה AR.ImageDrawable, את הטקסט בעזרת המחלקה AR.Label והצגנו אותם במיקום גיאוגרפי בעזרת המחלקה AR.GeoObject המקבלת מיקום גיאוגרפי ואובייקטים לצייר במיקום זה.

**Authentication**

באפליקציה שלנו יש צורך ביצירת משתמשים שכן יש צורך בשמירת מידע כמו העדפות משתמש וכן מידע על הדירוגים אשר כל משתמש ביצע ל-mARks של משתמשים אחרים.

רצינו לתת למשתמשים יכולת התחברות ישירה עם חשבונות Google ו-Facebook מבלי לדרוש רישום מול השרת שלנו. לצורך כך נעזרנו בספרייה Xamarin.Auth שלמעשה מממשת את הפרוטוקול OAuth באופן Cross Platform עבור Android ו-iOS.

ספרייה זו מפשטת את ההתחברות לחשבונות אלו בכך שהיא עוטפת את רוב העבודה הנדרשת. כמו כן השתמשנו בספרייה הזו בלוגיקה המשותפת ככה שה-UI יכול להיות הן של Android והן של iOS ולהשתמש באותו קוד על מנת להתחבר.

מדריך מפורט לשימוש בספרייה נמצא ב-GitHub של הספרייה:

<https://github.com/xamarin/Xamarin.Auth>

אתגרים:

1. העבודה עם התחברות ב-Google היוותה בעיות כיוון אותנטיקציה עם Google דורשת NativeUI. לצורך כך נעזרנו במדריך המפורט הבא:

<http://timothelariviere.com/2017/09/01/authenticate-users-through-google-with-xamarin-auth/>

אשר מסביר בפירוט כיצד ניתן להתגבר על המכשולים הן עבור Android והן עבור iOS.

מימוש יכולת AutoConnect ו-RefreshTokens:

לאחר שהצלחנו לבצע התחברות עם Xamarin.Auth רצינו לתת להוסיף לאפליקציה יכולת של התחברות אוטומטית מהפעמים הבאות שהוא מפעיל את האפליקציה.

לשם כך, נעזרנו בספרייה שנקראת [Xamarin.Essentials.SecureStorage](https://docs.microsoft.com/en-us/xamarin/essentials/secure-storage) המאפשרת לשמור לשמור את ה-Access Token במכשיר באופן מאובטח.

נעזרנו במדריך הבא:

<https://github.com/xamarin/Xamarin.Auth/wiki/Migrating-from-AccountStore-to-Xamarin.Essentials-SecureStorage>

באמצעות ה-API של ספרייה זו, יכולנו לשמור את ה-Access Token של המשתמש במכשיר, ולאחר הפעלה מחדש של האפליקציה, נוכל לפני שהוא עובר למסך ה-Login לבדוק האם נשמר כבר במכשיר Access Token ואם כן אז נוכל לבצע התחברות מהירה ולהתחיל את האפליקציה ישירות מהמסך הראשי.

אתגרים:

1. ה-Access Token הוא מוגבל בזמן, כלומר לאחר זמן מסוים (לדוגמא שעה עבור Access Token מ-Google) אי אפשר לבצע התחברות מהירה ויש צורך לבקש Access Token מחדש. הפתרון הוא שימוש ב-Refresh Token. למעשה, כאשר התחברנו עם Xamarin.Auth קיבלנו עוד Token בנוסף ל-Access Token שנקרא Refresh Token, יחד עם ה-Token הנוסף נוכל לבצע בקשה לחידוש הזמן של ה-Access Token. את כל זה אנחנו למעשה נעשה כבר בשלב ה-AutoConnect.

אולם, הספרייה Xamarin.Auth לא מספקת API עבור שימוש ב-Refresh Token ולזה מצאנו ספרייה נוספת, שהיא מעין הרחבה ל-Xamarin.Auth שנקראית: Xamarin.Auth.Extensions.

ניתן להעזר בעמוד ה-Github של הספרייה כאן:

<https://github.com/xamarin/Xamarin.Auth/tree/master/source/Extensions>

בנוסף למענה על הצורך בזיהוי המשתמשים, השתמשנו ב-Token המתקבל בסיומו של התהליך הנ"ל על מנת להירשם ל-Backend, מה שאפשר למנוע גישה לא מורשית לנקודות הגישה של ה-WEB API שלנו (למשל דירוג מקסימלי של השלט שלי ע"י כמה משתמשים שונים דרך Postman).

אופן השימוש ב-Token החברתי כדי להירשם ל-WEB API של Azure:

<https://github.com/Azure/azure-mobile-apps-net-client/issues/341>

**פיתוח צד-שרת:**

התחלת העבודה:

בשלב האפיון של הפרויקט הבנו שנצטרך להשתמש בבסיס נתונים (DB) לאחסון, ובגורם מקוון שיקשר בין המשתמשים באפליקציה למידע שב-DB (ראשית כל במחשבה להשמיט מימוש של שאילתות SQL מסורבלות מהקוד שב-Client, בהמשך גם משיקולים אחרים כמו הבטחה). לגורם זה גילינו בהמשך שקוראים WEB API, וגילינו שבאמצעותו נוכל לבצע הרבה מעבר מסתם שאיבת נתונים - אפשר לסנן ולעבד מידע מה-DB ואף לבצע קריאות HTTP לשירותים אחרים כחלק מחישוביו.

כל מה שנוגע ל-DB או ל-WEB API הוגדר כ"צד-שרת" או Backend והוטל על אחד מחברי הצוות.

כעת עמדנו בפני השאלה - אילו כלים קיימים כדי להקים Backend כזה?

בהשראת יומני המסע מהשנים הקודמות והמלצתו של גיא רונן בחנו את האפשרות להקים את ה-Backend באמצעות שירותי הענן של Azure. לאחר בחינה יסודית יותר ראינו כמה יתרונות:

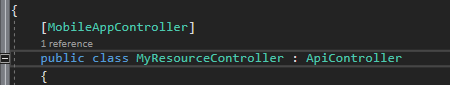
* ניתן להשתמש ברוב השירותים של Azure בחינם לתקופה של שנה עם מייל של חשבון סטודנט של האקדמית ([student\_mail@mta.co.il](mailto:student_mail@mta.co.il)), ובפרט DB ו-WEB API.
* התממשקות נוחה עם Visual Studio: מתוך הפורטל של Azure ניתן ליצור פרויקט WEB APIטמפלטי ב-VS כך שכבר יהיה מקושר ל-Mobile Service שהוקם ב-Azure ולשרת ה-SQL שהוגדר עבורו, כך שניתן להפעיל ולעדכן את ה-Backend בקלות ע"י ביצוע Publish על פרויקט זה מתוך VS.
* התממשקות נוחה בין ה-WebApi הטמפלטי שסופק ע"י Azure לבין קוד ה-Client שרץ באפליקציה עצמה באמצעות המחלקה Microsoft.WindowsAzure.MobileServices.MobileServiceClient
* ניתן לממש את כל מה שקשור ל-Backend באמצעות #C אשר אנו כבר מתמצאים בה.

לאור יתרונות אלו בחרנו להמשיך את העבודה עם Azure.

על אופן העבודה ב-WEB API והגדרת הנתונים ב-DB:

ה-WEB API הטמפלטי שסופק לנו ע"י Azure ממומש בטכנולוגיה הנקראת ASP.NET, המאפשרת להגדיר "נקודות גישה" אליהן Clients יכולים לשלוח HTTP Requests, ולקבל מהן HTTP Responses. באופן זה ניתן להגדיר ש-Path מסוים כגון https:myapp.net/api/myresource יכול לקבל קריאות GET ו-POST עם פרמטרים מסוימים, ובהתאם לסוג הקריאה להחזיר מידע פשוט או מורכב (בפועל מבוצע Deserialization על הערך המוחזר לצורה של JSON וסטרינג זה חוזר בגוף ה-HTTP Response).

נקודות גישה אלו מוגדרות ע"י הגדרת מחלקות היורשות מ-ApiController לפרויקט ה-WEB API כאשר שמן הוא זה שקובע את ה-path. למשל כדי ליצור את ה-path הנ"ל הנגמר ב-myresource היינו צריכים להוסיף לפרויקט מחלקה המוגדרת באופן הבא:



ואז לבצע Publish לפרויקט.

נקרא לנקודות גישה אלו מעתה בקרים או Controllers.

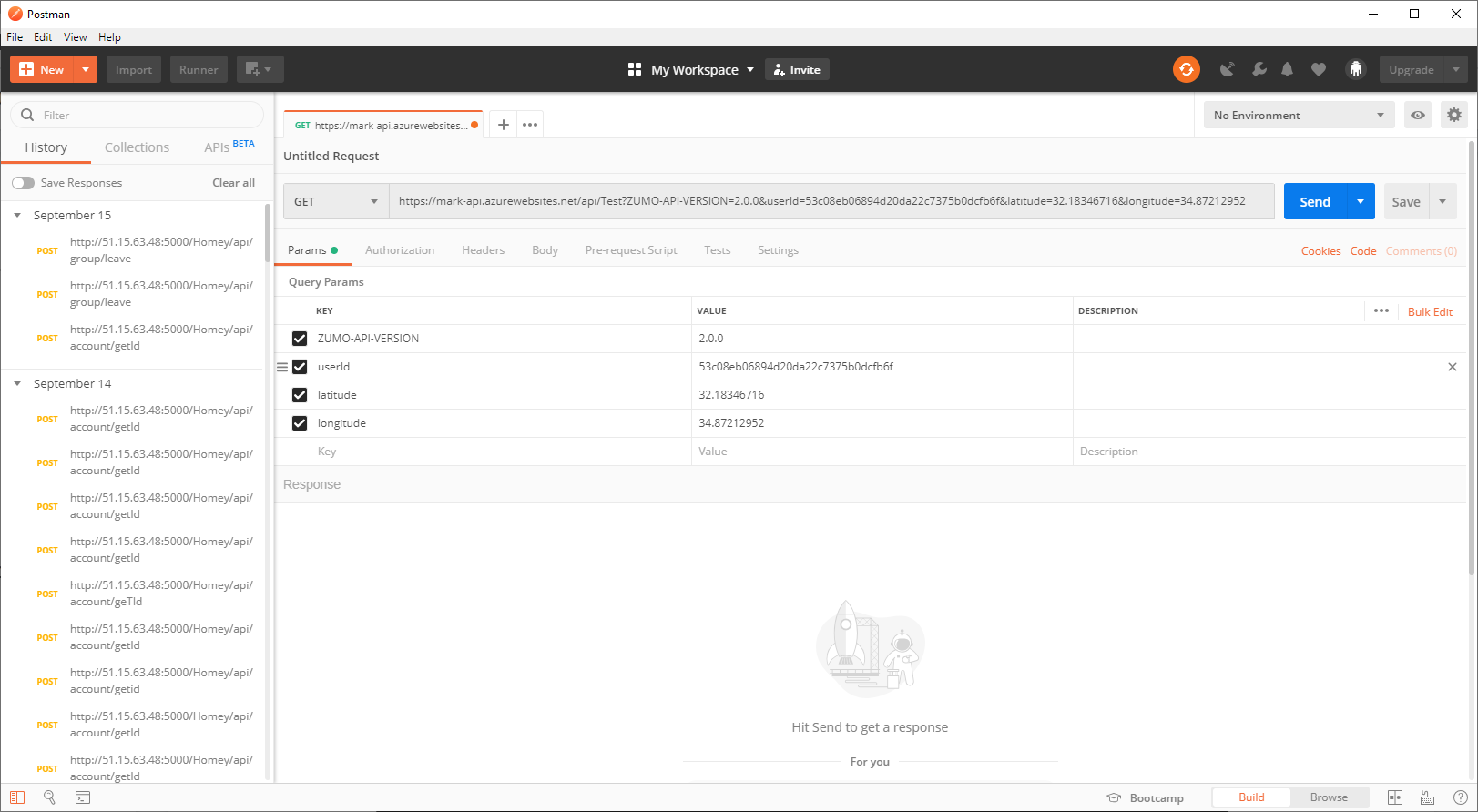
אחד הצדדים היפים לפיתוח ה-Backend באמצעות הכלים של Azure ו-Visual Studio הוא שניתן להגדיר את טבלאות ה-SQL שב-DB והקשרים ביניהם (foreign keys וכו') גם מבלי להכיר בכלל SQL.

ASP.NET משתמשת בטכנולוגיה הנקראת EntityFramework המאפשרת להגדיר את ה-data models והקשרים ביניהם כמחלקות #C, וע"י הגדרת automatic migrations המבנה החדש מתעדכן ב-DB ע"י ביצוע publish לפרויקט. (גישה זאת נקראת Code-First, בניגוד לגישה ההפוכה Database-First שמנגישה למחלקות #C את הנתונים ב-DB קיים).

חשוב לציין שישנם שני סוגים של בקרים: Table Controller שמטרתו להנגיש באופן ישיר יחסית את ה-DB, ו-Custom Controller שבו אפשר לבצע כל סוג של חישוב כולל גישה ל-DB. ואופן הגישה אליהם מה-Client באמצעות המחלקה Microsoft.WindowsAzure.MobileServices.MobileServiceClient הוא שונה (MobileServiceClient.GetTable לעומת MobileServiceClient.InovkeApi).

כלים:

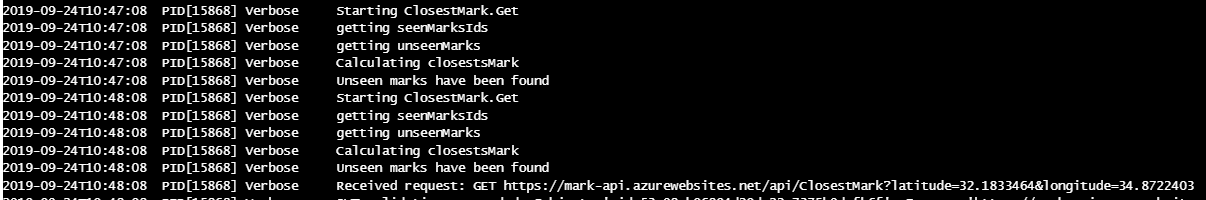
אחד הקשיים שעמדו בפיתוח ה-Backend היה לבדוק שנקודות הגישה שמומשו אכן עובדות כראוי, גם כשה-UI אינו מוכן, או כאשר ביצע הקריאה דרך ה-UI הייתה לוקחת הרבה זמן יחסית. התוכנה Postman איפשרה לנו לבצע קריאות HTTP ל-WEB API במאמץ מינימלי:



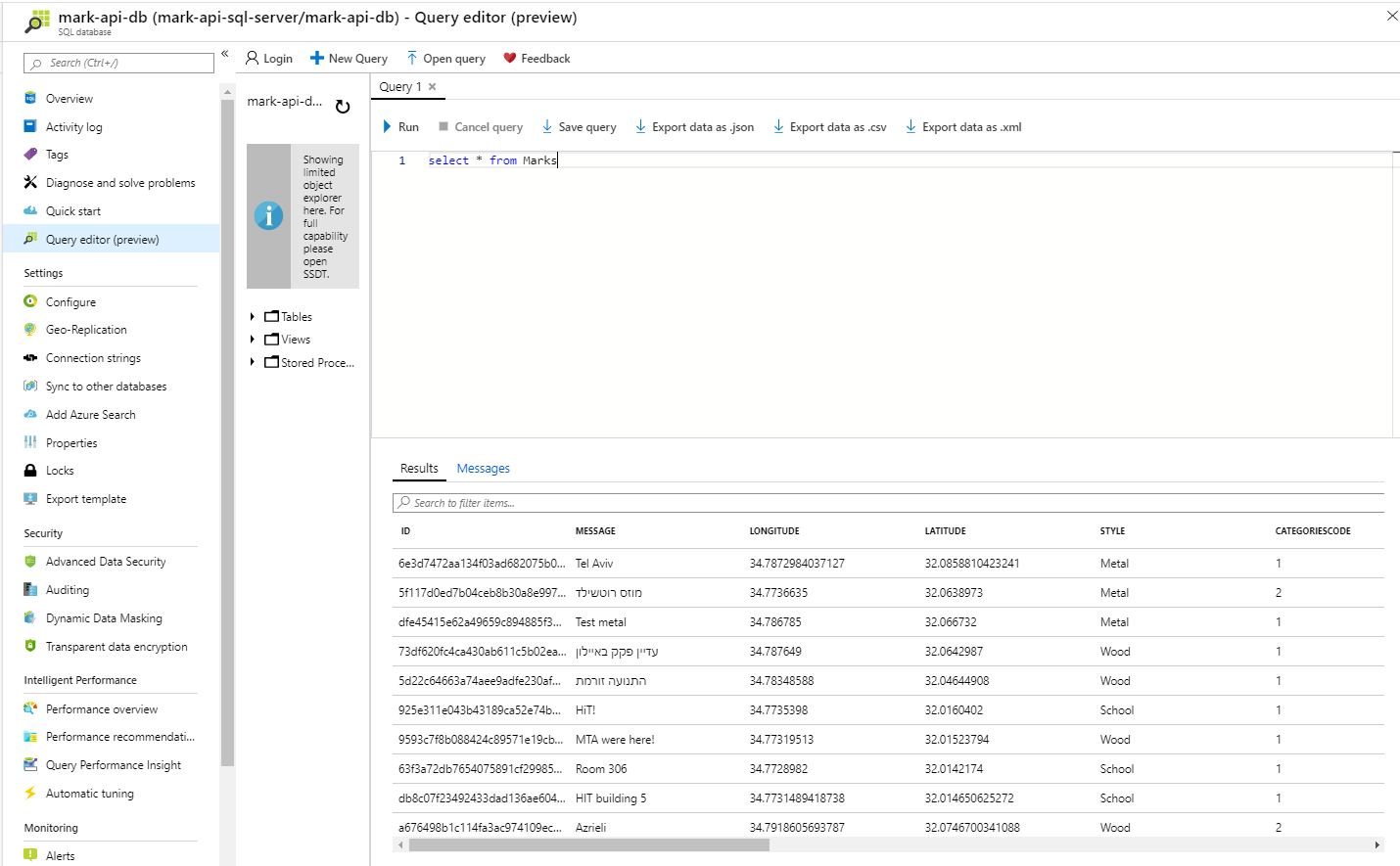
כדאי לשים לב שצריך לצרף את הפרמטר ZUMO-API-VERSION=2.0.0 לכל קריאה על מנת שהקריאה תצליח ל-WEB API של Azure.

קושי נוסף שבפיתוח ה-Backend לעומת ה-Client הוא הקושי בדיבוג. Azure מספקת פתרון אפשרי ע"י כלי ה-Log Stream המאפשר לראות במקום אחד ובזמן אמת את כל ה-Logs שבוצעו באופן הבא ב-WEB API:



ובכך ניתן לעקוב אחרי מאורעות רלוונטים ו-Exceptions:

כלי נוסף שהשתמשנו בו הרבה הוא ה-SQL Query Editor בפורטל של Azure שאפשר לנו לבדוק שהקריאות ל-WEB API אכן עדכנו את ה-DB באופן המתוכנן, ולהסיר מה-DB מידע באופן ישיר כאשר עלה הצורך:



לינקים שימושיים:

מדריך כללי ומאוד שימושי, אם כי מעט לא מעודכן. מומלץ להקים את ה-WEB API וה-DB באמצעותו:

<https://adrianhall.github.io/develop-mobile-apps-with-csharp-and-azure/>

על Database-First,Code-First בעבודה עם EntityFramework:

[Entity Framework 6 Tutorial: Learn Entity Framework 6 from Scratch](https://www.youtube.com/watch?v=ZX7_12fwQLU)

למקרה של הסתבכות עם Migrations:

<https://weblog.west-wind.com/posts/2016/jan/13/resetting-entity-framework-migrations-to-a-clean-slate>

**נוטיפיקציות:**

יש לשים לב שבדיבור על נוטיפיקציות יש שני סוגים עיקריים: נוטיפיקציות Push, ונוטיפיקציות מקומיות:

* נוטיפיקציה מקומית ניתן להציג בסמארטפון בדומה לכל אלמנט אחר של UI (מלבד ההבדל שניתן להציגה גם מתוך thread שאינו של ה-UI, ובכך למשוך את תשומת ליבו של המשתמש גם כאשר האפליקציה אינה פעילה).
* נוטיפיקציית Push היא סוג של אות שנשלח לסמארטפון ממקור חיצוני, אשר יכול לגרום להצגתה של נוטיפיקציה מקומית (נוטיפיקציית Push יכולה גם להעביר מידע לאפליקציה מבלי להסב את תשומת ליבו של המשתמש - נוטיפיקציה מסוג זה נקראת Silent Notification).

הבנו בשלב מוקדם שאנחנו נרצה להשתמש בנוטיפיקציות בפרויקט, אבל רק בשלב מאוחר למדנו על ההבדל הנ"ל. לכן התנסינו בנוטיפיקציות מסוג Push למרות שבסוף הצלחנו להגיע להתנהגות הרצויה באמצעות נוטיפיקציות מקומיות בלבד.

מימוש של נוטיפיקציות מסוג Push:

את אותו אות חיצוני שהוזכר לעיל ניתן לשלוח ע"י שירותים המיועדים באופן ספציפי למכשירי Android או iOS בנפרד. עבור מכשירי ה-Android השתמשנו ב-FireBase החינמי של גוגל, בעוד שעבור iOS היינו משתמשים ב-(Apple Push Notification Service (APNS לו אינו היה דורש חשבון מפתח של Apple אשר עלותו הכספית פסלה אותו מן הפרק (באישורו של גיא).

בנוסף השתמשנו בשירות שנקרא AppCenter כדי לקשר בין ה-Client ל-FireBase או APNS בהתאמה ע"י ה-NuGet שנקרא AppCenter.Push, וגם השתמשנו ב-API שלהם כדי לבקש שליחה של נוטיפיקציות Push מתוך קוד ב-WEB API שלנו.

על מנת לקבל נוטיפיקציות באופן הנ"ל יש על המכשיר להירשם לשירות ה-Push המתאים של AppCenter - כאשר הרישום של Android ושל iOS מתבצע בנפרד לפי ה-Key המתאים המתקבל מ-AppCenter.

עבור נוטיפיקציית Push כנ"ל קיימים שני תרחישים של קבלה במכשיר הסמארטפון: כאשר האפליקציה פעילה וכאשר אינה פעילה (ברקע או מכובה). באופן שרירותי למדי, כאשר האפליקציה אינה פעילה הנוטיפיקציה מוצגת ברמת דחיפות נמוכה (אינה "קופצת" מתקרת המסך, ניתן לראות אותה רק ע"י משיכה של "חלון הנוטיפיקציות" מתקרת המסך), וכאשר האפליקציה פעילה הנוטיפיקציה אינה מופיעה בכלל - אם כי אפשר לזהות את קבלתה באמצעות ה-debugger. כדי להציגה גם כאשר האפליקציה פעילה יש להוסיף קוד שבתגובה לקבלת נוטיפיקציית ה-Push מציג נוטיפיקציה מקומית (ל-Android ול-iOS בנפרד).

החיסרון בנוטיפיקציות מסוג זה אשר פסל מבחינתנו את השימוש בהן הוא הדיליי הפוטנציאלי מרגע הבקשה לשליחת ה-Push עד לקבלה אצל ב-Client. בניסויים רבים הנוטיפיקציות הגיעו באיחור לא מתקבל אם בכלל.

מימוש של נוטיפיקציות מקומיות:

חוסר העקביות של נוטיפיקציות מסוג Push הוביל אותנו לשנות גישה - במקום שהשרת יבדוק כל דקה האם יש מידע חדש אבל כל אחד מהמשתמשים, עבור כל משתמש האפליקציה תיזום פעם בדקה פנייה לשרת כדי לבדוק האם יש מידע חדש - ואם כן אז תציגו ע"י נוטיפיקציה מקומית. לגישה זו יש את היתרונות של אמינות בנוסף לכך שנוכל להציג "נוטיפיקציות" גם ב-iOS, אך היא ממש אינה האידיאלית מאחר והיא מכבידה על הסוללה וחבילת הגלישה של המשתמשים.

גישה זו הובילה לאתגר חדש - כיצד נפנה לשרת מהאפליקציה גם כאשר היא ברקע? או במילים אחרות, כיצד ניתן לממש Continuous long running task ב-Android וב-iOS? הפתרון ב-Android היה פשוט למדי - מפעילים טיימר ב-thread נפרד שבאינטרבל יבצע את המטלה הנדרשת. הקושי בעיקר עלה כאשר ניסינו לבצע את אותו הדבר ב-iOS מאחר ומערכת הפעלה זו מקפידה להשבית תהליכי רקע לא מוצדקים - thread נפרד היה מושבת תוך פחות משעה. לבסוף מצאנו פתרון בכך שניצלנו את העובדה שתהליך רקע שעוקב אחר מיקומו של המשתמש אינו מושבת בידי מערכת ההפעלה (זהו תהליך "מוצדק") וכמו כן את העובדה שמכשירי iOS מדווחים על שינוי במיקום גם כאשר המכשיר סטטי: השתמשנו במחלקה CLLocationManager אשר מדווחת על שינויים במיקומו של המכשיר, ובעזרת מדידת הזמנים שבין עדכונים אלו הפעלנו קריאה לשרת כל דקה.

**מסכי האפליקציה:**

