**מבנה הפרויקט:**

הקוד מחולק ל2 חלקים, api ששומר ושולף מהדאטה ו-client שאחראי על הצגת המידע למשתמש.

בכתיבת הקוד השתדלתי לשים דגש על מודולריות מרבית ככל שניתן תוך איסוף ושמירה בנפרד של כל מה שניתן.

Api מחולק לתיקיות הבאים:

1. Models – מה נשמר במסד הנתונים. המודלים מייצגים את אופן השמירה של הנתונים והקשרים ביניהם. לכל ישות שונה בתכנית שצריכה להישמר, ישנו מודל ייחודי.
2. Data – מחזיק בתוכו DbContext שאחראי על העבודה מול מסד הנתונים וקביעת הקשרים בין המודלים השונים במסד הנתונים.
3. Enums – כל הenums שקיימים בתוכנית.
4. Utils – class-ים סטטיסטיים שמבצעים פעילויות חישוביות כלשהן. הפעולות לא קשורות למשהו ספציפי אלא מהוות אלגוריתמים כלליים ולפיכך מצויים בנפרד.
5. Dto - data transfer object – מחזיק את כל הנתונים שצריכים לעבור בין הישויות השונות בתוכנית (rast-api, MVC, server) שאינם מודלים ומיועדים למטרת העברת מידע בלבד.
6. Service – אחראי על כל הלוגיקה שקשורה למודלים. על שליפת המידע ממסד הנתונים ועל השמירה שלו בדאטה. כמו כן, הוא אחראי על בניית האובייקטים לשליחה ואת כלל הבדיקות הנדרשות ומהווה את מרכז העצבים של הפרויקט ונקודת המפגש בין כלל הישויות השונות. ה Service-ים מחולקים לפי המודלים שקיימים בתוכנית כשהמטרה היא שכל קובץ יתכלל את כל הבקשות הקשורות למודל הספציפי תוך שאיפה ששאר הקבצים אף בתיקיה זו יימנעו מלגשת אליו וישתמשו בקובץ הייחודי שלו על מנת לקבל נתונים הקשורים למודל זה.
7. Controller – אחראי על הקשר עם "העולם החיצון" שמחוץ לאפליקציה הספציפית. דרכו מתקבלות הבקשות לקבלת נתונים, עדכון הנתונים ולהוספת נתונים חדשים. גם בתיקיה זו הקבצים מחולקים לפי נושאים כך שלכל נושא יש קובץ משלו ששולח בקשות לקובץ הService הייחודי.

Client מחולק גם הוא לתיקיות דומות תוך הקפדה שהחלוקה תישמר לפי המודלים הקיימים בApi ככל הניתן. ובאפליקציה זו ישנה תיקיה נוספת:

1. View-model – אחראי להצגת המודלים השונים ועוד נתונים שונים שיש להציג בView.

**תכנון הקוד**

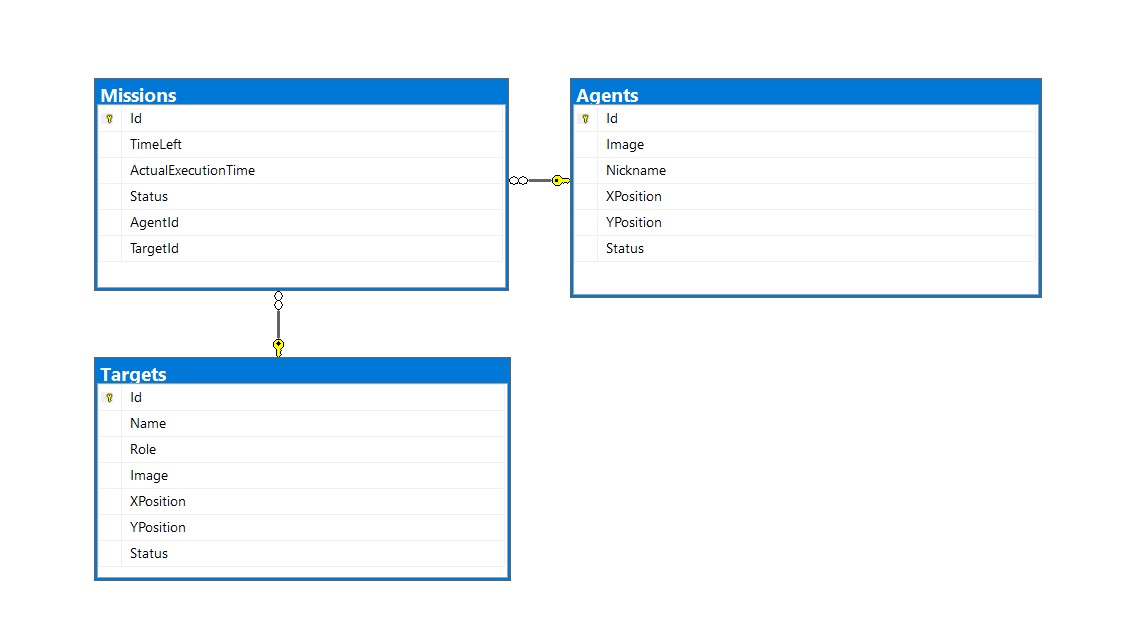
הקוד בנוי כך שלכל מודל יש Controller, וService משלו, כאשר Controller כמעט ולא עושה דבר למעט בקשות לCervice והצגתם למשתמש או שליחתם למבקש. רוב הלוגיקה נעשית בService והוא מחולק לפונקציות שבחלקן הן פונקציות עזר לService עצמו והיתר הינן פונקציות שנועדו לספק ל Controller בדיוק את מה שהוא צריך בכל פונקציה, כך של Controller נותר רק לשלוח או להציג את המידע.

רוב ככל הקוד בנוי על אובייקטים שעוטפים את הנתונים כאשר בכך מוודאים שהקוד נשמר פתוח לשינויים עתידיים והשינויים יידרשו במספר מינימלי של מקומות. כך גם כל משימה מוגדרת בדרך כלל נמצאת בפונקציה משל עצמה על מנת לשמר את האפשרות להתאמות בצורה מסודרת בעתיד מבלי להיאלץ לעשות שינויים בשאר חלקי הקוד.

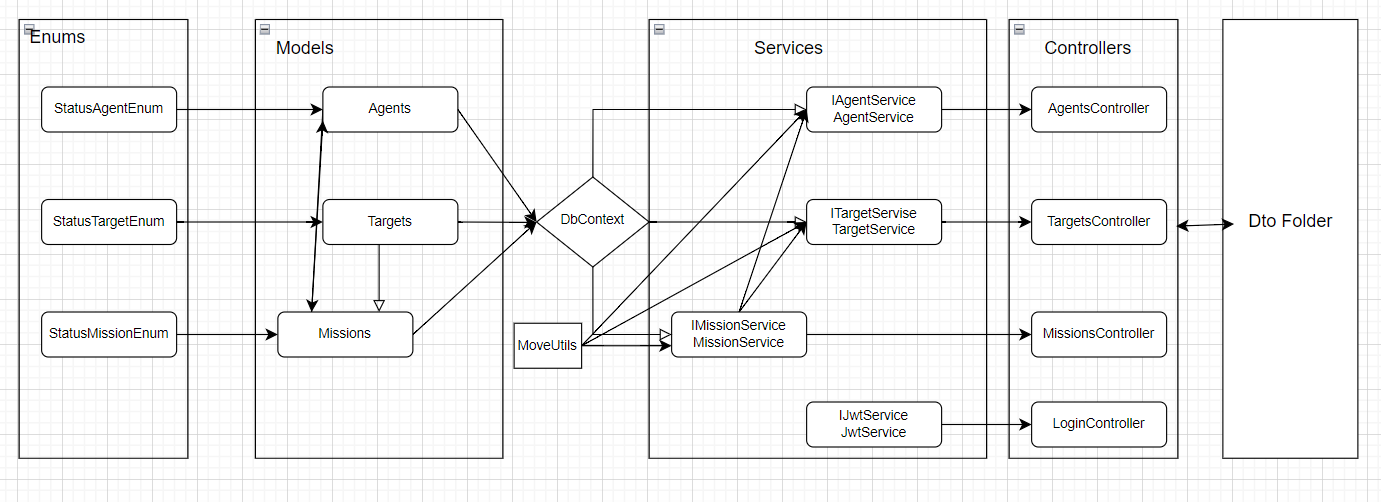
כל המשימות שנוגעות למסדי נתונים הינן אסינכרוניות ומתבצעות במקביל על מנת שלא לעכב את המשתמש בעוד המערכת מכניסה נתונים או מושכת את חלקם.

כל הסרוויסים מופעלים באמצעות DI, כך גם הסינגלטונים.

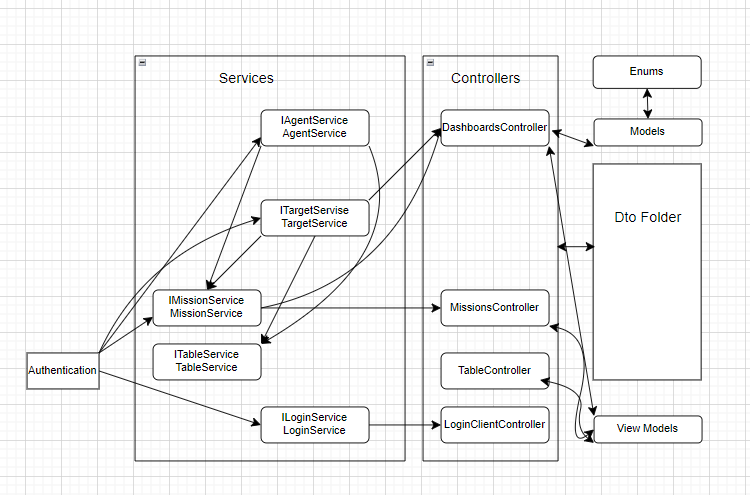
**מבנה המודלים והשמירה במסד הנתונים:**

****

**מבנה Rest-api**

****

**מבנה Client**

****

Models וenums זהים ולכן פורטו רק בapi

**בעיות שנתקלתי בהן במהלך הפרויקט והפתרונות:**

1. כאשר צריך להעביר את הagents מrast לclient, יש צורך להעביר כל סוכן עם המשימות שלו, כדי שניתן יהיה לראות את הסטטיסטיקות של הסוכן והמשימות שהוא ביצע, אבל אם עושים include למשימות, נוצר include מעגלי בו בכל סוכן מאותחלות כל המשימות ובתוך כל משימה מאותחל הסוכן וכו׳.

הפיתרון, בהכוונת המרצה וגוגל: השתמשתי ב asNoTracking בשביל לוודא את הבאת הנתון הבסיסי של הסוכנים. לאחר מכן הבאת כל המשימות ולחבר ביניהם באמצעות query כך שלא מתבצע איתחול מעגלי כלל.

1. שמירת הtoken המתקבל מהrest . ניתן לשמור אותו בsession, רק מתוך הcontroller, מה שהיה יוצר מגוון בעיות לאור השלב המאוחר בו קיבלנו את השרת עם הטוקנים לשלוח לכל פונקציה ופונקציה את הטוקן.

הפיתרון: יצרתי סינגלטון שמחזיק בטוקן כשבשימוש בDI, הוא מוזרק לכל סרוויס שזקוק לו.

1. ביצירת טבלה עבור המיקומים (טבלת x,y), היה מסובך מידי פשמור את זה כמו שצריך ולקבל את המיקום בנוחות וביעילות.

הפיתרון בהכוונת המרצה: שמתי אותם כעוד משתנים בסוכנים ובמטרות.

1. הדפדפן קרס בכל פעם שעלתה הטבלה שמציגה את כל הסוכנים והמטרות, מחמת גדלה (1000\*1000).

הפיתרון: יצרתי את הטבלה עד המיקום המקסימלי שגדול ביותר של המטרות והסוכנים.