**בוחן 1 API:**

* דוטנט: שכבת תוכנה שבאמצעותה מקודדות ומורצות תוכנות אחרות

מספקת כלים ושירותים כמו ניהול זיכרון התקני קלט פלט ועוד

וכך המתכנתים לא צריכים לכתוב ספריות ויכולים לכתוב רק את הלוגיקה

* שפת תכנות מול Framework:

שפת תכנות- בו אנו כותבים למחשב מה לעשות רצות בסביבה ספציפית או מול מערכת ההפעלה

Framework- בנוי על גבי שפת כנות ומסייע לכתוב תוכנה לדג: אנגולר הוא Framework הבנוי ע"ג JS

מאפשר להתמקד בפונקציונליות ברמה גבוהה הרמה הנמוכה מטפלת על ידו

* דוטדט core- עד להוצאתו היו מגבלות רבות לדוטנט **1.** רק WINDOWS **2.** רישיונות יקרים **3.** זמני ריצה יקרים וכו'

ועכשיו **1.** הוסרה המגבלה של WINDOWS **2.** קוד פתוח **3.** שרת עצמאי **4.** צמצום של כמות הספריות הדרושות עם יכולות להוספה **5.** הזרת תלויות וכו'

* API- application programming interface אוסף של כלים ואפליקציות זה דרך נוספת של חברות לספק שירותים לדג' Google Maps
* Controller- מעיין שער לשרת מקבל פניות HTTP ומחזיר תשובה לפונה
* Windows Authentication- היישומים של ASP.NET נמצאים בIIS(internet information server)(31/2) לבדוק...
* HTTP- פרוטוקול שרת- לקוח , מגדיר איך יראה המידע שעובר בין השרת ללקוח

Http Request- בקשה מהלקוח לשרת מורכבת מהנונים: השיטה, הכתובת, גרסת פרוטוקול, התוכן.

Http Response- תשובה מהשרת ללקוח ומכילה: פרוטוקול התשובה, קוד המציין את התוצאה, מידע נוסף על התשובה ועל השרת

Status codes- 100- 200- הצלחה 300-

400 -בעיה בשליחה בלקוח 500- בעיה בקבלה בשרת

* Ajax- Asynchronous JavaScript and xml)) איך זה עובד:
* מתרחש אירוע כמו לחיצת כפתור
* נוצר אובייקט של XMLHttpRequest ע"י JS
* ה XMLHttpRequestשולח בקשה לשרת
* השרת מעבד את הבקשה
* השרת שולח את התשובה חזרה לדף האינטרנט
* התשובה נקראת ע"י JS
* המשך הנתונים נעשים ע"י JS כמו הצדת הנתונים על הדף
* jQuery- ספריית JS להקלת כתיבת סקריפטים לצד לקוח
* Json- פורמט טקסטואלי שקריא לאדם מיועד להעברת מבני נתונים, מורכב מזוגות של מפתח וערך. השימוש העיקרי זה להעברת מידע בין שרת ללקוח כתחליף לxml

אינו זהה לJS ,הוא אינו תלוי שפה ,בהרבה שפות תכנות יש המרה לפורמט JSON

שימוש בJSON בצד לקוח:

- שהמידע נשלח לשרת כדי להמיר מאובייקט JS לJSON **JSON.strigify()**

* המידע חוזר מהשרת כמחרוזת JSON ע"מ שנוכל להשתמש בתוך הקוד הופכים אותו לJS **JSON.parse()**
* Promise- אובייקט JS שניתן להחזיר באופן סינכרוני מפונקציה אסינכרוני

זהו הבטחה למשהו עתידי שיתרחש, הדפדפן מבטיח לחזור כשהפעולה האסינכרונית תסתיים וזה יכול להסתיים בהצלחה או בכישלון!.

Resolve()- ע"מ לסמן הצלחה לפעולה נקרא לה.

Reject()- ע"מ לסמן שהפעולה נכשלה נקרא לה. (כשקוראים לו מומלץ לזרוק EROR לזיהוי הבעיה בהמשך.

מצב pending()- אין עדיו תשובה.

* Fetch- דרך מודרנית ליצור פניות AJAX ,מחזירה PROMISE מה שמאפשר לשרשר .then להמשך טיפול מיד כשחוזר חומר מהשרת.
* REST- ארכיטקטורה למימוש שירותי רשת, תפיסת שרת-לקוח השרת מעבד את הפנייה ומחזיר תגובות מתאימות...

ממשקי API של REST הם שירותי האינטרנט הנפוצים היותר, מאפשר ללקוחות או אפליקציות דפדפן לתקשר עם שרת באמצעות REST API לכן חשוב לעצב זאת בצורה נכונה

מאפיינים:

* העברת נתונים בצורת JSON :

גם לצד שרת וגם לצד לקוח לרוב יש המרה לJSON אבל צריך לציין זאת בשליחה

* שימוש בשמות עצם ולא בפעלים בEnd Points: (להשלים...)

שיטות: GET- מביא נתונים מהשרת

POST- שולחת נתונים חדשים לשרת

PUT- מעדכן נתונים קיימים

DELETE- מסיר נתונים

* קינון לוגי בנקודות קצה:

כאשר מעצבים End Points הגיוני לקבץ מידע קשור אם אובייקט אחד יכול להכיל אובייקט אחר יש לעצב את נקודת הקצה באופן שמשקף את זה

* טיפול בשגיאות:

יש לטפל בשגיאות ולהחזיר קוד שגיעה סטנדרטיים של HTTP המציינים איזו שגיעה אירעה

* סינון, מיון וחלוקה לעמודים:

מאחורי ממשקי API של REST יכולים לעמוד מסדי נתונים ענקיים ויהיה כבד מאד להחזיר אותם בבת אחת. לשם כך נבצע בURL סינונים ע"י ? ואחריו התנאים

נתונים ששולחים כך מקבלים אותם ע"י FROMQUERY]]

* Model- למבנה נתונים

חיבור המודל לאובייקט JS :

1.התאמה מדויקת של שמות המשתנים

2. אובייקטים מורכבים משתמשים בFROMBODY]]

3. מוגדרים כPROPERTY

* שיתוף מידע בין דפי האתר:

1. ע"י פרמטרים window.location.href="namepage.html?="+
2. WEB STORAGE- נתונים שישמרו בצורה של מפתח-ערך בזיכרון הפנימי של הדפדפן

LOCAL...- ישמרו ללא הגבלת זמן אבל יכולה להימחק גם ע"י המשתמש

ייחודי לכל DOMAIN לא ניתן לשתף מידע בין אתרים

SESSION- כמו LOCAL רק מהנתונים ישמרו עד סגירת הדפדפן