קורס ASP.NET CORE WEB API

כל.הזכויות.שמורות

מה זה WEB?

האינטרנט הוא רשת מחשבים גלובלית שמחברת ומקשרת רשתות מחשבים רבות ברחבי העולם. . הוא נבנה כדי לאפשר תקשורת בין משתמשים ברחבי העולם, גישה למידע ושירותים שונים כמו אתרי אינטרנט, דואר אלקטרוני ,ועוד.

עולם ה web בנוי על בסיס ארכיטקטורת.client server

Server - שרת – כשמו, מספק משאב כלשהו . מאחסן תוכן ומספק שירותים.

Client – לקוח - צורך (consumes) שירות כלשהו של השרת.

לכל צד במבנה הזה יש שפות שימושיות יותר :

לצד שרת למשל C, JAVA ו C#

לצד לקוח למשל HTML CSS ו.JS

Client ברוב המקרים הוא דפדפן, אבל ישנם עוד clients מסוגים שונים, טלפון טאבלט (אפליקציה) ועוד

ישנם דפדפנים שונים למשל כרום edge אקספלורר וכו

פרוטוקול HTTP

הדפדפן בעצם יודע להבין HTML ו JS ולצייר אותם.

התפקיד המשמעותי של הדפדפן הוא לנהל את התקשורת מול השרת.

השרת ,הסרבר, מחזיק את התוכן. הדפדפן יודע לקבל ממנו HTML ולהציג אותו.

התקשורת הזו, בין הקליינט לסרבר נקראת HTML Transfer Protocol : *HTTP*

TP זה פרוטוקול תקשורת , העברה HTML .זו השפה של הפרוטוקול הזה .

יש קו פיזי שמעביר מכתובת לכתובת (לדוגמה שני מחשבים) מידע, ושני הצדדים – הקליינט והסרבר - צריכים לדעת איך להעביר את המידע ואיך לפענח אותו . וזה ע"י ההגדרות בפרוטוקול. כמין הסכם של נתונים על ה"חבילה " שעוברת.

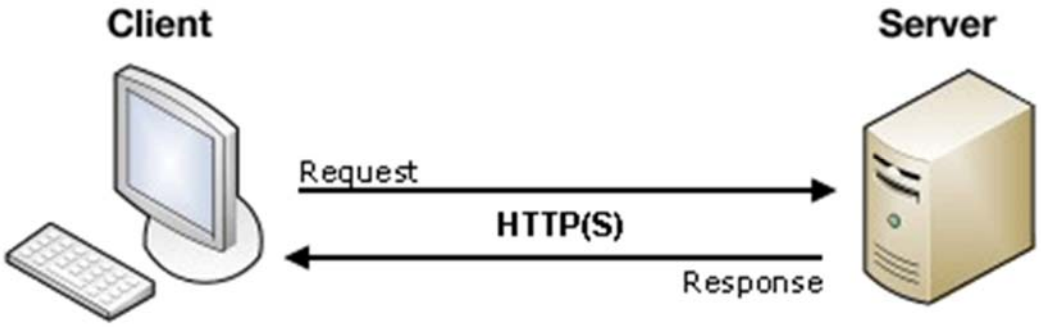
למשל, הנתון הראשון יהיה סוג החבילה, השני גודל החבילה, הנמען ועוד.

במילים אחרות, בפרוטוקול מגדירים באיזה אופן מבקשים ומקבלים נתונים בתקשורת המסוימת.

בפרוטוקול ה HTTP מוגדר איך ה HTML (קבצים בשפת HTML) עובר משרת אל קליינט וחזרה.

כל.הזכויות.שמורות

מודל REQUEST RESPONSE



נסביר את מבנה הרשת. המחשב מחובר לרשת ע"י:

1. תשתית .(לדוגמה בזק) זה הקו **הפיזי** המחובר מהקיר בכבל
2. ספק .שהוא מספק כתובת IP לכל צרכן וגם DNS

כתובת IP זה המזהה של כל פריט . כתובת זו ייחודית בכל מרחב הכתובות בעולם (לכן היא ארוכה) ותינתן לכל מכשיר שמתחבר לרשת .

לשרת יש כתובת IP קבועה בשונה מכל מכשיר שהוא קליינט שהכתובת שלו יכולה להשתנות .הספק יכול לתת בכל חיבור מחדש כתובת IP שונה.(כדי לקבל כתובת קבועה יש לשלם תשלום נוסף לספק).

DNS - dynamic name server זהו שם הניתן לכתובת IP.
אנחנו כותבים בדפדפן שם והספק יודע לפענח את זה לכתובת IP הנכונה (ע"י פניה לשרתי ה DNS שמחזיקים את כל השמות והכתובות שלהם) למשל כשכתבנו example זה מתורגם לכתובת IP כלשהיא נניח 194.50.2.66

תהליך טעינת דף אינטרנט

כניסה לאתר מסוים ע"י הקלדה בדפדפן כתובת מלאה או לחיצה על קישור.

לדוגמה https://www.example.com/default.htm

הכתובת נקראת , URL שבו יש את המידע כדי למצוא את הדף אליו אני רוצה לגשת

הדפדפן מיצר חבילה ושולח אותה (מכל קליינט שהוא) אל השרת .

הבקשה הזו נקראת .REQUEST

השרת מקבל בקשה, מעבד אותה ומחזיר לדפדפן נתונים כתשובה לחבילה שקיבל. זה בעצם ה.RESPONSE

מבנה ה REQUEST וה:RESPONSE

ה REQUSET / RESPONSE חייב להיות קובץ HTML שיכול לכלול גם קבצי JS ו CSS ותמונות בתוכו.
הדפדפן מציג ומצייר דברים על סמך התוכן שהתקבל.
התוכן מורכב מ headers שמתאר מידע כללי על התשובה, ו body שהוא התוכן עצמו.

דוגמה ל header של ה response זה ה.STATUS CODE

ישנם קודים שונים לסטטוס של ה response. בכל פעם חוזר קוד ב(header נפרט יותר בהמשך).

קומת ה200 מייצגת הצלחה,

400 שגיאה בצד הלקוח . הקליינט עשה משהו לא נכון.

500 זה כשיש שגיאה בצד שרת.

ויש גם 300 שזה תשובות שאומרות לדפדפן לעשות משהו. למשל 304 – תשתמש במה שיש לך ב CASHE מכיון שהתוכן שקיים לך כבר לא השתנה. 302 זה redirect לדף אחר ב body של הresponse יהיה ה location שאליו המשתמש יעבור.

הטבלה הבאה מסכמת את קודי התגובה הנפוצים שחשוב להכיר:

200	העברת המידע הצליחה
201	פריט המידע החדש נוצר על השרת, והתגובה מחזירה את כל הפרטים שלו כולל הid-
204	הבקשה מולאה, אבל לא מוחזר מידע בגוף התגובה .לדוגמה, כשמוחקים פריט מידע
301	הוזז לצמיתות - הפניית הבקשה הזו והבאות בעתיד לכתובת חדשה על השרת המרוחק
400	שגיאה בבקשה שהגיש צד הלקוח ולכן הבקשה לא תטופל
402	נדרש תשלום
403	הבקשה תקינה, אבל השרת לא מבצע בגלל שללקוח אין הרשאה
404	לא נמצא
405	המתודה אסורה. לדוגמה, כשרוצים לאסור על מחיקה של מידע.

כל.הזכויות.שמורות

שגיאה פנימית בשרת המרוחק	500

פרוט נוסף בעמוד 20

Server side -צד שרת

ההיסטוריה של האינטרנט מתחילה בסוף שנות ה90 אז התחילו הדיבורים הראשונים.

בעלי הענין העיקריים היו מתחום המחקר והביטחון. ובהמשך מה שפיתח את האינטרנט יותר היה המסחר.

בתחום המחקר: כמו אוניברסיטאות - התקשורת הראשונה היתה שם. מישהו כתב לחברו ."HI"

וכמובן צריך מי שישלם את המחקר וזה מערכות ביטחוניות ולכן משרד הבטחון האמריקני היה הראשון שמימן פרויקט כזה . רק שם פחות היה אכפת איך נראה העיקר שיעבוד ויהיה יעיל.

המסחר היה המנוע שגרם לזה להתפתח. סיפורה של חברת אמזון ממחיש מאוד את הנושא.

אנשים רבים התחילו להשתמש בזה .אמנם שוק האינטרנט סגר כמה מקומות עבודה אך פתח גם בעצמו ערוצי תעסוקה חדשים.

בצבא למשל יש רשת שחורה שבה אין כניסה ואין יציאה ויש רשת אדומה שמחוברת אך מנוטרת בזהירות. אם זה כל כך מפחיד למה אנשים מתחברים?

כפי שהסברנו המודל הזה ,של REQUEST ו RESPONSE נותן לנו הגנה. בפרוטוקול HTTP הקליינט מוגן, רק מה שהוא מבקש הוא מקבל .לכן הוא אופיין והוגדר כך. ואז יש בטחון להשתמש בו.

השרת לא יכול לגשת ללקוח. כשאני מבקשת לגשת לאתר אני בטוחה שהוא לא יכול לגשת לקבצים במחשב שלי בלי אישור, בשונה מאפליקציה שמותקנת על המחשב.

דפדפן למשל, הוא אפליקציה ולכן יש לו גישה לקבצים והוא גם עושה דברים בעצמו . זו אפליקציה שאנחנו נותנים בה אמון שלא יעשו דברים נגדינו. כמובן שהגישה מתאפשרת רק לאחר אישור של המשתמש.

הקשר בין השרת ללקוח

צד השרת , ה SERVER לא יכול לתקשר עם הקליינט מלבד להחזיר תוכן שהלקוח ביקש.

זאת מכיון שפרוטוקול Http הוא – *stateless* חסר מצבים. המשמעות היא שלאחר שהסתיים תהליך של request שאותו יוזם הקליינט ומתן תגובה response – שזה בסמכות הסרבר, לא נשמר שום מידע לגבי התהליך. בקשה נוספת תיחשב כבקשה חדשה, אלא אם כן נעשה שימוש ב cookies או מנגנונים אחרים שיסייעו לשרת לזהות בקשות חוזרות מאותclient . יפורט בהמשך.

כל.הזכויות.שמורות

נשים לב שכל תוכן חדש דורש בקשה חדשה לשרת. בתקשורת אנו פונים לקבל מידע מהשרת וקבצי ה RESPONSE נשמרים בתיקיה זמנית והדפדפן מציג משם את התשובה ל REQUEST. אך במידה ויש מידע חדש בשרת הלקוח לא יודע שזה עודכן, עד שיפנה שוב לשרת.

אפליקציות ואתרי ווב לרוב משמשים לצרכים של מידע דינמי.

כשאנחנו מדברים על צד לקוח בלבד, זה בעצם אוסף של דפי HTML סטטיים - קבועים - כלומר שלכל מי שיכנס בכל עת ובכל שעה יקבל את אותו דבר. מבחינת הresponse.

באפליקציית ווינדוס כמו למשל מחשבון, היכולת של מסכים ללא הגבלה והתקשורת ביניהם, כמו גם השליטה מלאה מה המשתמש יראה ומה יקרה לו, טובה ומצוינת .

ב WEB האתגר שלנו הוא לקחת את דפי ה HTML הסטטיים וליצור אפליקציה דינמית. וזה יהיה ע"י שניצור קשר בין request ל Request. לקחת אתר שהוא אוסף של דפים סטטיים ולהפוך אותו לאפליקציה.

לדוגמה:

1. חנות ספרים המשתמש בחר ספר כשחוזר לדף של כל הספרים נרצה לסמן לו שהספר הזה סומן
2. ספריה שמציגה ספרים לפי תחומי ענין של המשתמש שמחובר– הקטגוריות שמוגדרות לו.
   נניח שנכנסו לאתר והגענו לדף LOGIN (כניסה) עם משתמש וסיסמה. בלחיצה על כפתור מופעל ה REQUESTהראשון ומגיע אל השרת שמחזיר RESPONSEשל הצלחה: קוד 200 או 400 כשלא הצליח.
   עכשיו אני רוצה לקבל פרטים של קטגוריות שרלוונטיים ליוזר שעימו התחברתי.

איך אני יכולה ליצור קשר בין ה REQUEST->RESPONSE לבאים אחריו באותו SESSION ?

<u>SESSION</u> –זהו פרק הזמן מהרגע שנכנסתי לדפדפן עד שיצאתי מהאתר או סגרתי את הדפדפן.

יש שיטות שונות לייצר את הקשר והדינמיות הזו:

- Query string
- Cookies
- Local storge

ופתרונות ברמת שרת:

- ASP.net pages
- SPA

ולפני שנרחיב על כל שיטה נלמד על :

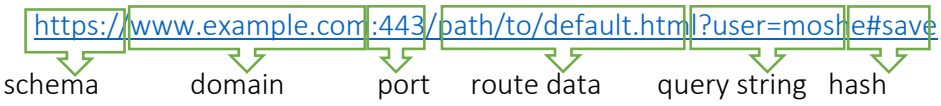
<u>חלקי הURL</u>

כשאנחנו ניגשים לכתובת מסוימת אנו רושמים URL שהוא מורכב מכמה חלקים:

1. *<u>פרוטוקול/סכמה</u>* http/https security – שאז התקשורת מוצפנת.
2. *<u>Domain</u>* - השם של האתר

כל.הזכויות.שמורות

3. *PORT* - השער דרכו עובר המידע ברשת. לרוב אין צורך לציין.
port 80 מקובל בפרוטוקול HTTP ו -port 433 בHTTPS .
4. *Rote data* - הדף שאליו אנו רוצים לגשת בתוך האתר. לא תמיד זה דף פיזי, זה יכול להיות נתיב וירטואלי.
5. *queryString* - שאילתה שהקליינט שולח לשרת . הטקסט שמופיע אחרי הסימן שאלה. בצורת KEY VALUE
6. *HASH* - התו -# הוראה לדפדפן לגלול לסימניה מסוימת בדף



- **Query string**

שיטה פשוטה ובסיסית להעביר טקסט כשאילתה שנשלח גם לשרת.

ב URL נציב סימן שאלה ולאחריו פרמטרים עם ערכים.

לדוגמה username=avi ואז השרת מחזיר מידע שרלוונטי רק למשתמש הספציפי וכך יש לנו קשר בין request לrequest .

החסרונות: 1. חשוף ולא מאובטח 2. URL ארוך ומסורבל 3. מוגבל במספר התווים. זה לא מספק כדי לשים הרבה פרמטרים.

כמו"כ צריך לדאוג לשתול ב URL בכל פעם.

- **COOKIES**

אלו קבצים קטנים עד 20 KB שמכילים נתונים ושמורים על המחשב של הקליינט שומר באותו SESSION ויכתוב או יקרא במעבר בין דפים וכך נוכל ליצור קשר בין בקשה לבקשה.

ניתן להגדיר לקוקי תפוגה , expiration כך זה יכול לשמור גם ל SESSIONS הבאים.

איך הקוקי ישמר מקריאה של אתר אחר? הדפדפן לפי הפרוטוקול שומר את ה cookies ברמת דומיין ולכן אנחנו סומכים שלא ישתף בין אתרים אחרים.

כחלק מהפרוטוקול של HTTP כל קוקי שנוצר בקליינט בכל request הם נשלחים ב headers לשרת ואז ב response הם חוזרים ויכולים להשתנות ואפילו להימחק .

היתרון הוא גם חסרון. לא כל דבר יש טעם לשלוח לשרת, זה מגדיל את גודל הבקשה והתשובה ומשפיע על המהירות.

- **Local storge**

ניתן לשמור נתונים בדומה ל cookiesאך נשמרים ברמת הלקוח בלבד.

זו יכולת של הדפדפן מאז HTML5

הנתונים לא עוברים לשרת מחליף את השימוש האינטנסיבי בcookies

יתרונות: יותר יעיל- יותר רזה ונוח

- ASP.net pages

עבור הצורך המסחרי נוצר הASP - Active Server Pages

דפים שהם פעילים בצד השרת. היוזר שולח בקשה לדף מסוג ASP כשהשרת מקבל בקשה זו לפני שהוא מחזיר תשובה הוא מריץ את הקוד שכתוב בתוך הדף (#C) מתרגם את זה ל HTML ומחזיר response .

צד השרת מחזיק את הנתונים – יש לו גישה ל DB . כך לדוגמה שולפים את רשימת הספרים שמורשים ליוזר ומיצרים ממנו דף HTML זה חוזר ללקוח והדפדפן מציר את זה. כך כל יוזר יקבל דף דינמי.

דפי ה ASP נראים כמו דף HTML שמכיל חלקים מתויגים של קוד שכל בלוק כזה רץ בשרת ומתורגם לHTML

ASP.net זוהי גרסה יותר מתקדמת שמקלה על המתכנת בכתיבת הקוד.

<u>ASP החסרונות</u>

בכל פעם נשלח כל הדף לשרת , משתמש חווה איטיות, דף לבן וקפיצות (scroll)

לדוגמה: בחירת ארץ ומדינה

המשתמש צריך להמתין זמן רב כי הנתונים רבים וכבדים, חווית המשתמש לא טובה ולפעמים בלתי נסלחת

- SPA

Single Page Application

לאתר יש דף אחד ללא ניווט בין דפים כך שאינו צריך להתרענן.

הפרוטוקול הוא XHTTP – מחזיר XML

XML, בדומה לHTML, היא *markup language* שפה המתארת נתונים/מידע באמצעות טקסט ומוכרת בכל מחשב שהוא (בשונה מקובץ word למשל)

מבנה ה XML, כמו HTML : פותחים תגית וסוגרים באמצע אפשר לרשום נתונים וגם מאפיינים על התגית למשל <user lastName="cohen">moshe<user/>

בXML משתמשים לתיאור של DATA וב HTML גם איך ה DATA יראה- מכיל גם אובייקטים וכו.

ב request הראשון קיבלנו HTML הדפדפן מצייר אותו ובלחיצה על תפריט כל שהוא במקום שהדפדפן יעבור לכתובת אחרת (window. Location) כותבים קוד בJS (למשל) שמוציא REQUSET לשרת. השרת מחזיר תשובה לקוד ולא משתלט על כל הדף.

כך שבמקום לקבל דפים נקבל מידע, הטירדה של הקליינט היא לחולל דפים ולצייר אותם ולא של השרת.

השרת צריך רק לספק מידע. המידע לא קיים בצד הלקוח.

כל.הזכויות.שמורות

השיטה הזו נותנת חוויית משתמש הרבה יותר טובה מבחינת המהירות והנראות

ככל שנמעיט בריענון הדף כולו ונקטין את גודל הREQUEST וה RESPONSE העבודה באתר תהיה יותר יעילה ונעימה.

קריאת השרת בJS נקראת AJAX בשפות קליינט מתקדמות מוכרים המושגים POST ו FETCH . אלו בעצם פקודות ששולחות בקשות לשרת ומגדירות את הצורה בין היתר גם את הcontent type- באיזה צורה יחזור ה DATA ברירת המחדל זה JSON/XML

לרוב JSON שימושית יותר היא נוחה גם כי זו הצורה לכתוב אובייקטים ב JS וגם כי זה יותר רזה מ XML שם כל תגית נכתבת פעמיים בפתיחה ובסגירה.

לסיכום

ניסינו להבין את השם של הקורס

ASP.net זו בעצם התשתית והטכנולוגיה. כמובן שיש גם שפות אחרות

CORE זו הגרסה המתקדמת של dotnet . הגרסה האחרונה והיציבה היא 8. dotnet framework זוהי התשתית לכתיבה בC# ובשפות נוספות. היא נותנת לנו את הCLR – מנוע הריצה ועוד ספריות רבות.

WEB API – ASP נותנת לנו לפתח סוגים שונים של אפליקציות ווב, למשל WEBFORM ועוד. בקורס שלנו נלמד על היכולת של WEBAPI שהיא חושפת שרותי אינטרנט ללא תצוגה.

API

API- **A**pplication **P**rogramming **I**nterface

זהו בעצם אוסף של פונקציות כדי לקבל מידע מהשרת.

GUI /UI - Graphic /user interface הינו ממשק למשתמש הכולל גם חוויית משתמש כמו איפה יופיע כל כפתור בדף ובאיזה הנפשה וכו.

בניגוד לממשק UI , API הינו ממשק שאין משתמש קצה שמפעיל אותו בד"כ אלא אפליקציה אחרת צורכת ומפעילה אותו.

נזהה את ההבדל בין API לWEB API :
DLL- Dynamic Link Library בשונה מקובץ EXE שזה תוכנית רצה, אלו ספריות שאפליקציות אחרות יכולות להשתמש ע"י הוספת קישור לאפליקציה.
DLL זו דוגמה לAPI שלא נגשים אליו דרך הWEB, הוא יושב אצלי לוקלית, מקומית. לעומת זאת ב WEB API השרות יושב בשרת אחר וכדי לגשת אליו יש לפנות בקריאת HTTP.

יש אתרים שמחזיקים בעצמם DB המכיל מידע ויש מצב של מספר אתרים המציגים נתונים שבהכרח לא שמורים אצלם. אתרים אלו נגשים לשרת אחר, לAPI, שמספק להם את הDATA.

API & WEB API דוגמאות

כל.הזכויות.שמורות

- ל WORD יש תכונה של הצעות איות , היות ומותקן לי על המחשב וורד ארצה לנצל את היכולת הזו גם באפליקציה אחרת, משתמש מכניס טקסט. בתוכנית שכתבתי הוספתי קישור , reference ל DLL של WORD והפעלתי את התוכנה של האיות . זוהי דוגמה לAPI רגיל.
- נתאר לעצמינו שעלינו להזמין כרטיס טיסה. בכל אתר. סוכן נסיעות שנפנה, נתבקש להגדיר כמה נוסעים, תאריך יעד ומהיכן. התוצאות במקומות השונים תהיינה זהות, כיצד? זאת אודות לכך שכל האתרים פונים ל API של רשות שדות התעופה שנותן להם מידע על הטיסות הפנויות. זוהי דוגמה לAPI WEB.

דוגמאות נוספות, אתרי תחזית מזג אויר הפונים לAPI של גוגל, מפות ועוד.

היסטוריה של API WEB

בעבר השתמשו בweb services להם היו פרוטוקולים מיוחדים שאפשרו לעבוד עם שירותים שעוברים דרך הWEB.(במקביל יש סרוויסים שונים שרצים לוקלית על נמחשב כמו אנטי וירוס.)

הפרוטוקולים היו מורכבים מאוד מכיון שנתנו יכולות רבות, התאמה לשפות ופרוטוקולים שונים, אך עם זאת נוחים כי ניתן היה לחלוק שירותים מתוכניות שרצות בשפות שונות.

בין הפרוטוקולים מוכר פרוטוקול Simple Object Application SOAP שהוא מבוסס XML וכן WCF.

RESTFUL API

בשנים האחרונות רווח השימוש ב RESTFUL API

REST (REpresentational State Transfer)- זוהי ארכיטקטורה שמתארת את אופן בנית שרתי ווב בצורה פשטנית ע"י שימוש בפרוטוקול HTTP. מגדירה אפשרות פשוטה לכל לקוח שהוא להתחבר לשרת.

הרעיון שעומד מאחורי זה שימוש בפרוטוקול HTTP שמספק לנו הרבה יכולות. כמו היכולת של GET וPOST. הרי כדי לקבל את הדף הראשון של האתר עשינו קריאת GET וכדי לשלוח את הדף לשרת (כמו בSUBMIT) השתמשנו ב POST. זהו בעצם הפרמטר METHOD שקיים בפרוטוקול HTTP.

השימוש המקובל הוא GET – לקבל מידע ו POST לשלוח נתונים לשרת. אם כך, במקום ליצר סרוויס בפרוטוקול מסובך נשתמש בבסיס של HTTP כדי לבצע קריאות API. כי הרי HTTP יודע לשלוח XML/JSON וזה מה ש API צריך , לא יותר מזה, כך ננצל את היכולות הרבות של HTTP ונשתמש בשפה אחידה ומובנת.

הסברנו שלאתר יש תוכן סטטי שמאוחסן בשרת – דפי HTML JS ו CSS שחוזרים לקליינט עמ"נ לרוץ שם. במקרה של RESTAPI נשתמש כשאנו רוצים לקבל תוכן של **נתונים**(DATA) מהשרת.

ב RESTFUL API יש להגדיר:1. URI בסיסי 2. Method 3. Media type למשל JSON או XML

כל.הזכויות.שמורות

בצד שרת נגדיר resource- משאב עם מזהה (ID), שעליו נרצה לעשות פעולות שונות – method.

כדי לפנות לAPI נרשום לאחר הדומיין את המילה API (כך מקובל, לא חובה) ואח"כ את שם המשאב

סוג הפעולה שתתבצע תקבע ע"י הmethod, חשוב לשים לב שלמרות שהURL זהה המטודה שונה.

HTTP METHODS

תוצאה	הערה	דוגמה	URL	Method
רשימת כל הספרים	-	Api/books	Api/entity	Get
ספר עם מזהה 123	-	API/books/123	API/entity/ID	Get
נוצר ספר חדש	יש לשלוח ב BODY את פרטי הספר החדש	Api/books	Api/entity	Post
עודכן ספר 123	יש לשלוח ב BODY את פרטי הספר לעדכון	API/books/123	Api/entity/ID	Put
נמחק ספר 123	-	API/books/123	Api/entity/ID	Delete

נשים לב:

- בPUT /POST השרת מקבל את הבקשה וקורא את הBODY וממנו יוצר ספר חדש או מעדכן קיים במידה ונשלח ID - מזהה הספר, שהינו פרמטר חובה במטודת PUT. (ניתן לשלוח את המזהה רק ב BODY אך זה פחות בטוח ומקובל שלוח גם כפרמטר וגם בגוף הבקשה).
- בתוך הפונקציה עצמה כמובן יש מקום לשיקולים עסקיים שונים למשל בפונקציה של PUT נרצה לבדוק שתאריך הלידה של הסטודנט תקין. או בDELETE לא באמת מוחקים את הישות אלא מסמנים כנמחק.

המונח CRUD מוכר בעולם התכנות כסט הפעולות הבסיסי על כל ישות : create, read, update, delete

בצד שרת אנו אחראים לתת את סט הפעולות האלו ע"י קריאות HTTP

מה הופך מחשב לשרת?

הסברנו שלא כל תוכן במחשב המחובר לרשת הינו חשוף לרשת, אם כך איך ניתן לגשת למחשב? בעצם כל מחשב יכול להפוך לשרת, כשמתקינים תוכנת שרת.

מלבד ששרתים צריכים להיות חזקים ומרובים, בכדי שאי מי (,גם ברשת הפנימיתintranet -,) יוכל לגשת עליהם צריך להתקין תוכנת סרוויס, שרות שמאזין לבקשות ומחזיר תגובות.

תוכנות לדוגמה - Apache Tomcat שרת ווב חינמי ל Linux, IIS -שרת ווב של מייקרוסופט, הן מאזינות לפורטים של תקשורת (80,443) שמוגדרים להן וברגע שמתקבלת בקשה מעבדות ומנווטות לשרות המבוקש ומחזירות תגובה מתאימה.

התוכנה הזו כבר מוכנה ולא משנה איזה שרת יארח אותנו אנו לא נתעסק בתוכנה הזו אלא בכתיבת השירותים עצמם. כמובן שיש לתוכנות האלו הגדרות אלו קבצים לחשוף ועוד.

אנו נשתמש במהלך הפיתוח בkestrel. זהו שרת שמגיע כחלק מסביבת הפיתוח בVS. פעם זה היה נקרא IIS EXPESS כדי למנוע חשיפה לאינטרנט במהלך הפיתוח ע"י התקנת IIS. זה מגיע כחלק מההתקנה של VS. בשנים האחרונות מיקרוסופט עברה ל kestrel שהוא יודע לרוץ גם על Linux וmac.

Visual studio code

נפתח בסביבת VS CODE

זה חינמי, OPENSOURCE קוד פתוח בGitHub הוא עובד בשיטה פתוחה.

אמנם הוא רק text editor אבל זה העוצמה שלו שהוא נורא רזה ואנחנו יכולים להוסיף בו כל מיני הרחבות לפי היכולות שאנחנו רוצים.

למשל בקורס הזה נרצה את dotnet– התשתית כדי לכתוב C#. כך נכיר ונבין את הפקודות יותר טוב.

יש את המאגר של nuget זהו מאגר עצום של ספריות קוד פתוח:
בעבר מיקרוסופט היתה נותנת כחלק מההתקנה של VS המון ספריות.
בCORE זה השתנה, ובVS code במיוחד. ההתקנה היא מינימלית ורזה, אין טעם להתקין סתם את כל הספריות ולהעמיס על המחשב, מה שנצטרך נמשוך מנוגט.

כל.הזכויות.שמורות

יצירת פרויקט

- נפתח תיקיה חדשה
- נפתח חלונית של טרמינל – ctrl+~, בתוך הטרמינל הזה נכתוב את הפקודות.
- נבדוק האם קיים התקנה של דוטנט 6 ואם לא – להתקין: dotnet --list-sdks
- כדי ליצור פרויקט חדש נקליד : dotnet new webapi
- לפעמים ידרוש אימות "Unable to configure HTTPS endpoint." במקרה כזה נריץ : dotnet dev-certs https –trust ונאשר.

נוצר לנו פרויקט ברירת מחדל של תחזית מזג האויר. השם נגזר משם התיקיה.

זהו בסיס לWEBAPI :

בתוך הפרויקט נראה מחלקה דיפולטיבית שמגדירה איך נראה ישות של מזג אויר (תאריך, טמפ', וכו). מקובל לשים ישויות בתיקיית models, קבצי הגדרות, קובץ program.cs שזהו main, ו controller – שבתוכו יש פונקציה שתופעל בבקשה של - get בדוגמה יש פרוט רק ל GET פשוט.

הרצת הפרויקט

- נריץ את הפרויקט ע"י הפקודה: dotnet run
- כדי לבנות בלי להריץ ולראות שעובר קומפילציה : dotnet build

נוצרו תיקיות BIN ו . OBJב bin יש קבצי exe – dll של ה API שיצרנו והספריות שהוא צורך.

נוצר לנו אתר ב- LOCALHOST יושב על המחשב המקומי. שרת הווב שמאזין לבקשות, מעביר לאפליקציה הזו את הפניות ל port המסוים בו נבנה האתר. נגלוש אל ה URL שקיבלנו בדפדפן, נשים לב שיש 2 פורטים אחד בhttp ואחד ב https ניתן לשנות את הפורט בקובץ launchSettings.json

Controller

Controller הוא מחלקה ציבורית עם public Method אחת או יותר הידועה בשם actions . על-פי המוסכמות, controller ממוקם בספריית controllers. הפעולות נחשפות כנקודות קצה של HTTP באמצעות ניתוב. לכן, בקשת HTTP GET ל- https://localhost:{PORT}/weatherforecast גורמת לפעולת השירות Get() של המחלקה WeatherForecastController להתבצע.

Controller יורש מ **ControllerBase** זה נותן פונקציונליות לטיפול בבקשות HTTP כך שנוכל להתמקד בלוגיקה עצמה.

[ApiController] מאפשר התנהגויות שמקלות על בניית ממשקי API של אינטרנט. למשל: attribute routing ושיפורים בטיפול בשגיאות .

[route] מגדיר את תבנית הניתוב , הסימון [controller] מוחלף בשם ה controller (case insensitive, ללא הסיומת Controller).

בתוך API אחד יכולים להיות מספר שירותים שונים אך קשורים נניח בעולם בית הספר יש מורים תלמידים כיתות ציונים ועוד עבור כל תחום נקים controller ייעודי.

כל.הזכויות.שמורות

היתרונות של יצירת ממשקי API ב- ASP.NET Core

באמצעות ASP.NET, נוכל להשתמש באותה מסגרת ובאותה תבנית כדי לבנות דפי אינטרנט ושירותים. משמעות הדבר היא שניתן לעשות שימוש חוזר במחלקות מודל ובלוגיקת אימות, ואפילו לשרת דפי אינטרנט ושירותים זה לצד זה באותו פרויקט. לגישה זו יתרונות:

Simple serialization : ASP.NET תוכנן עבור חוויות אינטרנט מודרניות. לכן עורך באופן אוטומטי את ה class שנגדיר ל- JSON כיון ש HTTP מחזירה רק טקסט. אין צורך בתצורה מיוחדת.

Authentication and authorization : עבור אבטחה, נקודות קצה של API כוללות תמיכה באימות והרשאות.

Routing : ASP.NET מאפשר להגדיר ניתוב בתוך הקוד באמצעות attributes. הבקשה מנותבת אוטומטית לפונקציה המתאימה.

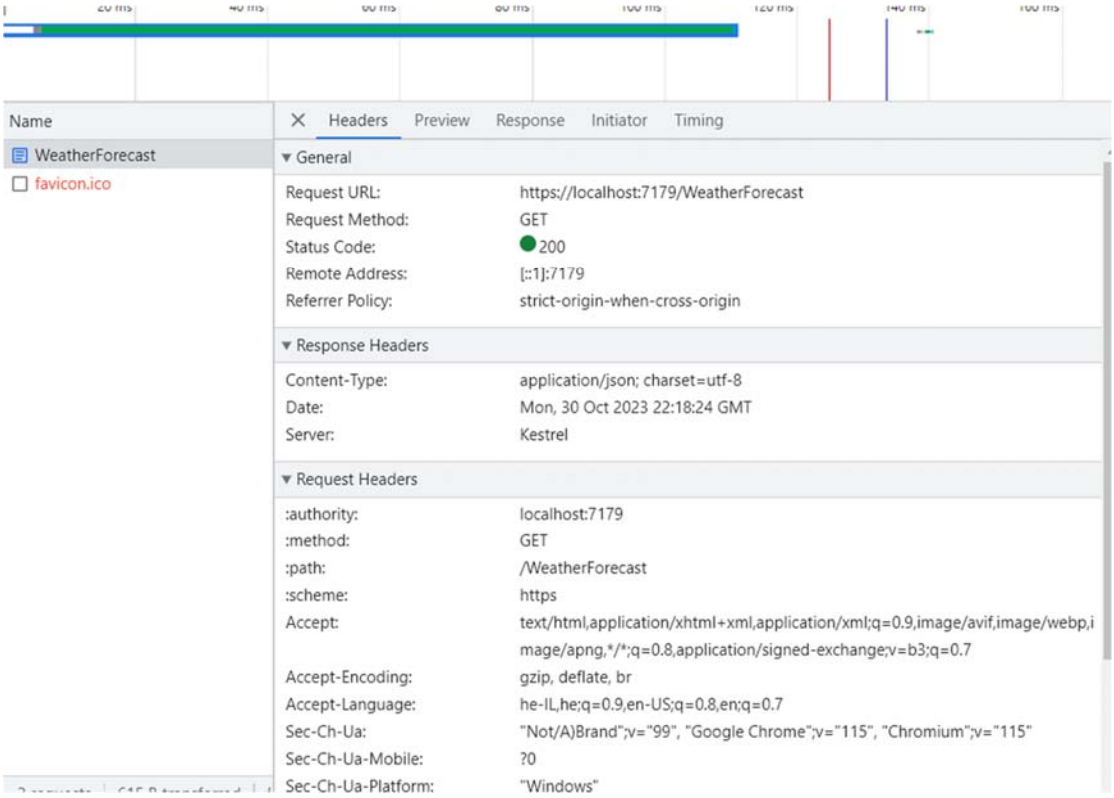
HTTPS כברירת מחדל: HTTPS הוא חלק חשוב בממשקי API אינטרנטיים מודרניים ומקצועיים. הוא מסתמך על הצפנה מקצה לקצה כדי לספק פרטיות וכדי להבטיח שקריאות ה- API לא ישתנו בין הלקוח לשרת. ASP.NET מספק תמיכה עבור HTTPS . הוא יוצר באופן אוטומטי אישור בדיקה ומייבא אותו בקלות כדי לאפשר HTTPS מקומי, כך שנוכל להפעיל ולאתר באגים ביישומים בזמן פיתוח.

בדיקה של אפליקציית WEB API

עמ"נ לבדוק בסביבת הפיתוח יש מספר אפשריות כמובן שעולם האמיתי יש אפליקציה שצורכת את הAPI:

- לפונקציה של get ניתן פשוט לגלוש בדפדפן. ב dev tools בטאב network נוכל לראות את ה request והresponse.

כל.הזכויות.שמורות



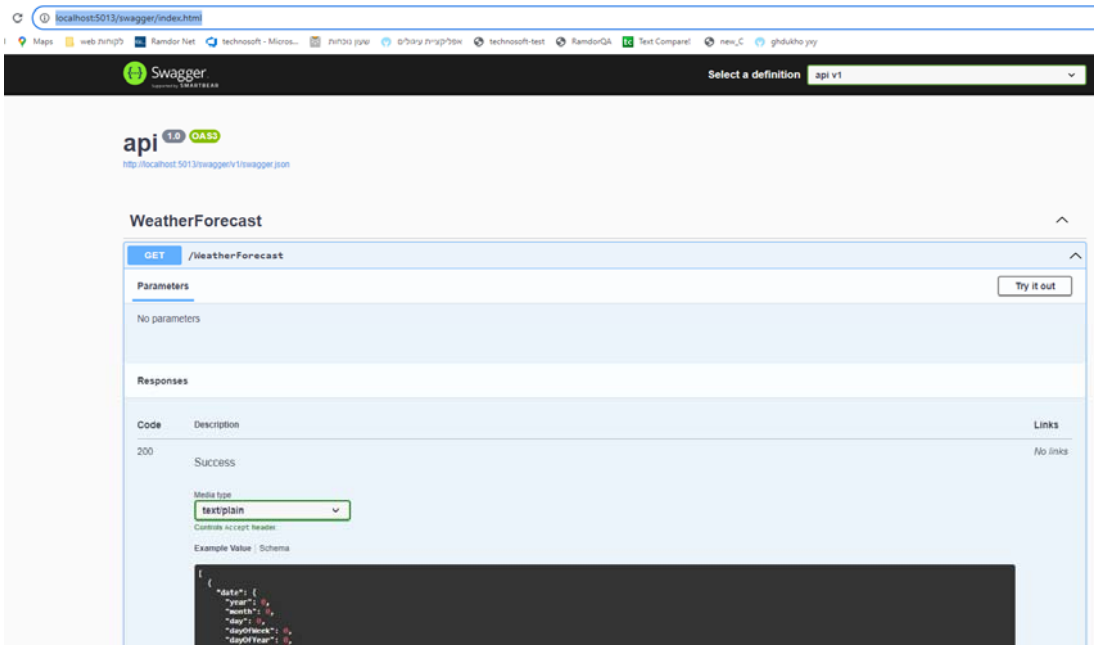
- Swagger

לשאר המטודות נוח מאוד להשתמש בswagger.

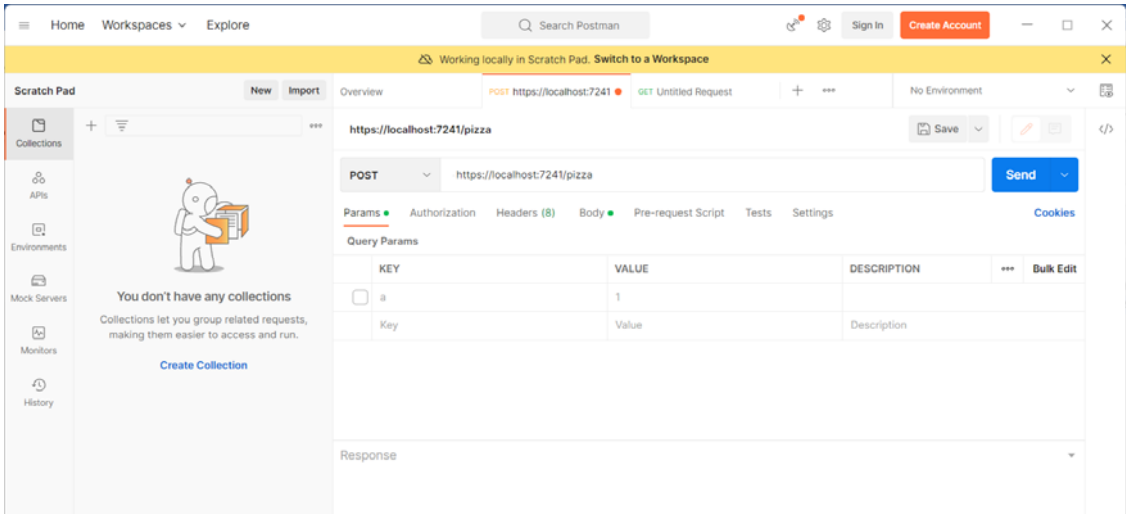
זוהי ספריה שמיועדת לטסטים, בדיקות . היא מציירת דף HTML שמראה את כל הפונקציות והסכמות של ה . API בנוסף, נותן להריץ request ולקבל .response

כדי לגשת נכנס לכתובת: http://localhost:[port]/swagger/index.html

```
builder.Services.AddSwaggerGen();//create openapi.json
if (app.Environment.IsDevelopment())
{
//adds swagger ui only in development env
app.UseSwagger();
app.UseSwaggerUI();
}
```



- postman זוהי פלטפורמה למפתחי API לניהול request



Attribute routing

טכנולוגיית ה ASP .NET CORE נותנת לנו להגדיר תגיות לניתוב, המפשיט את הניתוב לcontroller והפעולה הרצויים.

למשל אם נרצה בתוך controller של employees לכתוב 2 actions למטודה get , ביצד השרת ידע מתוך ה URL שהקליינט ביקש לאיזה פונקציה לפנות?

לכן נדאג ל routing ייחודי עבור כל action בכל controller . חשוב להבין שעבור כל משאב חייב להיות URL ייחודי .לא ניתן לגשת ליותר מפונקציה אחת באותו URL אך ניתך לגשת לאותו משאב עם מספר URL שונים במידה והגדרנו מספר ניתובים.

במידה ולא נגדיר ניתוב ייחודי לכל אחד מה action של כל method נקבל שגיאה של AmbiguousMatchException

ישנן מספר אפשרויות לrouting :

1. מחרוזת ייחודית קבועה.
2. פרמטרים
3. Token replacement
4. דריסה של הניתוב הבסיסי

ניתן לבצע הגדרות routing כחלק מהגדרת ה method.

```
[HttpGet("routing...")
```

כאשר נרצה לקבל נתונים של ישות ששייכת לישות אחרת, לדוגמה: ציונים של תלמיד, מקובל לרשום ניתוב שמכיל את שני האובייקטים כמו : student/{id}/grade.

שימי לב :[ApiController] מחזיר 400 כשלא מוצא לאן לגשת.

פרוט:

1. מחרוזת ייחודית קבועה.

אם דברנו על הemployeesController ניתן לפונקציה 1 ניתוב יחסי של /Emp/All ולשניה /Emp/ById

כמו בקטע הקוד הבא:

```
using Microsoft.AspNetCore.Mvc;
namespace RoutingInASPNETCoreWebAPI.Controllers
{
[ApiController]
public class EmployeeController : ControllerBase
{
[Route("Emp/All")]
public string GetAllEmployees()
{
return "Response from GetAllEmployees Method";
}
[Route("Emp/ById")]
public string GetEmployeeById()
{
return "Response from GetEmployeeById Method";
}
}
}
```

נריץ ונראה את התוצאה ל http://localhost/emp/all ולhttp://localhost/emp/byid

2. משתנים

כשנרצה ליצור ניתוב דינמי לפי חיתוך כלשהוא כמו מזהה וכדומה נשתמש במשתנים , צורת הכתיבה היא כזו {parameterName}. כלומר, בכדי להעביר ערכים למשתני הפונקציה נפתח סוגריים מסולסלות ובתוכם נרשום את שם המשתנה, כך הקליינט יידרש להעביר כחלק מה URL את הערך של הפרמטר הזה במקום הבלוק. טכנולוגית הASP ממפה את הערך לתוך המשתנה של הפונקציה.

לדוגמא

```
[Route("Employee/{Id}")]
public string GetEmployeeById(int Id)
{
return $"Return Employee Details : {Id};"
}
```

והניתוב כעת : http://localhost/Employee/5

ניתן להעביר מספר ערכים דינמיים לדוגמא

כל.הזכויות.שמורות

```
[Route("Employee/Gender/{Gender}/City/{CityId}")]
public string GetEmployeesByGenderAndCity(string Gender, int CityId)
{
return $"Return Employees with Gender : {Gender}, City: {CityId}";
}
```

נזכיר כי פרמטרים לפונקציה שלא נגדיר ב ROUTING ניתן להעביר בquery String

3. Token replacement

זהו פיצ'ר נוסף ש ASP .NET CORE נותנת. בו ניתן להחליף את השם של הcontroller או הaction דינמית. נרשום בניתוב [controller] וזה יוחלף עם השם של ה controller וכנ"ל ב[action]. כך במקרה שנשנה את שמותם לא נצטרך לתקן גם את הניתוב

אם נחזור לדוגמה הראשונה ונוסיף לפני הגדרת הcontroller :

```
[Route("[controller]")]
public class EmployeeController : ControllerBase
```

חסכנו לרשום את המילה employee בניתוב של כל action.

וכשנרשום לפני פונקציה GetAllEmployees:

```
[Route("[action]")]
public string GetAllEmployees()
```

בעת הניתוב יראה כך: http://localhost/employee/getallemployees

4. דריסה של הניתוב הבסיסי

כאשר נרשום בתחילת הניתוב ~ , זה אומר שהניתוב דורס את הניתוב הבסיסי שהוגדר במסגרת הcontroller .

לדוגמה :

```
[Route("[controller]")]
public class EmployeeController : ControllerBase

[Route("~/department/all")]
```

```
public string GetAllDepartment()
{
return "Response from GetAllDepartment Method";
}
```

ניגש אל http://localhost/department/all

ActionResult

ניתן להחזיר מה action מספר סוגים של ערכים:

type ספציפי כמו Int, string או כל אובייקט אחר. שימושי כאשר אין סיבה להחזיר סטטוס שונה מ200.

IActionResult אלו סוגים שונים לhttp status codes

לדוגמה: NotFound BadRequest CreatedAtAction

המחלקה controllerBase מגדירה מספר פונקציות שמחזירות סוגים של IActionResult

כמו הפונקציה ok(object) או notFound() ובקיצור new OkObjectResult(object)

כך נוכל להחזיר סטטוסים שונים וגם ערכים בפונקציה אחת

ActionResult<T> גם כאן ניתן להחזיר סטטוסים שונים אך בצורת הכתיבה מחזירים את האובייקט עצמו והוא הופך לactionResult.

1XX Informational Requests	100 Continue 101 Switching Protocols 102 Processing
2XX Successful Requests	200 OK 201 Created 202 Accepted 203 Non-Authoritative Information 204 No Content 205 Reset Content 206 Partial Content 207 Multi-Status 208 Already Reported
3XX Redirects	300 Multiple Choices 301 Moved Permanently 302 Found 303 See Other 304 Not Modified 305 Use Proxy 307 Temoprary Redirect 308 Permanent Redirect
4XX Client Errors	400 Bad Request 401 Unauthorized 402 Payment Required 403 Forbidden 404 Not Found 405 Method Not Allowed 407 Proxy Authentication Required 408 Request Timeout 409 Conflict 410 Gone 412 Precondition Failed 416 Request Range Not Satisfaible 417 Expectation Failed 422 Unprocessable Entity 423 Locked 424 Failed Dependency 426 Upgrade Required 429 Too Many Requests 431 Request Header Fileds Too Large 451 Unavailable for Legal Reasons
5XX Server Errors	500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable 504 Gateway Timeout 505 HTTP Version Not Supported 506 Variant Also Negotiates 507 Insufficient Storage 508 Loop Detected 510 Not Extended 511 Network Authentication Required

כל.הזכויות.שמורות