**פרויקט לדוגמה באנגולר 2+ ו-WebAPI**

1. יש לקחת את כל קבצי הפרויקט, כולל תיקיית node modules על מנת לאפשר עבודה בניתוק מהרשת.
2. **שינוי** **שמות הפרויקטים** (ע"י rename) ואת קובץ ה- solutionלשם המתאים לפרויקט האישי שלכן:
   * אם שיניתן את שמות הפרויקטים (רצוי, לא חובה) שימו לב על כל פרויקט לחיצה על Properties בכרטיסיה הראשונה (Application) לשנות את שם ה-dll וכן את ה- default namespace.
   * כמו כן יש לשנות את קבצי ה-AssemblyInfo בכל פרויקט לשם החדש.
   * בתוך הקבצים של הקוד עצמו יש לשנות את כל השימושים ב-references (using...) לשם של ה-namespace החדש.
   * אם אתן משנות גם את שמות התיקיות (דרך folder explorer ולא דרך visual studio) יש לעשות הסרה לפרויקטים והוספה מחדש מהתיקיות החדשות.
   * אל תיבהלו משגיאות קומפילציה. לאחר השלמת התהליך יש לעשות clean ואז rebuild והשגיאות אמורות להיפתר.
   * לפעמים השגיאות הן כי ה-project references לא התעדכנו כמו שצריך (לחיצה ימנית על references בכל פרויקט -> Add בכרטיסיה השניה: Solution. ) במקרה כזה גם עוזר הרבה פעמים להוריד ולהוסיף מחדש את הפרויקטים.
3. **עדכון ה-edmx** למסד הנתונים הרצוי שלכן.
   * **שינוי ה-connection string :** עשו זאת רק על ידי שינוי ב-app.config של פרויקט ה-DAL. שנו ידנית לשם השרת ולשם הDB.
   * **עדכון הטבלאות במודל:** יש למחוק קודם את כל הטבלאות ורק אז לעשות Update Model from Database. אתן אמורות לקבל את רשימת הטבלאות ב-DB אליו קישרתן.
   * **שימו לב בפרויקט זה הפרדתי בין ה-**entities ל-model. זאת על מנת שכל השכבות (מלבד BL) לא יצטרכו reference לפרויקט הDAL, אלא ל-refernece המוקטן של Entities
4. **וזה המקום להסביר על השכבות בפרויקט**:
   * **Common –** מכיל פונקציונאליות שניתן להשתמש בה בכל שכבות הפרויקט (כמו פונקציה שמעגלת מספר לפי פרמטר כמה לעגל – הגיוני שנשתמש בה בכל שכבות הפרויקט). במקום זה ניתן לשים גם את כל ה-enums שנשתמש בהם בפרויקט.
   * **Entities –** מכיל את קובץ ה-tt של המודל (עם כל הישויות של ה-DB מתחתיו).  
     בפרויקט זה מוסיפים גם ישויות **Dto** שהם קלאסים מבניים שאינם קשורים ל-DB. (בדרך כלל משמשים להעברת נתונים מה-Client ל-Server, אך ניתן להשתמש בהם בכל הרמות.  
     אם רוצים להוסיף על אוביקט ה-entity שהגיע מה-DB (מג'ונרט בכל פעם שעושים update למודל) ניתן ליצור קלאסים בשם זהה עם קידומת **partial** בקובץ נפרד. אז המאפיינים של הקלאס הקשורים לישות אך לא שמורים ב-DB הם חלק בלתי נפרד מהקלאס, ולא נדרסים בכל עדכון המודל.
   * **DAL –** פונקציונאליות שמתקשרת ישירות מול ה-DB. כאן נשתמש ב-**DBContext** של המודל לבצע עדכונים ושליפות ממסד הנתונים.  
     נהוג ליצור קלאס של DAL עבור כל טבלה (או ישות) במסד הנתונים.  
     הפונקציות בקלאסים אלו בד"כ מוגדרות כ-static על מנת לחסוך הגדרת instances לקלאס בכל קריאה מה-Client.
   * **BL –** ה-Business Logic של הפרויקט. אם יש חישובים, אלגוריתמים או תהליכים אחרים מהנתונים שקיבלנו מה-client ולפני השמירה במסד הנתונים.  
     נהוג ליצור קלאס של BL עבור כל ישות DAL. ישות ה-BL היא היוזמת קריאות ל-DAL שיקרא ל-DB.  
     הפונקציות בקלאסים אלו בד"כ מוגדרות כ-static על מנת לחסוך הגדרת instances לקלאס בכל קריאה מה-Client.
   * **BPL –** שכבה שלא קימת בפרויקט הזה, אבל ניתן להוסיף אותה במקרה של שימוש בדף WebAPI אחד בכמה קלאסים של BL. על מנת שהקונטרולר של WebAPI לא יצטרך לנהל את הקריאות השונות ל-BL יוצרים את השכבה הזו – פונקציה שתקרא לכמה פונקציות בתוך שכבת הBL. ניתן לדלג על שכבה זו, ולבצע קריאות מ-BL אחד למשנהו. (כן להקפיד שהקונטרולר יקרא ל-BL אחד בלבד).
   * **WebAPI –** שכבת ה-Web שמתקשרת מול ה-Client (פרויקט האנגולר). בפרויקט זה יש מימוש רק של controllers מסוג WebAPI.
5. יש לשנות את ה-**Connection string** למסד הנתונים הרצוי גם בפרויקט ה-WebAPI וגם בפרויקט ה-Client.  
   (הקובץ **web.config** מופיע זהה בפרויקט ה-WebAPI ובפרויקט ה-Client – שימו לב לשינויים שיבוצעו בשני הקבצים).
6. על מנת שה-solution כולו יתקמפל יש לעשות unload לפרויקט ה-Client (שתעבדו איתו עם Visual studio Code). הוא חלק מה-solution רק למען הסדר הטוב.
7. להוספת ספרייה ב-Server יש לעמוד על ה-solution (ולא על פרויקט נפרד. זה מסבך) לחיצה ימנית -> Manage NuGet Packages for solution לבחור מתוך כרטיסיית Browse את הספרייה הרצויה. (כדאי קודם לחקור עליה ב-Git Hub שזה המקור האמין ביותר לספריות). לבחור את הפרויקטים שבהם רוצים להוסיף את הספרייה וללחוץ על install.   
   **שלב זה יכול להתבצע רק עם חיבור זמין לרשת**.

**פרויקט האנגולר (מתוך visual studio code)**

1. הפרויקט מכיל ספריות כלליות כמו JQuery, rxJs ועוד. (ראו packages.json)
2. ספריית הרכיבים בפרויקט היא priming. ניתן לשנות אותה בקובץ packages.json.
3. למי ששינתה את שמות הפרויקטים: יש לשים לב לשנות:
   * בקובץ ProjectTemplate\Client\e2e\app.e2e-spec.tsאת הכותרת. (**רצוי גם אם לא שיניתן את כל שמות הפרויקטים**)
   * בקובץ ה- AssemblyInfo
   * בקובץ package-lock.json את השם בתחילת הקובץ.
   * בקובץ package.json את השם בתחילת הקובץ.
4. שורש הפרויקט ב-view: ProjectTemplate\Client\src\index.htmlאשר מכיל את הכותרת שתתנו לכל האתר, אייקון אם רוצים, ואת הקומפוננטה הראשית (AppModule).
5. מתחת תיקיית environmnent יש הגדרה לשתי סביבות dev ו-prod אתן תעבדו כמובן על dev. (ProjectTemplate\Client\src\environments\environment.ts) שימו לב ל-port שאפשר לשנות. יש לשים את ה-port שהתקבל כאשר מריצים את פרויקט ה-WebAPI.
6. התיקיות המקובלות בפרויקט אנגולר:
   * **Main** – המודול הראשי של הפרויקט. תחתיו ייווצרו כל הקומפוננטות.
   * **Models** – הגדרת המודלים לשימוש בפרויקט (בדרך כלל מקביל ל-Entities ול-Dto שהוגדרו בשרת.
   * **Services** – כל השירותים מול הסרבר. נהוג לעיתים לשים גם לוגיקה כבדה (של client כמו בדיקות תקינות.) בפרויקט זה הלוגיקה נמצאת בעיקר בקומפוננטות עצמן וה-services משמשים להתקשרות לשרת בלבד.
   * **Shared** – **התשתית של המערכת**. רצוי מאד להשתמש בתיקייה זו. גם היא מחולקת לתיקיות:
     + Models – מבנים שהם בשימוש של כל המודולים (כולל מודול ההתחברות)
     + Services – שירותים שצריכים להיות זמינים בכל המערכת.   
       בפרויקט שלנו יש service תשתיתי:  
       ProjectTemplate\Client\src\app\shared\services\base-http.service.ts   
       וכן מודל Request תשתיתי (שכל הקלאסים מסוג services יורשים ממנו):  
       (**שימו לב יש לשלוח ל-super – הבונה של האבא את שם הקונטרולר ב-WebAPI שאליו הservice שלנו קורא. זאת על מנת לבנות תקין את ה-url.)**  
       הקלאס הזה הוא השירות היחידי שמנהל את קריאות ה-ajax. כל יתר ה-services בפרויקט רק מזמנים את הקריאה מהקלאס התשתיתי.
     + **Uix** – קומפוננטות תשתיתיות ומודלים תשתיתיים הקשורים ל-view (לדוגמה השארתי את המודל menu-item שישמש עבור כל שימוש ב-combo box
     + השארתי דוגמה לקומפוננטה בסיסית בתיקייה זו. אך היא לא בשימוש בפרויקט ה-template. **אל תשכחו למחוק אותה**!
   * עבור כל מודול שאינו קשור ישירות לפרויקט יש ליצור תיקיה חדשה, כמו בפרויקט שלנו המודול Auth (התחברות – לוגין). נהוג גם להפריד את הסרביסים והמודלים שלה בתוך התיקיות המתאימות.
7. בתוך המודול main מוגדרות תיקיות של models ו-services. יש להגדיר service בדרך כלל עבור כל קומפוננטה, ו-model עבור כל ישות.
8. כל service או model או component יש להוסיף לקובץ index.ts.  
   שימו לב: כרגע מכיון שלא היו services ב-main כלל, לא ניתן היה ליצור את קובץ ה-index של תיקיית main:  
   ProjectTemplate\Client\src\app\services\main\index.ts.

לאחר הוספת ה-services השונים לאינדקס יש להסיר את ההערות מקובץ זה, ולמלא את המערך בהתאם. כמו כן להסיר את ההערה מהמודול של כל ה-services:  
.ProjectTemplate\Client\src\app\services\services.module.ts  
כנ"ל בכל התיקיות שכרגע ריקות.

1. עיצוב: כרגע עיצוב הפרויקט לוקח מעיצוב לאתר קיים. לא הורדתי בכלל קלאסים של עיצוב. לכן יש להשגיח היטב אלו קלאסים רוצים למחוק ואלו להשאיר.  
   עיצוב כללי לפרויקט יהיה בקובץ:  
   ProjectTemplate\Client\src\styles.scss.  
   עיצוב עבור כל קומפוננטה בנפרד ייכתב בתיקייה של הקומפוננטה עצמה. לדוגמה:  
   ProjectTemplate\Client\src\app\+auth\+login\login.component.scss.  
   תמונות יישמרו בתיקייה:  
   ProjectTemplate\Client\src\assets
2. חבילת רכיבים:  
   כרגע הפרויקט משתמש בחבילת הרכיבים של PrimeNg בהחלט מומלץ שלא כולכן תשתמשנה בחבילה זו. ספריות נוספות מומלצות הן: AngularMaterial ו-Bootstrap.

על מנת להתקין ספרייה חדשה, כמו כן לנקות את הקובץ מספריות מותקנות שאינן רצויות יש לערוך את קובץ ה-packages.json עם הספריות (והגרסאות) הרצויות.  
על מנת להקין את הספרייה יש להריץ ב-Terminal: npm i.   
**פקודה זו חייבת להתבצע בעת חיבור זמין לרשת.**

1. על מנת להריץ את הפרויקט יש לכתוב: ng serve - -open. (שימו לב שיש להריץ במקביל (קודם) את פרויקט ה-Server.  
   על מנת לבנות את הפרויקט בלי להריץ יש לכתוב: ng build

**בהצלחה!**